**Task 1: Designing an ER Model**

Assumptions:

* The “OnDemand” entity is created as agents may be required to travel to different locations other than the store they are based in.
* The “Order” and “Customer” entities were created as it can be assumed that products are being sold to customers.
* A “Website” entity was creating assuming that the company’s website allows customers to select the location of the store and the availability of any given product, with their selling prices and available quantities.
* As stated in the business requirements, some products are not available and replacement products are then recommended. Therefore, a “Replacement” entity is created to allow similar products to be recommended to customers.

**Task 2A: Initial Design**

Assumptions:

* Not all staff supervises projects
* All projects will have at least one staff working on
* As some vehicles of the company are not categorised by any type, it is assumed that these uncategorised vehicles are stored in a pool of other uncategorised vehicles. Therefore, an “UncategorisedVehicleType” entity was created.

**Task 2B: Client Adjustments**

Given the initial design of the ER model:

1. It is possible to include customer information in the model. A separate entity for the customers can be created which will be linked to the “Vehicle” entity.
2. To support this requirement, a “FavouriteStaff” entity is created which stores the unique identifier of the customer and a multi-valued attribute of staff members identified by their StaffNo. This entity allows for any given customer to list up to multiple of their favourite staff members.
3. With a “Customer” entity added, a relationship can be created between “Staff” and “Customer”. As each staff member is allowed to blacklist only up to one customer, a relationship attribute is created. A many-to-one relationship is created between “Staff” and “Customer” as it is assumed that more than one staff can blacklist any particular customer.
4. This can also be achieved by removing the “Active” entity in the model. This allows for the history of all contracts associated with each project to be maintained regardless of activity of the contract.

**Task 3: Mapping an ER Model to a Relational Database Schema**

Step 1: Strong Entities

Branch (BranchNo, Address, Telephone)

Staff (StaffNo, Name, Address, Position, Salary)

Property (PropNo, Address, Type, NoOfRooms, WeeklyRent, AvailableForRent, AdOnOtherWebsites)

Step 2: Weak Entities

None

Step 3: One-to-One Relationships

None

Step 4: One-to-Many Relationships

Staff (StaffNo, Name, Address, Position, Salary, BranchNo\*)

Branch (BranchNo, Address, Telephone, StaffNo\*, StartDate, MonthlyBonus)

Staff (StaffNo, Name, Address, Position, Salary, SuperStaffNo\*)

Property (Prop No, Address, Type, NoOfRooms, WeeklyRent, AvailableForRent, AdOnOtherWebsites, StaffNo\*)

Step 5: Many-to-Many Relationships

None

Step 6: Multi-Valued Attributes

TelephoneNo (BranchNo\*, Telephone)

Step 7: High-Degree Relationships

None

Final Relational Database Schema:

Staff (StaffNo, Name, Address, Position, Salary, BranchNo\*, SuperStaffNo\*)

Branch (BranchNo, Address, Telephone, StaffNo\*, StartDate, MonthlyBonus)

Property (Prop No, Address, Type, NoOfRooms, WeeklyRent, AvailableForRent, AdOnOtherWebsites, StaffNo\*)

TelephoneNo (BranchNo\*, Telephone)

**Task 4: Relational Database Model**

1. Assuming that the entity integrity constraint is applied during the creation of the “Jobs” relation, then the database schema ensures that there is a job associated with each employee. The “JobHistory” relation contains the “employee\_id” and “job\_id” attributes as a foreign keys from the “Employees” and “Jobs” relations respectively.
2. The “JobHistory” relation has identified “employee\_id”, “start\_date” and “end\_date” as the primary keys of the relation. The keys constraint is enforced here, therefore the database will not allow for storage of multiple job histories of the same employee with the same start and end dates but with different job titles.
3. The given SQL statements are only sufficient in capturing the data on the three new sub-departments, namely Ongoing Staff, Casuals and External Contractors. However, as the first SQL statement has overwritten the previous tuple of Human Resource, the requirement of Joseph being the Director of Human Resources is not satisfied.

Therefore, these SQL statements only record the information on the managers of the new sub-departments, but left out the initial department, which is Human Resources.

1. The given query only updates the “empjob\_id” in the “Employees” relation. The query only updates Adam Smith’s new position but does not add/update any of his past contracts. Hence, a separate query needs to be created that updates the “Job\_History” relation with Adam’s past job positions.

E.g. INSERT INTO Job\_History VALUES(50, start\_date, end\_date, 33, 1)

1. Executing this SQL statement will result in an error as there already exists a “location\_id” with the value of 10. As “location\_id” is the primary key of the “Locations” relation, this SQL statement violates the keys constraint which requires all primary keys to be unique.
2. CREATE TABLE Departments

(

department\_id INTEGER NOT NULL,

department\_name VARCHAR(50),

manager\_id INTEGER,

location\_id INTEGER,

PRIMARY KEY (department\_id),

FOREIGN KEY (location\_id) REFERENCES Locations(location\_id)

);

1. CREATE TABLE JobHistory

(

employee\_id INTEGER NOT NULL,

start\_date INTEGER NOT NULL,

end\_date INTEGER NOT NULL,

job\_id INTEGER,

department\_id INTEGER,

PRIMARY KEY (employee\_id, start\_date, end\_date),

FOREIGN KEY (job\_id) REFERENCES Jobs(job\_id),

FOREIGN KEY (department\_id) REFERENCES Departments(department\_id)

);

As SQLite does not have a storage class for dates, the data type for “start\_date” and “end\_date” are set to INTEGER. This stores the dates in the database as Unix Time.