Questions from 3.2.3

Description of Our Project:

The goal of our project was to represent the current state of a university residence. To do this, we created a database to store information about the state of dorms, buildings, units, and rooms. In our database, we also tracked different kinds of residents, such as students and residence advisors (RAs). With these entities, we kept track of which room each resident stayed in, as well as which student was monitored under which residence advisor. We also stored information about sublets, packages, maintenance requests and maintenance staff, so we could monitor the activity of the residents.

Changes in Our Final Schema:

Our final schema differed in a few ways. One, we had to change all BOOLEAN datatypes to VARCHAR as the BOOLEAN datatype was not supported. In the Unit table, we also changed the column 'number' to 'unitNumber' as 'number' is not a valid column name. For the same reason, in the Room_R2 table, we changed the column 'number' to 'roomNumber'. In the PermanentResident table, we also removed the column 'subletId' as our Sublet table already referenced a PermanentResident with a participation constraint. In MaintenanceRequest, we also updated the 'staffId' column to be NOT NULL as it is foreign key referencing a MaintenanceStaff entity, so it would make sense for each request to not have a null staff assignment.

SQL Queries Used:

2.1.1 – 2.1.6:

SQL query	copy	Where it can be found
INSERT	INSERT INTO PermanentResident (studentId, roomNumber, unitNumber, buildingName, age, name, email) VALUES (:id, :room, :unit, :bld, :age, :nm, :email)	appService.js, line 308
UPDATE	UPDATE PermanentResident SET	appService.js, line 332

	${field} = :valstring WHERE studentId= :id UPDATE PermanentResident SET ${field}=:valnum WHERE studentId=:id	
DELETE	DELETE FROM PermanentResident WHERE studentId=:id	appService.js, line 361
Selection	SELECT studentId, age, name, email, roomNumber, unitNumber, buildingName FROM PermanentResident ${q} ((q) is a valid WHERE query generated from user input)	appService.js, line 385
Projection		appService.js, lines 227 - 230
Join		appService.js, lines 247 - 251

2.1.7 – 2.1.10:

Sql query	Copy	Description	Where it can be found
Aggregation with GROUP BY	SELECT p.studentId, p.name, d.earliest_delivery FROM (SELECT studentId, min(deliveryDate) as earliest_delivery FROM Package GROUP BY studentId) d JOIN PermanentResident p ON d.studentId = p.studentId	Gets the earliest date a package was delivered for every resident with at least one package delivery. Joins with PermanentResident to return the resident names in addition to the resident IDs.	appService.js, lines 159-168
Aggregation with HAVING	SELECT buildingName, COUNT(*) FROM PermanentResident GROUP BY buildingName	Gets the number of residents in each building. Filters out	appService.js, lines 185-190

	HAVING COUNT(*) >= :min	all the buildings where the number of residents is less than the given requirement.	
Nested aggregation with GROUP BY	SELECT buildingName, avg(sqfeet) FROM Room_R2 GROUP BY buildingName HAVING avg(sqfeet) >= (SELECT avg(sqfeet) FROM Room_R2)	Gets the average room square footage in each building. Filters out all the buildings where the average room sqft is less than the average room sqft in all buildings.	appService.js, lines 206-212
Division	SELECT DISTINCT r.buildingName FROM Room_R2 r WHERE NOT EXISTS (SELECT r1.sqFeet FROM Room_R1 r1 WHERE NOT EXISTS (SELECT 1 FROM Room_R2 r2 WHERE r2.buildingName = r.buildingName AND r2.sqFeet = r1.sqFeet))	Get buildings that contain rooms with every possible size (by sqft)	appService.js, lines 267 - 276