Database Systems Development

M21269 - Group 29

CW Group contribution statement

	UP2109066	UP2118085	UP2246765
ERD Design	☑	☑	☑
Assumptions	☑	☑	☑
Data Dictionary	☑	☑	☑
Coding (CREATE+INSERT)	☑	☑	☑
Queries	☑	☑	☑
Theoretical Aspects	☑	☑	☑
Document writing	☑	☑	☑
Brainstorming	☑	☑	☑
DDT	☑	☑	☑
Percentage Allocation	**33.3%**	**33.3%**	**33.3%**
Signature	**B.S**	**Israr**	**Osar**

Database Systems Development
M21269

Group Number: [29]

Database Development Tracker

Date	Task Description	Member ID	Task Details	Time	Signature
11/10/2024	Initial meeting	UP2109066 UP2118085 UP2246765	A call on discord to go through the mark scheme together. A shared lucidchart document was created and we listed all the potential tables, connections and how the tables should be connected in order to understand the structure of the business and how it would reflect within the database before designing the ERD.	1H	**B.S** Israr Osar
14/10/2024	ERD first draft	UP2109066 UP2118085 UP2246765	We all dedicated an hour to sit on a discord call and create a first draft of the ERD.	1H	**B.S** Israr Osar
18/10/2024	ERD second draft	UP2109066 UP2118085 UP2246765	A discord call occurred taking 3 hours to take the first draft and expand it more. Ensuring all normalisation rules are met and that the logic of the database makes sense. After 3 hours we were happy with the ERD.	3H	**B.S** Israr Osar
24/10/2024	Feedback	UP2109066 UP2246765	UP2246765 and UP2109066 took the ERD to their lab and showed it to Val for feedback with the main takeaways being to avoid overlapping lines and clearing up some constraints and attributes	1H	**B.S** Osar
24/10/2024	Applying feedback	UP2109066 UP2118085 UP2246765	In the afternoon we agreed to call to discuss the feedback and apply the changes discussed as well as make the data dictionary.	1H	**B.S** Israr Osar
28/10/2024	Creating the Physical Side	UP2109066 UP2118085 UP2246765	We called and took the ERD we are happy with and split it into three to distribute the workload for creating the database through the code.	1H	**B.S** Israr Osar
30/10/2024	Updating our tasks	UP2109066 UP2118085	We called on discord to share our code, put it together and tested it to see if it worked. After a tiny bit of tweaking the creation of the database is	1H	**B.S** Israr

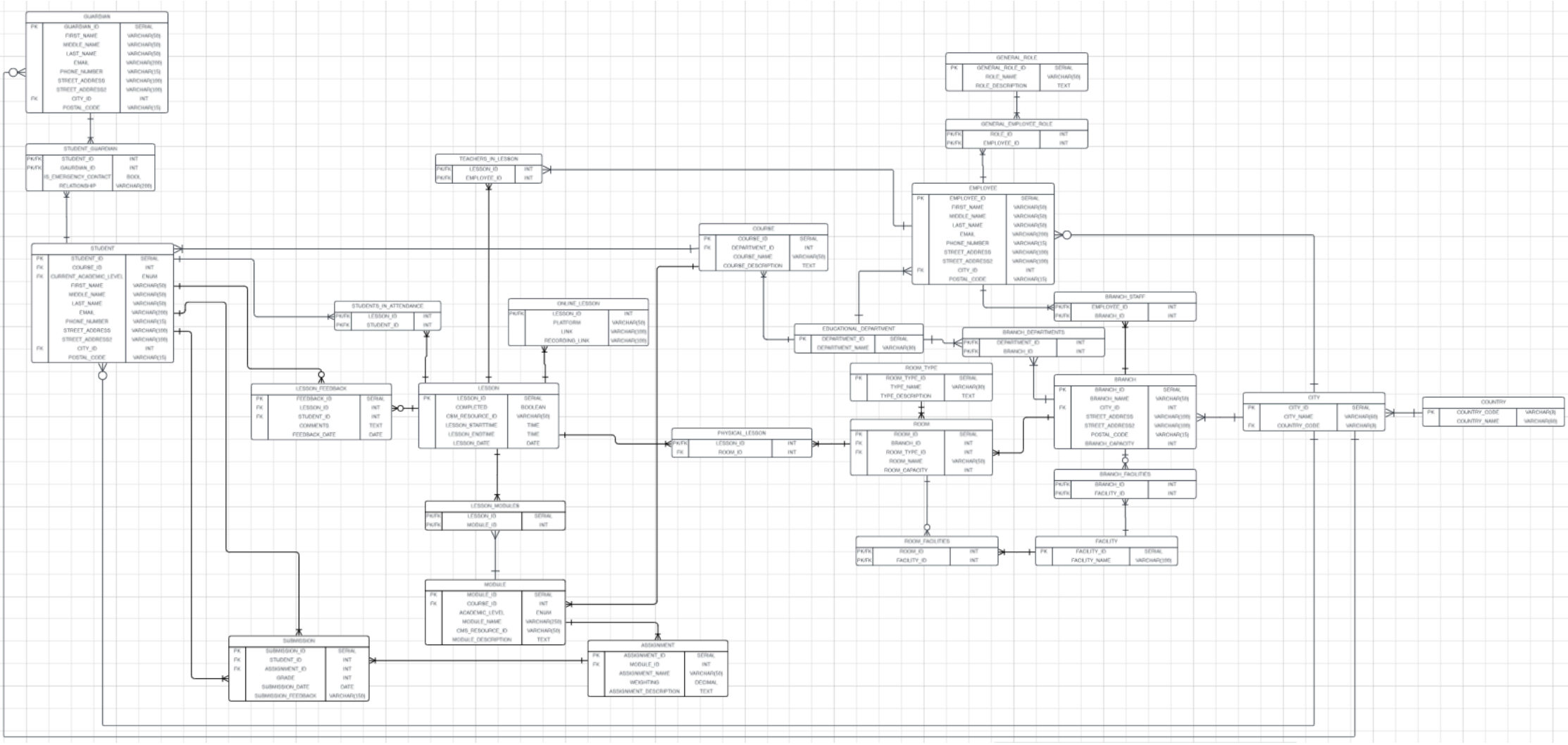
		UP2246765	done. We went over what is left to do, which is, making the inserts and making the queries. We split up the tables and split making inserts for these tables and broke down how many inserts we believe each should have. The numbers were put on the shared lucid chart for all three of us to see and see if we're done.		Osar
19/11/2024	Re-doing the inserts	UP2109066 UP2118085 UP2246765	After regular messaging on the discord and updating each other, most of the inserts were done until the submission and assignment table. Upon trying to make it, UP2109066 realised that the number of assignments and submissions would be in the upper thousands with the way we structured our numbers for the inserts e.g. 200 students. So we had an emergency meeting where we went over the numbers again to make sure it was doable. This second draft of the numbers was then put in LucidChart too.	1H	**B.S** Israr Osar
28/11/2024	Checking the numbers	UP2246765	In order to not make the mistake of creating a bunch of inserts that are too much, UP2246765 asked Val to go over our numbers to see whether it is more appropriate than previously. Junior then let us know in the discord that Val approved of our numbers with a few changes like increasing the students from 50 to 100.	1H	Osar
30/11/2024	Quick call to double check	UP2109066 UP2118085 UP2246765	We had a quick call to double check everyone knew what they were doing, going over the numbers again and redistributing the work equally.	0.5H	**B.S** Israr Osar
05/12/2024	Moving on from inserts	UP2109066 UP2118085 UP2246765	With updates through the week, we had all finished our inserts so we got on call to see if they all worked. Once we had it working and were sure it was all good we decided to move on to discussing the next tasks which were the 5 queries then the 3 paragraphs. We split the 3 paragraphs with each person doing one and decided on what we wanted the queries to be before assigning 2 queries to UP2118085, 2 to UP2246765 and 3 to UP2109066.	0.5H	**B.S** Israr Osar
08/12/2024	Done with queries	UP2109066 UP2118085 UP2246765	We all called again in discord to show our queries. After discussing them and how we could improve them we decided to move on to the theoretical discussions. For three separate topics we did one for each person: -2109066: Security	1H	**B.S** Israr Osar

GROUP 29

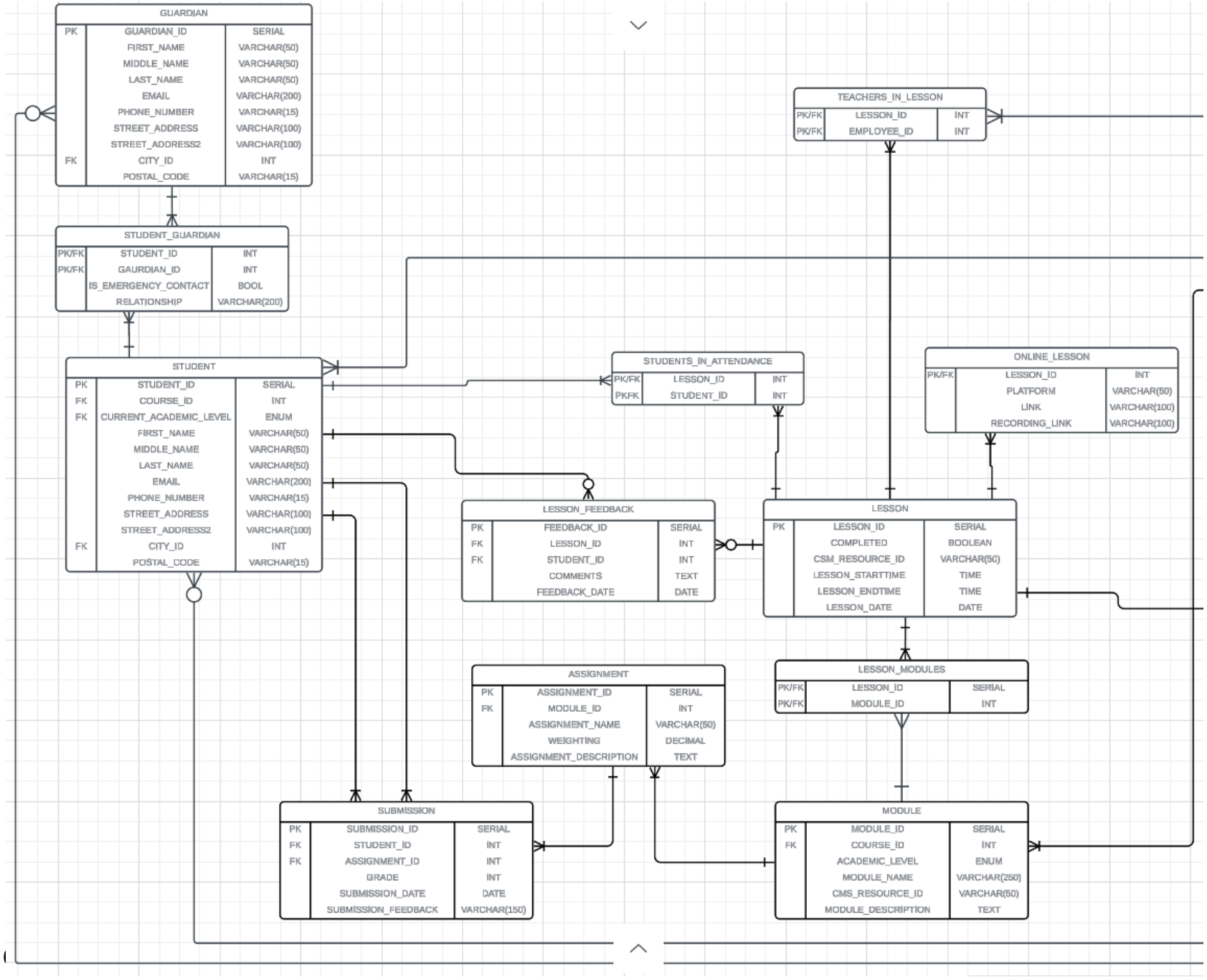
			-2246765: Optimisation -2118085: Legal and Ethical Considerations		
12/12/2024	Finishing Off	UP2109066 UP2118085 UP2246765	Called one last time to review our paragraphs and our work in general to check everything is covered and we are happy with everything. After briefly reviewing our work we decided we are happy with what we have.	1H	**B.S** Israr Osar

GROUP 29

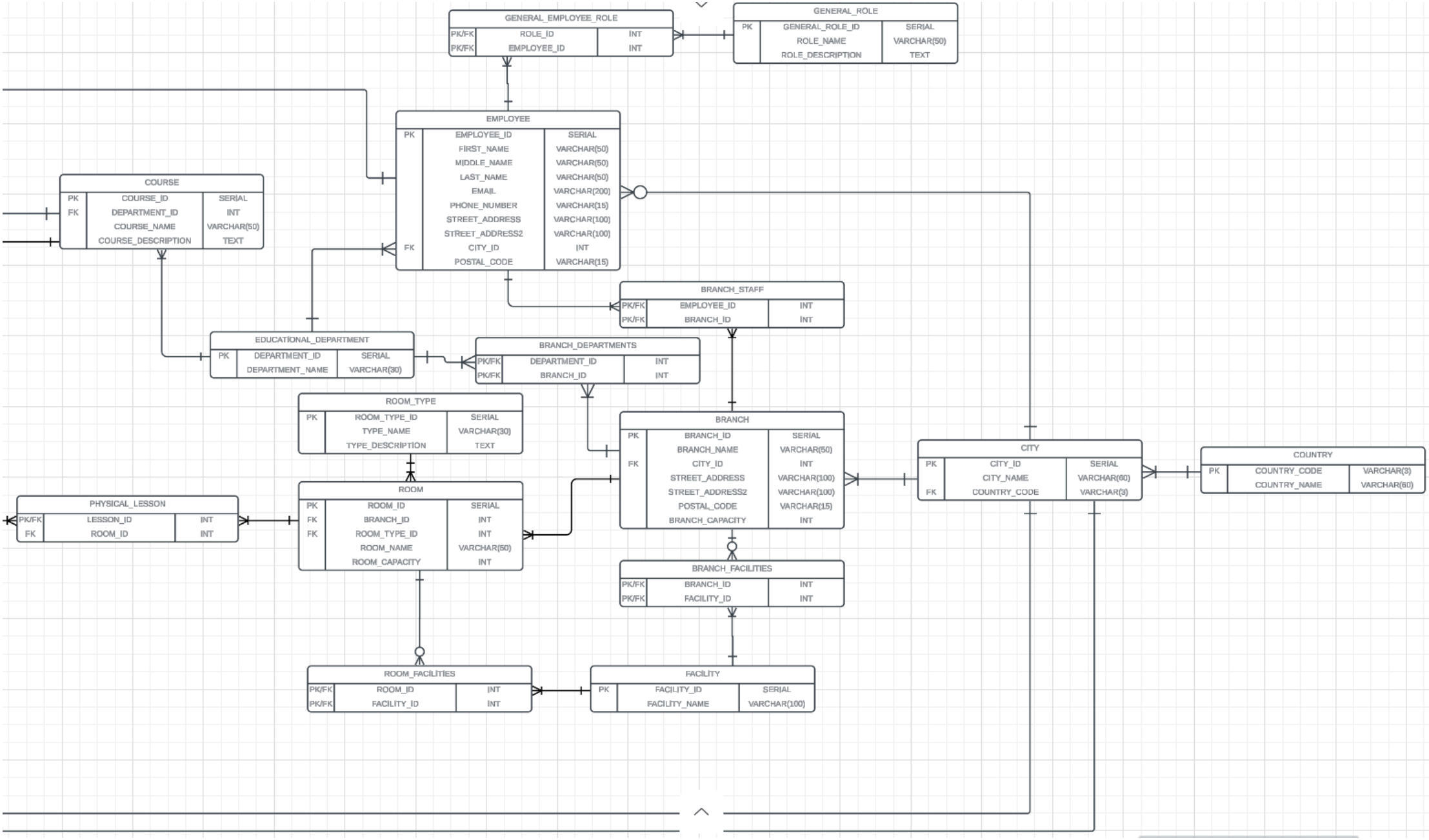
ERD


GUARDIAN
GUARDIAN_ID SERIAL
FIRST_NAME VARCHAR(50)
MIDDLE_NAME VARCHAR(50)
LAST_NAME VARCHAR(50)
EMAIL VARCHAR(200)
PHONE_NUMBER VARCHAR(15)
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
CITY_ID INT
POSTAL_CODE VARCHAR(15)
STUDENT_GUARDIAN
STUDENT_ID INT
GAURDIAN_ID INT
IS_EMERGENCY_CONTACT BOOL
RELATIONSHIP VARCHAR(200)
STUDENT
STUDENT_ID SERIAL
COURSE_ID INT
CURRENT_ACADEMIC_LEVEL ENUM
FIRST_NAME VARCHAR(50)
MIDDLE_NAME VARCHAR(50)
LAST_NAME VARCHAR(50)
EMAIL VARCHAR(200)
PHONE_NUMBER VARCHAR(15)
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
CITY_ID INT
POSTAL_CODE VARCHAR(15)
TEACHERS_IN_LESSON
LESSON_ID INT
EMPLOYEE_ID INT
STUDENTS_IN_ATTENDANCE
LESSON_ID INT
STUDENT_ID INT
ONLINE_LESSON
LESSON_ID INT
PLATFORM VARCHAR(50)
LINK VARCHAR(255)
RECORDING_LINK VARCHAR(255)
LESSON_FEEDBACK
FEEDBACK_ID SERIAL
LESSON_ID INT
STUDENT_ID INT
COMMENTS TEXT
FEEDBACK_DATE DATE
LESSON
LESSON_ID SERIAL
COMPLETED BOOLEAN
CMS_RESOURCE_ID VARCHAR(50)
LESSON_STARTTIME TIME
LESSON_ENDTIME TIME
LESSON_DATE DATE
LESSON_MODULES
LESSON_ID SERIAL
MODULE_ID INT
MODULE
MODULE_ID SERIAL
COURSE_ID INT
ACADEMIC_LEVEL ENUM
MODULE_NAME VARCHAR(255)
CMS_RESOURCE_ID VARCHAR(50)
MODULE_DESCRIPTION TEXT
SUBMISSION
SUBMISSION_ID SERIAL
STUDENT_ID INT
ASSIGNMENT_ID INT
GRADE INT
SUBMISSION_DATE DATE
SUBMISSION_FEEDBACK VARCHAR(150)
ASSIGNMENT
ASSIGNMENT_ID SERIAL
MODULE_ID INT
ASSIGNMENT_NAME VARCHAR(50)
WEIGHTING DECIMAL
ASSIGNMENT_DESCRIPTION TEXT
COURSE
COURSE_ID SERIAL
DEPARTMENT_ID INT
COURSE_NAME VARCHAR(50)
COURSE_DESCRIPTION TEXT
GENERAL_ROLE
GENERAL_ROLE_ID SERIAL
ROLE_NAME VARCHAR(50)
ROLE_DESCRIPTION TEXT
GENERAL_EMPLOYEE_ROLE
ROLE_ID INT
EMPLOYEE_ID INT
EMPLOYEE
EMPLOYEE_ID SERIAL
FIRST_NAME VARCHAR(50)
MIDDLE_NAME VARCHAR(50)
LAST_NAME VARCHAR(50)
EMAIL VARCHAR(200)
PHONE_NUMBER VARCHAR(15)
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
CITY_ID INT
POSTAL_CODE VARCHAR(15)
EDUCATIONAL_DEPARTMENT
DEPARTMENT_ID SERIAL
DEPARTMENT_NAME VARCHAR(30)
BRANCH_STAFF
EMPLOYEE_ID INT
BRANCH_ID INT
BRANCH_DEPARTMENTS
DEPARTMENT_ID INT
BRANCH_ID INT
ROOM_TYPE
ROOM_TYPE_ID SERIAL
TYPE_NAME VARCHAR(30)
TYPE_DESCRIPTION TEXT
PHYSICAL_LESSON
LESSON_ID INT
ROOM_ID INT
ROOM
ROOM_ID SERIAL
BRANCH_ID INT
ROOM_TYPE_ID INT
ROOM_NAME VARCHAR(50)
ROOM_CAPACITY INT
BRANCH
BRANCH_ID SERIAL
BRANCH_NAME VARCHAR(50)
CITY_ID INT
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
POSTAL_CODE VARCHAR(15)
BRANCH_CAPACITY INT
BRANCH_FACILITIES
BRANCH_ID INT
FACILITY_ID INT
ROOM_FACILITIES
ROOM_ID INT
FACILITY_ID INT
FACILITY
FACILITY_ID SERIAL
FACILITY_NAME VARCHAR(100)
CITY
CITY_ID SERIAL
CITY_NAME VARCHAR(60)
COUNTRY_CODE VARCHAR(3)
COUNTRY
COUNTRY_CODE VARCHAR(3)
COUNTRY_NAME VARCHAR(60)

GROUP 29


GUARDIAN
PK GUARDIAN_ID SERIAL
FIRST_NAME VARCHAR(50)
MIDDLE_NAME VARCHAR(50)
LAST_NAME VARCHAR(50)
EMAIL VARCHAR(200)
PHONE_NUMBER VARCHAR(15)
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
FK CITY_ID INT
POSTAL_CODE VARCHAR(15)
TEACHERS_IN_LESSON
PK/FK LESSON_ID INT
PK/FK EMPLOYEE_ID INT
STUDENT_GUARDIAN
PK/FK STUDENT_ID INT
PK/FK GAURDIAN_ID INT
IS_EMERGENCY_CONTACT BOOL
RELATIONSHIP VARCHAR(200)
STUDENT
PK STUDENT_ID SERIAL
FK COURSE_ID INT
FK CURRENT_ACADEMIC_LEVEL ENUM
FIRST_NAME VARCHAR(50)
MIDDLE_NAME VARCHAR(50)
LAST_NAME VARCHAR(50)
EMAIL VARCHAR(200)
PHONE_NUMBER VARCHAR(15)
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
FK CITY_ID INT
POSTAL_CODE VARCHAR(15)
STUDENTS_IN_ATTENDANCE
PK/FK LESSON_ID INT
PKFK STUDENT_ID INT
ONLINE_LESSON
PK/FK LESSON_ID INT
PLATFORM VARCHAR(50)
LINK VARCHAR(100)
RECORDING_LINK VARCHAR(100)
LESSON_FEEDBACK
PK FEEDBACK_ID SERIAL
FK LESSON_ID INT
FK STUDENT_ID INT
COMMENTS TEXT
FEEDBACK_DATE DATE
LESSON
PK LESSON_ID SERIAL
COMPLETED BOOLEAN
CSM_RESOURCE_ID VARCHAR(50)
LESSON_STARTTIME TIME
LESSON_ENDTIME TIME
LESSON_DATE DATE
ASSIGNMENT
PK ASSIGNMENT_ID SERIAL
FK MODULE_ID INT
ASSIGNMENT_NAME VARCHAR(50)
WEIGHTING DECIMAL
ASSIGNMENT_DESCRIPTION TEXT
LESSON_MODULES
PK/FK LESSON_ID SERIAL
PK/FK MODULE_ID INT
SUBMISSION
PK SUBMISSION_ID SERIAL
FK STUDENT_ID INT
FK ASSIGNMENT_ID INT
GRADE INT
SUBMISSION_DATE DATE
SUBMISSION_FEEDBACK VARCHAR(150)
MODULE
PK MODULE_ID SERIAL
FK COURSE_ID INT
ACADEMIC_LEVEL ENUM
MODULE_NAME VARCHAR(250)
CMS_RESOURCE_ID VARCHAR(50)
MODULE_DESCRIPTION TEXT

GROUP 29


GENERAL_EMPLOYEE_ROLE
PK/FK ROLE_ID INT
PK/FK EMPLOYEE_ID INT
GENERAL_ROLE
PK GENERAL_ROLE_ID SERIAL
ROLE_NAME VARCHAR(50)
ROLE_DESCRIPTION TEXT
EMPLOYEE
PK EMPLOYEE_ID SERIAL
FIRST_NAME VARCHAR(50)
MIDDLE_NAME VARCHAR(50)
LAST_NAME VARCHAR(50)
EMAIL VARCHAR(200)
PHONE_NUMBER VARCHAR(15)
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
FK CITY_ID INT
POSTAL_CODE VARCHAR(15)
COURSE
PK COURSE_ID SERIAL
FK DEPARTMENT_ID INT
COURSE_NAME VARCHAR(50)
COURSE_DESCRIPTION TEXT
BRANCH_STAFF
PK/FK EMPLOYEE_ID INT
PK/FK BRANCH_ID INT
EDUCATIONAL_DEPARTMENT
PK DEPARTMENT_ID SERIAL
DEPARTMENT_NAME VARCHAR(30)
BRANCH_DEPARTMENTS
PK/FK DEPARTMENT_ID INT
PK/FK BRANCH_ID INT
ROOM_TYPE
PK ROOM_TYPE_ID SERIAL
TYPE_NAME VARCHAR(30)
TYPE_DESCRIPTION TEXT
BRANCH
PK BRANCH_ID SERIAL
BRANCH_NAME VARCHAR(50)
FK CITY_ID INT
STREET_ADDRESS VARCHAR(100)
STREET_ADDRESS2 VARCHAR(100)
POSTAL_CODE VARCHAR(15)
BRANCH_CAPACITY INT
CITY
PK CITY_ID SERIAL
CITY_NAME VARCHAR(60)
FK COUNTRY_CODE VARCHAR(3)
COUNTRY
PK COUNTRY_CODE VARCHAR(3)
COUNTRY_NAME VARCHAR(60)
PHYSICAL_LESSON
PK/FK LESSON_ID INT
FK ROOM_ID INT
ROOM
PK ROOM_ID SERIAL
FK BRANCH_ID INT
FK ROOM_TYPE_ID INT
ROOM_NAME VARCHAR(50)
ROOM_CAPACITY INT
BRANCH_FACILITIES
PK/FK BRANCH_ID INT
PK/FK FACILITY_ID INT
ROOM_FACILITIES
PK/FK ROOM_ID INT
PK/FK FACILITY_ID INT
FACILITY
PK FACILITY_ID SERIAL
FACILITY_NAME VARCHAR(100)

GROUP 29
Link:
https://lucid.app/lucidchart/4eaa2f22-156c-4f87-b89d-99b3b3f63b21/edit?viewport_loc=22600%2C-2258%2C3357%2C1461%2C0_0&invitationId=inv_2eb3c3e2-4865-4215-94f0-faa70327dfd2

Assumptions

1. There is a possibility that an employee can be moved around branches, as a result an intersection table (branch_staff) is needed between employee and branch to address the many to many relationship.

2. There is a possibility that an employee can have more than one role, for example: A teacher can also be a branch manager. An intersection table (general_employee_role) is needed between general_role and employee in order to address the many to many relationship.

3. All online lessons are recorded therefore the link to access the lesson as well as the recording link needs to be stored.

4. Both branches and rooms can have facilities. E.g branch has computer lab facility, room 29 of branch 3 is a computer lab therefore has that facility

5. Multiple modules are covered within a lesson session due to the limited amount of lessons within a year, as a result there is a many to many relationship between lesson and modules, solved by lesson_modules intersection table.

6. Globally postcodes are longer than 8 characters and can be null too (Mockaroo provides some postcodes as either null or longer than 8 characters)

7. Each branch covers multiple educational departments, as a result there will be a many to many relationship between branch and educational department, addressed through branch_departments

GROUP 29

8. Students may have guardians that aren't their parents, as a result, a relationship attribute must be defined as well as an intersection table between guardian and student called student_guardian. The relationship attribute would be within this table.

-

Data Dictionary

Country					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Country_Code	PK	Varchar(3)	NOT NULL		
Country_Name		Varchar(60)	NOT NULL		

City					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
City_ID	PK	SERIAL	NOT NULL		
City_Name		VARCHAR(60)	NOT NULL		
Country_Code		VARCHAR(3)	NOT NULL		

GROUP 29

employee					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
employee_ID	PK	Serial	NOT NULL		Staff identifier
firast_name		VARCHAR(50)	NOT NULL		
last_name		VARCHAR(50)	NOT NULL		
email		VARCHAR(200)	NOT NULL, Unique		
Phone_Number		VARCHAR(15)	NOT NULL		
street_adress		Varchar(100)	NOT NULL		
street_adress2		Varchar(100)			
city_ID	FK	INT	NOT NULL	CITY. City_ID	
postal_code		Varchar(15)			

GENERAL_ROLE						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
GENERAL_ROLE_ID	PK		SERIAL			
ROLE_NAME			VARCHAR(50)	NOT NULL		
ROLE_DESCRIPTION			(TEXT)			

GROUP 29

GENERAL_EMPLOYEE_ROLE					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
ROLE_ID	PK/FK	(INT)	NOT NULL	GENERAL_ROLE.ROLE_ID	
EMPLOYEE_ID	PK/FK	(INT)	NOT NULL	EMPLOYEE.EMPLOYEE_ID	

BRANCH						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
BRANCH_ID	PK		SERIAL			
BRANCH_NAME			VARCHAR(50)	NOT NULL		
CITY_ID	FK		(INT)	NOT NULL	CITY.CITY_ID	
STREET_ADDRESS			VARCHAR(100)	NOT NULL		
STREET_ADDRESS2			VARCHAR(100)			
POSTAL_CODE			VARCHAR(15)			
BRANCH_CAPACITY			(INT)			

BRANCH_STAFF						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
EMPLOYEE_ID	PK/FK		(INT)	NOT NULL	EMPLOYEE.EMPLOYEE_ID	

GROUP 29

BRANCH_ID	PK/FK		(INT)	NOT NULL	BRANCH.BRANCH_ID	

FACILITY						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
facility_id	PK		SERIAL			
facility_name			VARCHAR(100)	NOT NULL		

BRANCH_FACILITIES						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
branch_id	PK/FK		(INT)	NOT NULL	BRANCH.branch_id	
facility_id	PK/FK		(INT)	NOT NULL	FACILITY.facility_id	

EDUCATIONAL_DEPARTMENT						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
department_id	PK		SERIAL			
department_name			VARCHAR(30)			

BRANCH_DEPARTMENTS

GROUP 29

Attribute_Name	*KEY*	*INDEX*	*Data Type & Size*	*Domains & Constraints*	*FK Reference*	*Description*
department_id	PK/FK		(INT)	NOT NULL	DEPARTMENT.department_id	
branch_id	PK/FK		(INT)	NOT NULL	BRANCH.branch_id	

ROOM_TYPE						
Attribute_Name	*KEY*	*INDEX*	*Data Type & Size*	*Domains & Constraints*	*FK Reference*	*Description*
room_type_id	PK		SERIAL			
type_name			VARCHAR(30)	NOT NULL		
type_description			(TEXT)			

ROOM						
Attribute_Name	*KEY*	*INDEX*	*Data Type & Size*	*Domains & Constraints*	*FK Reference*	*Description*
room_id	PK		SERIAL			
branch_id	FK		(INT)	NOT NULL	BRANCH.branch_id	
room_type_id	FK		(INT)	NOT NULL	ROOM_TYPE.room_type_id	
room_name			VARCHAR(50)	NOT NULL		
room_capacity			(INT)	NOT NULL		

GROUP 29

ROOM_FACILITIES						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
room_id	PK/FK		(INT)	NOT NULL	ROOM.room_id	
facility_id	PK/FK		(INT)	NOT NULL	FACILITY.facility_id	

COURSE						
Attribute_Name	***KEY***	***INDEX***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
course_id	PK		SERIAL			
department_id	FK		(INT)	NOT NULL	DEPARTMENT.department_id	
course_name			VARCHAR(50)			
course_description			(TEXT)			

STUDENTS					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Studnet_ID	PK	SERIAL	NOT NULL		
Course_ID	FK	INT	NOT NULL	Course. Course_ID	
current_academic_level		ENUM	NOT NULL		

first_name		Varchar(50)	NOT NULL		
last_name		VARCHAR(50)	NOT NULL		
Email		Varchar(200)	NOT NULL, Unique		
phone_number		Varchar(15)	NOT NULL , Unique		
street_adress		Varchar(100)	NOT NULL		
street_adress2		Varchar(1000			
City_ID		INT	NOT NULL		
postal_code		Varchar(15)			

Guardian					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Guardian_id	PK	SERIAL	NOT NULL		
first_name		Varchar(50)	NOT NULL		
last_name		Varchar(50)	NOT NULL		
relationship		Varchar(200)	NOT NULL		
Email		Varchar(200)	NOT NULL, UNIQUE		
phone_number		Varchar(15)	NOT NULL, UNIQUE		
street_adress		Varchar(100)	NOT NULL		
street_adress2		Varchar(1000			
City_ID	FK	INT	NOT NULL	CITY.City_ID	

postal_code		Varchar(15)			

Student_Guardian					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Student_ID	PK/FK	INT	NOT NULL	GENERAL_ROLE.ROLE_ID	
Guardian_ID	PK/FK	INT	NOT NULL	EMPLOYEE.EMPLOYEE_ID	
is_emergency_contact		BOOL	NOT NULL		
relationship		Varchar(200)	NOT NULL		

MODULE					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
module_id	PK	SERIAL			
course_id	FK	(INT)	NOT NULL	COURSE.course_id	
academic_level		ENUM	NOT NULL		
module_name		VARCHAR(250)			
cms_resource_id		VARCHAR(50)			
module_description		(TEXT)			

Assignment					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Assignment_ID	PK	Serial	NOT NULL		Assignment identifier
Module_ID	FK	INT	NOT NULL	module.module_ID	
Assignemnt_name		Varchar(50)	NOT NULL		
Weighting		Decimal	NOT NULL		
Assignment_Description		TEXT	NOT NULL		

Submission					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Submission_ID	PK	SERIAL	NOT NULL		Submission identifier
Module_ID	FK	INT	NOT NULL	module.module_ID	
Grade	FK	INT	NOT NULL		
Submission_Date		DATE	NOT NULL		
Submission_Feedback		TEXT	NOT NULL		

Lesson					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
lesson_Id	PK	SERIAL	NOT NULL		
Completed		Boolean	NOT NULL		
cms_resource_id		VARCHAR(50)			
lesson_starttime		TIME	NOT NULL		
lesson_date		DATE	NOT NULL		
lesson_endtime		TIME	NOT NULL		

Lesson_moduels					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
lesson_Id	PK/FK	INT	NOT NULL	Lesson.lesson_id	
module_id	PK/FK	INT	NOT NULL	module.module_id	

GROUP 29

Lesson_Feedback					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
lesson_Id	PK/FK	INT	NOT NULL	Lesson.lesson_id	
student_id	PK/FK	INT	NOT NULL	student.student_id	
Feedback_id	PK/FK	INT	NOT NULL	feedback.feedback_id	

Online_Lesson					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Lesson_ID	PK/FK	INT	NOT NULL		Lesson identifier
Platform	PK/FK	INT	NOT NULL		Platform usage identifier

GROUP 29

Physical_lesson					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
Sta	PK	Serial	NOT NULL		Staff identifier
staff_First_Name		VARCHAR(50)	NOT NULL		
Staff_Last_Name		VARCHAR(50)	NOT NULL		
staff_Role		VARCHAR(100)	NOT NULL		
Staff_Phone_Number		VARCHAR(20)	NOT NULL		
staff_Email		Varchar(50)	Not Null		

TEACHERS_IN_LESSON					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
LESSON_ID	PK/FK	(INT)	NOT NULL	LESSON.LESSON_ID	
EMPLOYEE_ID	PK/FK	(INT)	NOT NULL	EMPLOYEE.EMPLOYEE_ID	

GROUP 29

STUDENTS_IN_ATTENDANCE					
Attribute_Name	***KEY***	***Data Type & Size***	***Domains & Constraints***	***FK Reference***	***Description***
LESSON_ID	PK/FK	(INT)	NOT NULL	LESSON.LESSON_ID	
STUDENT_ID	PK/FK	(INT)	NOT NULL	STUDENT.STUDENT_ID	

Queries

```
--Select all branch managers, useful for when you want to contact any branch manager

--Fetches employee details, joining employee to branch staff and then branch in order to fetch branch name
--also fetches general employee and general role in order to query the 'Branch Manager' role in order to find the branch managers

SELECT
    e.first_name AS "First Name",
    e.last_name AS "Last Name",
    e.email AS "Email",
    e.phone_number AS "Phone",
    b.branch_name AS "Branch"
FROM employee e
JOIN branch_staff bs ON e.employee_id = bs.employee_id
JOIN branch b ON bs.branch_id = b.branch_id
JOIN general_employee_role ger ON e.employee_id = ger.employee_id
JOIN general_role gr ON ger.role_id = gr.general_role_id
```

```
WHERE gr.role_name = 'Branch Manager';
```

First Name	Last Name	Email	Phone	Branch
Kasey	Healing	khealing0@bluehost.com	7105309683	Main Branch
Jorge	Boshier	jboshiera@dailymotion.com	8605794325	Second Branch
Damiano	Karsh	dkarshk@washingtonpost.com	4296394266	Third Branch

(3 rows)

```
--Select top performing students and their emergency guardians details to contact about their high performance using their
emergency contact
--as they are more comfortable with being contacted.

--Selects students amd guardian where the query is filtered using AVG(submission grade) be above 90
SELECT
    CONCAT(s.first_name, ' ', s.last_name) AS "Student Name",
    ROUND(AVG(sub.grade), 1) AS "Average Grade",
    CONCAT(g.first_name, ' ', g.last_name) AS "Guardian Name",
    g.email AS "Guardian Email",
    g.phone_number AS "Guardian Phone",
    sg.relationship AS "Relation"
FROM student s
JOIN submission sub ON s.student_id = sub.student_id
JOIN student_guardian sg ON s.student_id = sg.student_id
JOIN guardian g ON sg.guardian_id = g.guardian_id
WHERE sg.is_emergency_contact = TRUE
GROUP BY
    s.student_id, s.first_name, s.last_name,
```

```
    g.first_name, g.last_name, g.email, g.phone_number,
    sg.relationship
HAVING AVG(sub.grade) > 90
ORDER BY "Average Grade" DESC;
```

```
    Student Name     | Average Grade |  Guardian Name   |       Guardian Email        | Guardian Phone
---------------------+---------------+------------------+-----------------------------+---------------
 Othella Kay         |          93.5 | Juan Bauduccio   | jbauduccio38@blinklist.com  | 4368757603
 Jonell Finley       |          93.5 | Bobbe MacRierie  | bmacrieriee@tuttocitta.it   | 3134885437
 Conni Clinning      |          92.5 | Maurie Robinett  | mrobinett4y@indiegogo.com   | 1156816623
 Sanders Koubek      |          92.5 | Farley Dominici  | fdominici3g@infoseek.co.jp  | 7851138722
 Valentijn Redihough |          91.5 | Arch McAvinchey  | amcavincheyu@ask.com        | 6381627413
 Thaddeus Waadenburg |          91.5 | Ginelle Boxhall  | gboxhall3s@independent.co.uk | 9398123242
 Bax Buckel          |          91.0 | Rina Sully       | rsully4€@berkeley.edu       | 9669188391
 Merrick Karpets     |          91.0 | Rollie Gallagher | rgallagher2y@about.me       | 1873228008
 Hobard Schollick    |          91.0 | Kelby Eriksson   | keriksson54@sitemeter.com   | 5442368457
 Rickard Shovel      |          91.0 | Ardis McGrorty   | amcgrorty1a@si.edu          | 5584295664
(10 rows)
```

```
-- View average grade achieved in every course to see where resources should be invested in, improvements should be made and where the
--underachieving students lie (on average). The number of students enrolled is also selected to show whether the numbers are skewed due to size of course

--Selects all courses, the number of students within those courses and the average grade to one decimal point ordered by average grade ASC
SELECT
    c.course_name AS "Course Name",
    COUNT(s.student_id) AS "Number of Students Enrolled",
    ROUND(AVG(sub.grade), 1) AS "Average Grade"
FROM course c
JOIN student s ON c.course_id = s.course_id
JOIN submission sub ON s.student_id = sub.student_id
JOIN assignment a ON sub.assignment_id = a.assignment_id
GROUP BY c.course_id
ORDER BY "Average Grade" ASC;
```

GROUP 29

Course Name	Number of Students Enrolled	Average Grade
Fine Arts	20	66.6
Philosophy 101	22	68.7
Computer Science 101	20	71.1
History	20	71.6
Art History	20	72.8
Hospitalilty Management	18	73.6
Graphics Design	20	75.7
Mathematics for Engineers	20	75.8
Business Administration	20	76.8
Physics 101	22	76.9
Electrician Fundamentals	18	79.1
Literature and Society	20	83.2

(12 rows)

```sql
-- All teachers who have taught online lessons with the platform and date, and link to the online lesson in order to access the lesson. This makes it so that if you want to check what was covered in the lesson, it is possible to do so.
SELECT
    CONCAT(e.first_name, ' ', e.last_name) AS "Teacher",
    ol.platform AS "Online Platform",
    ol.recording_link AS "Link",
    l.lesson_date AS "Lesson Date"
FROM employee e
JOIN teachers_in_lesson tl ON e.employee_id = tl.employee_id
JOIN lesson l ON tl.lesson_id = l.lesson_id
JOIN online_lesson ol ON l.lesson_id = ol.lesson_id
ORDER BY l.lesson_date ASC, l.lesson_starttime ASC;
```

GROUP 29

```
      Teacher       | Online Platform |             Link              | Lesson Date
--------------------+-----------------+-------------------------------+-------------
 Fitz Lalonde       | Zoom            | https://recording.zoom.uk/2   | 2024-10-02
 Sterling Rummery   | Zoom            | https://recording.zoom.uk/2   | 2024-10-02
 Ketti Daftor       | Google Meet     | https://recording.google.uk/4 | 2024-10-04
 Danella Icke       | Google Meet     | https://recording.google.uk/4 | 2024-10-04
 Randal Fishbourn   | Zoom            | https://recording.zoom.uk/6   | 2024-10-06
 Jasmin Garrigan    | Zoom            | https://recording.zoom.uk/6   | 2024-10-06
 Arlan Brisset      | Google Meet     | https://recording.google.uk/8 | 2024-10-08
 Dillie Matuszinski | Google Meet     | https://recording.google.uk/8 | 2024-10-08
 Sterling Rummery   | Zoom            | https://recording.zoom.uk/10  | 2024-10-10
 Tommy Monger       | Zoom            | https://recording.zoom.uk/10  | 2024-10-10
 Danella Icke       | Google Meet     | https://recording.google.uk/12 | 2024-10-12
 Jourdan Rapsey     | Google Meet     | https://recording.google.uk/12 | 2024-10-12
 Jasmin Garrigan    | Zoom            | https://recording.zoom.uk/14  | 2024-10-14
 Austin Hawke       | Zoom            | https://recording.zoom.uk/14  | 2024-10-14
 Dillie Matuszinski | Google Meet     | https://recording.google.uk/16 | 2024-10-16
 Emlynne Sibthorp   | Google Meet     | https://recording.google.uk/16 | 2024-10-16
 Tommy Monger       | Zoom            | https://recording.zoom.uk/18  | 2024-10-18
 Camel Fellibrand   | Zoom            | https://recording.zoom.uk/18  | 2024-10-18
 Jourdan Rapsey     | Google Meet     | https://recording.google.uk/20 | 2024-10-20
 Pierette Gummow    | Google Meet     | https://recording.google.uk/20 | 2024-10-20
 Austin Hawke       | Zoom            | https://recording.zoom.uk/22  | 2024-10-22
 Frazer Adkins      | Zoom            | https://recording.zoom.uk/22  | 2024-10-22
 Emlynne Sibthorp   | Google Meet     | https://recording.google.uk/24 | 2024-10-24
 Fitz Lalonde       | Google Meet     | https://recording.google.uk/24 | 2024-10-24
(24 rows)
```

```
--Find the 3 lowest performing modules and the teachers who taught those modules, to either warn those teachers or fire them

--Uses a Common Table Expression to get a temporary result set. It finds and limits the lowest average grade modules to 3 AS
lowest_modules.
--Lowest modules is then used to retrieve the Module, the Teacher and average grade, joining lesson modules, lesson,
teachers in lesson, employee, assignment and submission
--to select the correct columns
WITH lowest_modules AS (
    SELECT
        m.module_id,
```

GROUP 29

```
        m.module_name AS "Module",
        AVG(sub.grade) AS average_grade
    FROM module m
    JOIN lesson_modules lm ON m.module_id = lm.module_id
    JOIN lesson l ON lm.lesson_id = l.lesson_id
    JOIN assignment a ON m.module_id = a.module_id
    JOIN submission sub ON a.assignment_id = sub.assignment_id
    GROUP BY m.module_id, m.module_name
    ORDER BY average_grade ASC
    LIMIT 3
)
SELECT
    lm."Module",
    CONCAT(e.first_name, ' ', e.last_name) AS "Teacher",
    ROUND(AVG(sub.grade), 1) AS "Average Grade"
FROM lowest_modules lm
JOIN lesson_modules lm2 ON lm.module_id = lm2.module_id
JOIN lesson l ON lm2.lesson_id = l.lesson_id
JOIN teachers_in_lesson tl ON l.lesson_id = tl.lesson_id
JOIN employee e ON tl.employee_id = e.employee_id
JOIN assignment a ON lm.module_id = a.module_id
JOIN submission sub ON a.assignment_id = sub.assignment_id
GROUP BY lm."Module", e.first_name, e.last_name
ORDER BY lm."Module", "Average Grade" ASC;
```

GROUP 29

Module	Teacher	Average Grade
Baroque Art	Randal Fishbourn	57.5
Baroque Art	Jasmin Garrigan	57.5
Introduction to Business Management	Arlan Brisset	56.5
Introduction to Business Management	Dillie Matuszinski	56.5
Introduction to Poetry	Randal Fishbourn	58.0
Introduction to Poetry	Pierette Gummow	58.0

(6 rows)

Security Considerations

Within a database, users should rarely be able to access the raw data. Data integrity should be maintained through the use of views. This ensures that from the data within the database, what a user can see can be controlled. This can also be further ensured by making views specific to the user of the database. For example, within our database these views are made:

```
CREATE VIEW student_grades_view AS
SELECT s.student_id, a.assignment_name, sub.grade
FROM submission sub
JOIN assignment a ON sub.assignment_id = a.assignment_id
JOIN student s ON sub.student_id = s.student_id;


CREATE VIEW student_info_view AS
SELECT student_id, first_name, last_name, email, phone_number, course_id
FROM student;
GRANT SELECT ON student_info_view TO teacher_role;
GRANT SELECT, INSERT, UPDATE, DELETE ON lesson, online_lesson, physical_lesson TO teacher_role;


CREATE VIEW employee_view AS
SELECT e.employee_id, e.first_name, e.last_name, e.email, e.phone_number, bs.branch_id
FROM employee e
JOIN branch_staff bs ON e.employee_id = bs.employee_id;
GRANT SELECT ON employee_view TO branch_manager_role;


CREATE VIEW performance_view AS
SELECT c.course_id, c.course_name, AVG(sub.grade) AS avg_grade
FROM course c
JOIN student s ON c.course_id = s.course_id
JOIN submission sub ON s.student_id = sub.student_id
GROUP BY c.course_id, c.course_name;
GRANT SELECT ON performance_view TO branch_manager_role;
```

The issue with these views is that a user could still be able to access all data within the relevant view. For example, for the student_grades_view, if the student is the user and wants to see their own grades, it should be such that they can only view their own data. This view does not ensure that. In order to implement this, Row Level Security must be introduced. Row Level Security is controlling access to data by row, ensuring a user is only able to access data they are authorised for (Berning, 2024). By using "ENABLE ROW LEVEL SECURITY" and "CREATE POLICY" on the submission table where the policy defines that the student_id must match the current_user, this can be achieved (*Row Level Security (RLS)*, n.d.). This should be done for employee_view where the branch id must equal to the user's branch id, where the user is a branch manager.

Then, roles that will be assigned to users can be introduced in order to control who can access these views, for example, a student user must not have access to employee data through the employee view.

```
CREATE ROLE student_role;
CREATE ROLE teacher_role;
CREATE ROLE branch_manager_role;
CREATE ROLE admin_role;
```

Then you can grant select on the views based on the user:

```
GRANT SELECT ON student_grades_view TO student_role, teacher_role, branch_manager_role, admin_role;
```

```
GRANT SELECT ON student_info_view TO teacher_role, branch_manager_role, admin_role;
GRANT SELECT, INSERT, UPDATE, DELETE ON lesson, online_lesson, physical_lesson TO teacher_role, branch_manager_role,
admin_role;
```

Only students are not allowed to access the lessons.

```
GRANT SELECT ON employee_view TO branch_manager_role, admin_role;
GRANT SELECT ON performance_view TO branch_manager_role, admin_role;
```

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin_role;
```

Then by creating users to assign these roles with passwords, you are able to ensure data security:

```
CREATE USER student_user WITH PASSWORD 'student';
CREATE USER teacher_user WITH PASSWORD 'teacher';
CREATE USER branch_manager_user WITH PASSWORD 'manager';
CREATE USER admin_user WITH PASSWORD 'admin';

ALTER ROLE student_user WITH LOGIN;
ALTER ROLE teacher_user WITH LOGIN;
ALTER ROLE branch_manager_user WITH LOGIN;
ALTER ROLE admin_user WITH LOGIN;

GRANT student_role TO student_user;
GRANT teacher_role TO teacher_user;
GRANT branch_manager_role TO branch_manager_user;
GRANT admin_role TO admin_user;
```

This way if a student user tries to access employee data:

```
postgres=# \c secondyearcw student_user
Password for user student_user:
You are now connected to database "secondyearcw" as user "student_user".
secondyearcw=> SELECT * FROM EMPLOYEE_VIEW;
ERROR:  permission denied for view employee_view
```

While if a branch manager tries to:

```
postgres=# \c secondyearcw branch_manager_user
Password for user branch_manager_user:
You are now connected to database "secondyearcw" as user "branch_manager_user".
secondyearcw=> SELECT * FROM employee_view;
 employee_id | first_name |   last_name   |              email              | phone_number | branch_id
-------------+------------+---------------+---------------------------------+--------------+-----------
           1 | Kasey      | Healing       | khealing0@bluehost.com          | 7105309683   |         1
           2 | Dillie     | Matuszinski   | dmatuszinski1@psu.edu           | 6895002353   |         1
           3 | Emlynne    | Sibthorp      | esibthorp2@xrea.com             | 8687380348   |         1
           4 | Fitz       | Lalonde       | flalonde3@ocn.ne.jp             | 6884413875   |         1
           5 | Sterling   | Rummery       | srummery4@rambler.ru            | 3553060067   |         1
           6 | Tommy      | Monger        | tmonger5@wordpress.org          | 9615460611   |         1
           7 | Merilyn    | Boyse         | mboyse6@privacy.gov.au          | 1431909200   |         1
           8 | Griffy     | Swetman       | gswetman7@jugem.jp              | 9479223744   |         1
```

A log table that only the database administrator has access to could be created in order to log all activity that occurs within the database itself. This would be done through the creation of an action_log table. A function could then be created that inserts the appropriate information about the log into the table. Then a trigger can be created so that when a CRUD operation happens, the function can be executed. This ensures that activity is tracked, adding a layer of data security.

GROUP 29

Optimisation

Three most performance-sensitive transactions:
Find the 3 lowest performing modules and the teachers who taught the modules (QUERY 5)

TRANSACTION / TABLE	MODULE	ASSIGNMENT	SUBMISSION	LESSON	LESSON_MODULE
CREATE					
READ	X	X	X	X	X
UPDATE					
DELETE					

Reasons:

- Multiple Joins – Module, lesson_modules, lesson, assignment, submission, teachers_in_lesson and employee tables
- Redundant Computation – Avg(sub.grade) calculated twice, adding computational overhead
- Nested Query – Additional computation

Performance Requirements:

- Query must handle high volume of data effectively
- Maintain low execution time
- Maintain low planning time

Optimisation Techniques:

- Remove redundant avg(sub.grade) calculation. Can be reused from the first instance rather than repeating it
- Denormalisation – Store precomputed average grades for each module and teacher in a separate table with the table being updated every time new submissions are added. This eliminates the need for the query as a whole as you just need to select on the new table, which is much more optimised
- Indexing – By adding indexes on the targeted JOIN tables, the need for sequential scanning is reduced, speeding up the joins and improving planning and execution times.
- Materialised Views – Store results of the CTE in a materialised view to select from, reducing computational overhead as rather than manipulating data within the query itself, we can just fetch it from a pre-calculated table storing all the data we need. The materialised view performs the percomputation at regular intervals meaning when we use the data, it will be up to date. (*What Is a Materialized View?*, n.d.)

GROUP 29

Select top performing students and their emergency guardian details (QUERY 2)

TRANSACTION / TABLE	STUDENT	SUBMISSION	STUDENT_GUARDIAN	GUARDIAN
CREATE				
READ	X	X	X	X
UPDATE				
DELETE				

Reasons:

- HAVING clause – Filtering by average grade causes increased computational complexity due to processing all rows before applying the condition
- AVG(sub.grade) – Used for every student, adding computational overhead
- Multiple Joins – Student, submission, student_guardian and guardian tables

Performance Requirements:

- Company expansion means the database will expand rapidly with more students and these students will have many submissions each. Query must be able to handle large datasets within the thousands or even tens of thousands.
- Execution Time – Although data size is relatively small currently, execution time should be around under a second due to the potential increase in students
- Planning Time – Similar to Execution Time.

Optimisation Techniques:

- AVG(sub.grade) – Can be calculated using CTE or as previously mentioned, materialised view, storing the calculated average for each student in a table, saving calculations and increasing efficiency.
- Indexing – Indexing on commonly used columns: student_id, guardian_id, grade and is_emergency_contact, reduces need for sequential scanning, speeding up execution time of the JOINs
- Denormalisation – Storing students and their guardians into a single table, avoiding JOINs and simplifying the query.

GROUP 29

Select lesson, online_lesson with details and teacher (QUERY 4)

TRANSACTION / TABLE	EMPLOYEE	TEACHERS_IN_LESSON	LESSON	ONLINE_LESSON
CREATE				
READ	x	x	x	x
UPDATE				
DELETE				

Reasons:

- Expansion – With expansion looming, the number of lessons will grow with the company. Queries involving the lesson table will then be more performance sensitive as it will handle a much larger data set than it currently does.
- Multiple Joins – Employee, teachers_in_lesson, lesson and online_lesson tables

Performance Requirements:

- Handling Growth – Query must process larger datasets while maintaining performance
- Execution time – Execution time remains low regardless of increased data volume
- Planning time – Indexes needed to ensure planning time remains low despite data increase

Optimisation Techniques:

- Indexing – Indexing columns: lesson_id, employee_id and lesson_date speeds up the join and sorting process, reducing sequential scanning and ultimately improving performance
- Denormalisation – By storing the lesson with lesson type and the relevant columns all in one table, the need for JOINs is eliminated and as such the query becomes simplified.
- Materialised View – Rather than sorting the data within the query, it can be done within a materialised view in order to save computation and improve query performance.

Professional, Legal and Ethical issues Considerations

Legal, professional And Ethical issues

When considering ethics and intellectual property in regard to our database, an example may be the work of students, such as assignments, as their property. To avoid issues in our database, we have implemented attributes such as submission_ID. This gives our students individual identities, making it easier for us to detect plagiarism. Unless otherwise stated in the terms of use, we guarantee that the students retain the intellectual property rights to their work.Privacy is an important ethical factor for our database, as we are trusted with personal information such as addresses and emails from our students. Respecting their privacy is something we value for our database. This allows us to comply with the UK's Data Protection Act of 2018. Within our database, we only collect necessary data, as we don't want to overwhelm users or collect unnecessary information. This ensures that our users don't feel like we are going to exploit their data.Going through our code, it's evident that we do not ask for ethnicity prior to joining, as we don't want anyone to feel discriminated against based on their ethnicity. This aligns with the UK's legal protection laws. The ethical issues of whether it is appropriate to collect certain data about individuals, use it without their consent, or share it with third parties are the subject of much debate. Within our database, we don't include the salaries of staff, as we believe this to be a personal and unnecessary attribute of data.As mentioned in our security paragraph, access to the database follows a hierarchical structure, or what you could call a pyramid. Students have limited access, which is less than teachers, and teachers have less access than administrators. This ensures that personal information is unlikely to be accessed by untrusted workers within our database. We do this because we understand the conflict between an individual's right to privacy and the desire of governments and businesses to access useful information. We followed and drew inspiration from various codes of ethics, such as those from ACM and IEEE.

We balanced usability and utility during the design of our database, applying the five qualities of usability: easy learnability for our students, quick efficiency throughout the database, memorability for logging back in, easy recovery from errors, and overall user satisfaction. Usability was a priority in our design to ensure a satisfying user experience.

GROUP 29

Professional

By following the five usability qualities—easiness of learning, efficiency, memorability, mistake recovery, and user satisfaction—we gave usability and utility first priority when designing our database. Having these rules make sure that our users have smooth and simple to use experience, teachers, administrators, and students have easy access to the database. A hierarchical access system also guarantees the safe and expert handling of private data. The database we have created has strict professional requirements for usability and functionality by implementing from IEEE and ACM.

References:

Berning, T. (2024, January 5). What is Row-Level Security? *NextLabs*. https://www.nextlabs.com/what-is-row-level-security/

Row Level Security (RLS): Basics and Examples. (n.d.). Satori. Retrieved 8 December 2024, from https://satoricyber.com/postgres-security/postgres-row-level-security/

What is a Materialized View? - Materialized View Explained - AWS. (n.d.). Amazon Web Services, Inc. Retrieved 14 December 2024, from https://aws.amazon.com/what-is/materialized-view/