Team 37 – CSC2042 QA Project

Nathan Donaghy 40226743

[ndonaghy12@qub.ac.uk](mailto:ndonaghy12@qub.ac.uk)

David Mackenzie 40238376

[dmackenzie01@qub.ac.uk](mailto:dmackenzie01@qub.ac.uk)

Keelan Logan 40206821

[klogan03@qub.ac.uk](mailto:klogan03@qub.ac.uk)

Peter Sleith 40237264

[psleith01@qub.ac.uk](mailto:psleith01@qub.ac.uk)

Daniel White 40233631

[dwhite23@qub.ac.uk](mailto:dwhite23@qub.ac.uk)

**Entity Relationship Diagram**

A close up of a map

Description automatically generated

**List of Constraints and/or Assumptions in E-R Diagram Design**

A building can contain one or many apartments, but an apartment must be contained within one building.

From the assignment, it states that QA manages two disjoint categories of people, tenants and employees, therefore it is assumed than a person can be either an employee or a tenant but not both. There exists an “is a” relationship between these entities, i.e. an employee is an instance of a person, or a tenant is an instance of a person. Furthermore, employee can be either a technician, a manager or both. There also exists an “is a” relationship between these entities, i.e. a technician is an instance of an employee, and or a manager is an instance of an employee. Each person must also have their bank details recorded as employees are paid a salary and tenants must pay rent. The constraint in the ERD shows that each instance of a bank account must have one related person.

Each “Technician” can have one or more of the 3 outlined skills. Therefore, a “Skills” table was created to store the 3 skills and a separate TechnicianSkills table was created to store what skills each instance of a technician has. This was required for technicians with more than one skill in order to follow the relational database convention, otherwise it would be stored in a multi-valued attribute which is not allowed. Therefore, each Technician can have one or many TechnicianSkills, and each instance of TechnicianSkills must have one associated Technician. As well as this, each instance of TechnicianSkills must have one associated Skill and each Skill can have many associated TechnicianSkills.

In relation to “Lease”, a lease must be signed by one or more tenants, must be signed only one manager and must have only one related apartment. Another entity called “LeaseTenants” was created to allow the storage of multiple tenants under one lease agreement. This was necessary to ensure we were following relational database convention as multi-valued attributes are not allowed. A tenant can sign one or many leases as they may stay in different apartments over time, and the system will have both expired and current leases stored as required in the assignment. Additionally, a lease must have only one apartment, but each apartment can have one or many leases as the system will have both current and expired leases. Also, each lease must have one manager to approve the lease and each manager can approve one or many leases.

In relation to “Apartment” and “Manager”, by following the assignment document, it can be assumed than each apartment requires only one manager, therefore in the created ERD each manager manages one or many apartments and each apartment must have one manager.

In relation to “Office”, each manager is assigned to one office and each office can have one manager. Also, in the assignment it states that each office is located within an apartment owned by QA, hence in the ERD each office must be contained within one apartment and each apartment can have zero or many offices.

**Table Schemas**

Listed below are the tables that have been created, the relationships that have been modelled and the primary and foreign keys of each table. Primary keys are underlined. Foreign keys have a \* beside them. Relationships are discussed under the appropriate table.

Person (PersonID, FName, SName, ContactNumber, DoB, EmergencyContactName, EmergencyContactNumber)

* Related to Bank\_Account table. Each person ‘has a’ bank account.

Employee (EmployeeID, PersonID\*, Salary)

* Related to Person table. Every employee is a person.

Tenant (TenantID, PersonID\*)

* Related to Person table. Every tenant is a person.

Bank\_Account (BankAccountID, PersonID\*, AccountNumber, SortCode)

Manager (ManagerID, EmployeeID\*)

* Related to Employee table. Every manager is an employee.

Building (BuildingID, BuildingNameOrNum, Street, PostCode)

Apartment (ApartmentID, ApartmentNo, NumBedrooms, NumBathrooms, TotalArea, BuildingID\*, ManagerId\*)

* Related to Building table. Every Building ‘contains’ an apartment.

Lease (LeaseID, ApartmentID\*, ManagerID\*, StartDate, Duration, Rent)

* Related to Apartment and LeaseTenant table. Every lease is ‘assigned to’ an apartment and every tenant ‘signs’ a lease which creates a LeaseTenant table that uses the TenantID and LeaseID as foreign keys.

LeaseTenant (LeaseID\*, TenantID\*)

Office (OfficeID, ApartmentID\*, ManagerID\*)

* Related to Apartment. Every office is ‘contained in’ an apartment.

Technician (TechnicianID, EmployeeID\*)

* Related to Employee table. Every technician is an employee.

Skill (SkillID, SkillName)

* Related to Technician through a TechnicianSkills table. Every technician is ‘skilled in’ a skill. As this is a many to many relationship a TechnicianSkills table is created using primary keys from Technician and Skills as foreign keys – TechnicianID and SkillID.

TechnicianSkill (TechnicianID\*, SkillID\*)

**SQL Querying – Query 1**

QA wants to know which of its leases and apartments are still able to accommodate tenants and how many it can accommodate.

1. SELECT alt.LeaseID, alt.ApartmentNo, CONCAT(Building.BuildingNameorNum, ' ', Building.Street, ' ', Building.PostCode) AS 'Address',(alt.NumBedrooms - alt.NumTenants) AS 'Number of Free Bedrooms'

FROM (SELECT Apartment.ApartmentNo,Apartment.NumBedrooms,Apartment.BuildingID,lt.LeaseID,lt.NumTenants

FROM Apartment

2 .INNER JOIN (SELECT LeaseTenants.LeaseID,LeaseTenants.TenantID, Lease.ApartmentID, COUNT(\*) AS 'NumTenants'

FROM LeaseTenants

3. INNER JOIN Lease ON LeaseTenants.LeaseID = Lease.LeaseID WHERE (SUBDATE(CURRENT\_DATE, INTERVAL Lease.Duration MONTH) < Lease.StartDate) GROUP BY LeaseTenants.LeaseID)lt ON lt.ApartmentID = Apartment.ApartmentID)alt

4. INNER JOIN Building ON alt.BuildingID = Building.BuildingID WHERE (alt.NumBedrooms - alt.NumTenants > 0)

This query finds all current, non-expired leases for apartments where there are more bedrooms than tenants living in the apartment.

The query selects the leaseID, ApartmentNo, Address(concatenation of BuildingNameorNum, Street and PostCode) and the Number of Free Bedrooms(which is the number of tenants subtract number of bedrooms in the apartment), from apartment and inner joins it with a subquery.

This subquery selects all ongoing leases and checks that they are not expired, by using: WHERE (SUBDATE(CURRENT\_DATE, INTERVAL Lease.Duration MONTH) < Lease.StartDate).

This takes the current date and subtracts the number of months of the duration of the contract. If this is less than the lease start date, the contract has not yet expired.

These found ongoing leases are then inner joined to the LeaseTenants table. Now the number of tenants in each ongoing lease can be discovered by counting entry, using: “COUNT(\*) AS 'NumTenants' ”, and grouping the LeaseID counts together using: “GROUP BY LeaseTenants.LeaseID“.

The previous query selects the buildingID of the selected apartments on the lease This query is then inner joined with the building table using buildingID. This is done so that the address of the apartment can be displayed as it is stored within the building table.

The final where clause then ensures that only apartments where there are more bedrooms than tenants are selected.

**Coping with changes**

**Expansion Plan**

There is an increasing number of tenants in Queens Accommodation and there are no longer enough rooms to fulfil all the requests. As a result of this Queens would like to add more buildings to their portfolio. They have decided to purchase space in a different area of Belfast where the rent for tenants would be cheaper as this was another concern. Queens will need to hire more technicians and managers to cope with this change, however with there being more buildings queens would like more employees to carry out trivial jobs such as cleaning empty apartments or helping new tenants move in. Due to the seasonal nature of these jobs they have decided to offer these as casual contracts to current tenants instead of year-round contracts. They would also like all their employees to be assigned to one area to ensure work can be carried out quickly and efficiently should problems arise.

**System Changes**

Firstly, there will need to be an Area table created that would be connected to Building in the ERD with a one-to-many relationship. This allows prospective tenants to decide between different areas for their stay and they can research the advantages of them. The Area table would contain a primary key of AreaID and contain the area name, e.g. “Malone Road” or “City Centre”. Additionally, AreaID will be used as a foreign key in the Building table and another which I’ll discuss later.

In order to allow tenants to be employees there needs to be some changes to the current ERD and the relationships modelled. As there are now two different types of employee, I would model it the same way as Person with the table splitting into two directly under it. Employee now needs to have PermanentEmployee and CasualEmployee under it with many-to-many relationships instead of Technician and Manager which will be directly under Permanent Employee with the same relationships to PermanentEmployee as was previously modelled when linked to Employee. The CasualEmployee table will also have a link to the Tenant table. This relationship would be one-to-one.

In the same way as the Technician and Skills were modelled CasualEmployee will have a many-to-many ‘skilled in’ relationship with a new table called CasualSkills which will have a primary key of SkillID and skill names such as ’cleaning’ or ‘admin’. Due to this many-to-many relationship when it comes to implementing the database a table will need to be created called CasualEmployeeSkills which will have the same principle as the TechnicianSkills table with the primary keys of the participating entity sets - CasualEmployeeID and CasualSkillsID used as foreign keys in this table. I do not see any need for this table to be shown on the new ERD.

With there now being such a range of employees there would also be reason to include an EmployeeContract table that has a one to one relationship with Employee. This table could potentially be left out of the ERD. Due to the expansion causing there to be jobs across Belfast this table should include the foreign key AreaID which means an employee is only specific to that area.

The salary attribute would be taken from employee and put in PermanentEmployee. The corresponding attribute in the CasualEmployee table would be hourly rate. The reason for this is because casual employees essentially have zero-hour contracts and will be paid an hourly rate as opposed to Technicians and Managers who will be paid a fixed salary.

**Individual Contributions Record**

Nathan Donaghy 40226743:

I spent a lot of my time working on the ER diagram trying to figure out what entities were needed and the the links and constraints between these entities. However, my initial diagram was quite complex, so I worked alongside Daniel White to simplify it down to what was used in the submission. In addition to this, I created the cleandb37 and David Mackenzie and I worked together completing the createdb37. Furthermore, I was responsible for designing and creating query 1 in our groups querydb48 file.

David Mackenzie 40238376:

I opted to using a GitHub repo to keep all the work together, which helped our group a lot in seeing what everyone worked on. Nathan, Keelan and I drew up the first version of the ER diagram in the first meeting, this gave us a basis to work on in the future and it has since been changed a lot but follows a similar pattern to the first version. I wrote the queries for what the tables would largely follow throughout the rest of the project, which were in older files to the newly submitted createdb37.sql file, this meant simple changes and additions were easily added. I created the 2nd query in our querydb37 file, I added a comment within the file to make it clear what it does