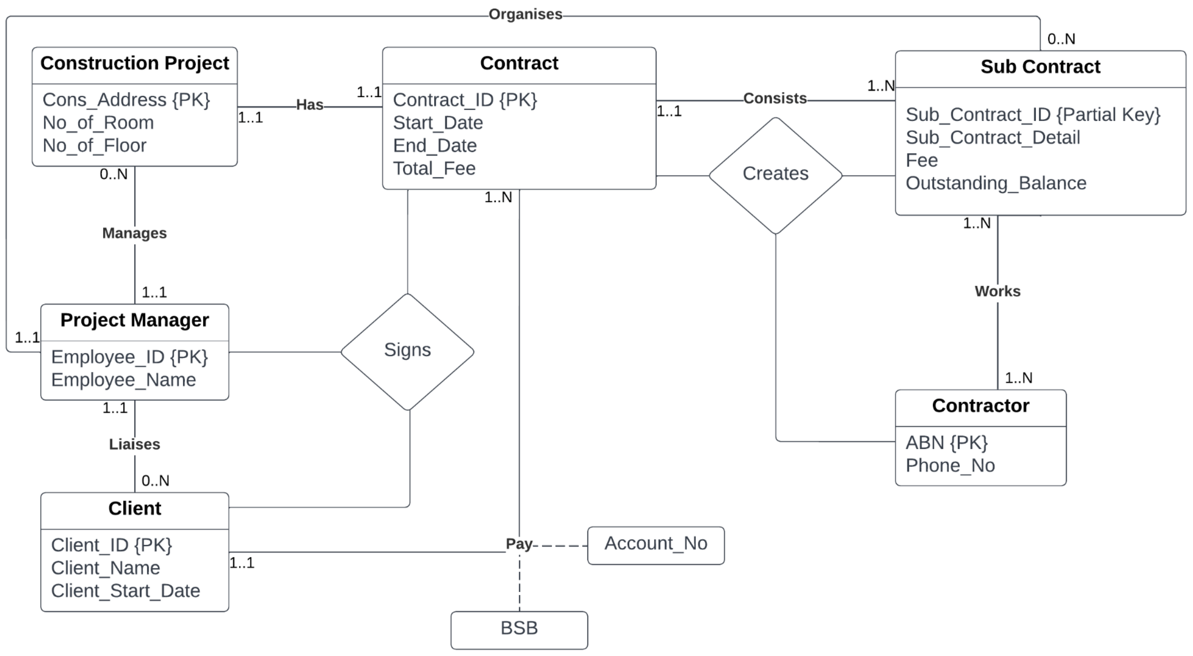
SUKHUM BOONDECHARAK (S3940976)
ISYS1055 Database Concept: Assignment 1

Task 1: Elite Construction Case Study



Construction Project is a strong entity as the unique address for each project has been stated clearly as a primary key. Contract, on the other hand, has not been given with a keyword for primary key. However, Contract and Constructions Projects are related with a one-to-one relationship because we can only have a contract for each project.

And since their jobs are rarely related, staff has been divided into two entities, which are Project Manager and Contractor. Project Manager with a unique employee ID is apparently a strong entity. While it is clear with Project Manager, Contractor is given with a hint of ABN, which will never be the same, so it can also be counted as a primary key for each Contractor. Each project manager can manage more than one project, but can also manage none assuming they are newcomers. In contrast, contractors are only assigned when there is a subcontract, therefore; it is a must to have at least one contractor for each subcontract.

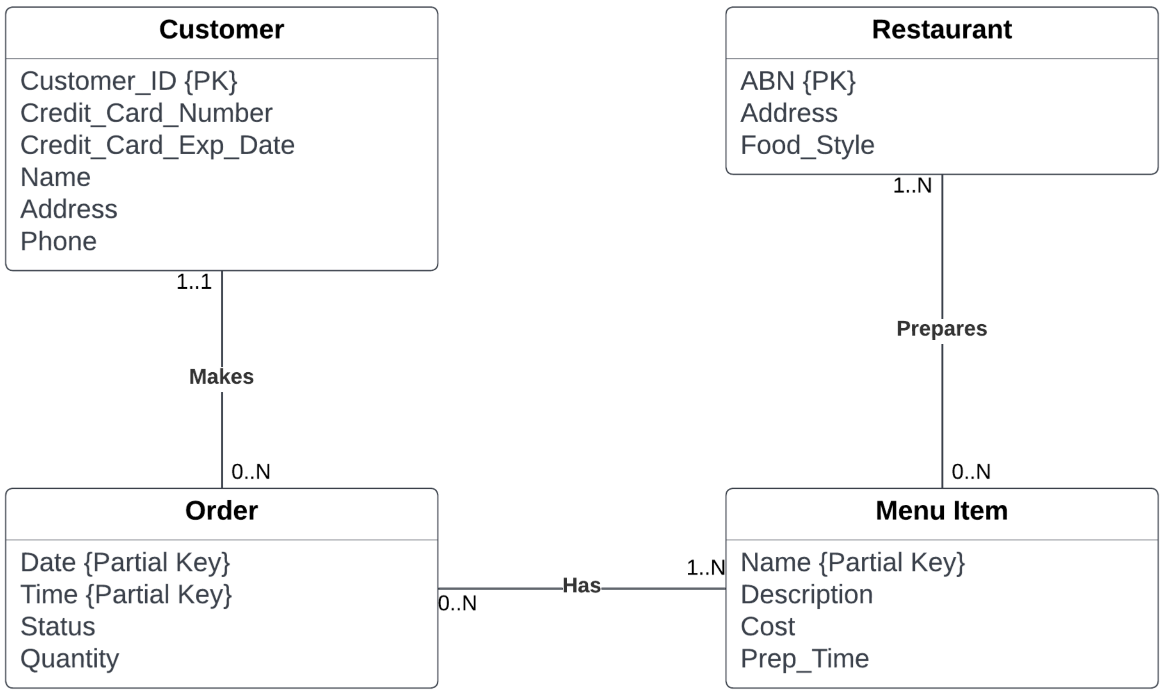
Sub Contract is a weak entity depending on Contract, and since a subcontract can never be without a contractor, we have Creates as a ternary relationship between Contract and Contractor to create Sub Contract. Sub Contract also has one-to-many relationship with Project Manager, while with Contractor, it becomes many-to-many relationship because each subcontract can have more than one contractor and one contractor can possibly work on multiple subcontracts.

Client is the last strong entity with ID as a primary key. It has a ternary relationship with Contract and Project Manager when signing a contract because a project manager must be assigned. It has been stated that clients can pay for more than one contract, but has not been mentioned that contracts can also be shared with many clients, so I only assume that each contract can only be paid by a client. While the payment is being proceeded, the account number and BSB of the Bank account are being recorded for each payment.

To be honest, this is on a different level compared to what we’ve learnt for six weeks in class. I am not even sure if my understanding is correct. A part of interpreting is so wide-open even given all of these constraints.

Task 2: Faster Food Case Study

Part A:

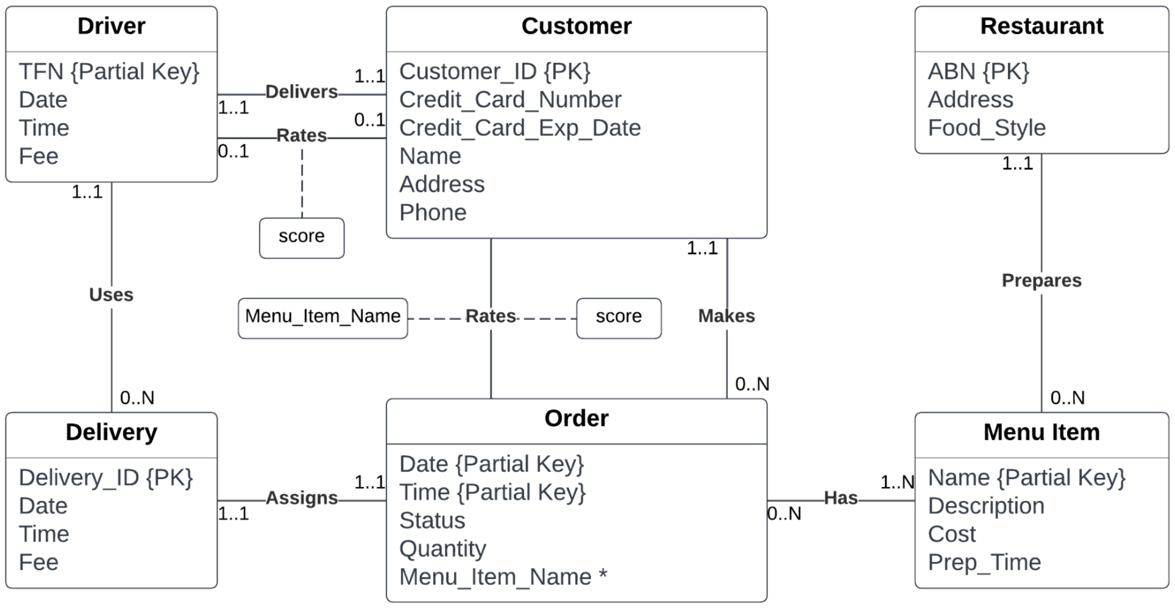


The only candidate key for Customer is the credit card number. However, it is not desirable in terms of security. Therefore, I added Customer_ID as a primary key for the entity instead. And Because Order is a weak entity, depending on each customer that makes an order and menu items it contains, Order can use the customer ID to be one of its composite primary keys, along with date and time of the order, menu ordered, and the restaurant's ABN to identify unique orders.

The same goes for Menu Item, where it can identify which order it is in using its name together with date and time of the order, along with the customer ID and the restaurant's ABN. Since one menu item can be in different order, while one order can also consist of several menu items, the relationship between these two weak entities is a many-to-many relationship. This is similar to a relationship between Restaurant and Menu Item. While one menu item can be made by different restaurants, one restaurant can also have multiple menu items to choose from.

The only unique relationship in this ER diagram is between the Customer and Order. While one customer can make multiple orders, each order must only be uniquely identified for only one customer. This forms a -one-to-many relationship between the two entities.

Part B:

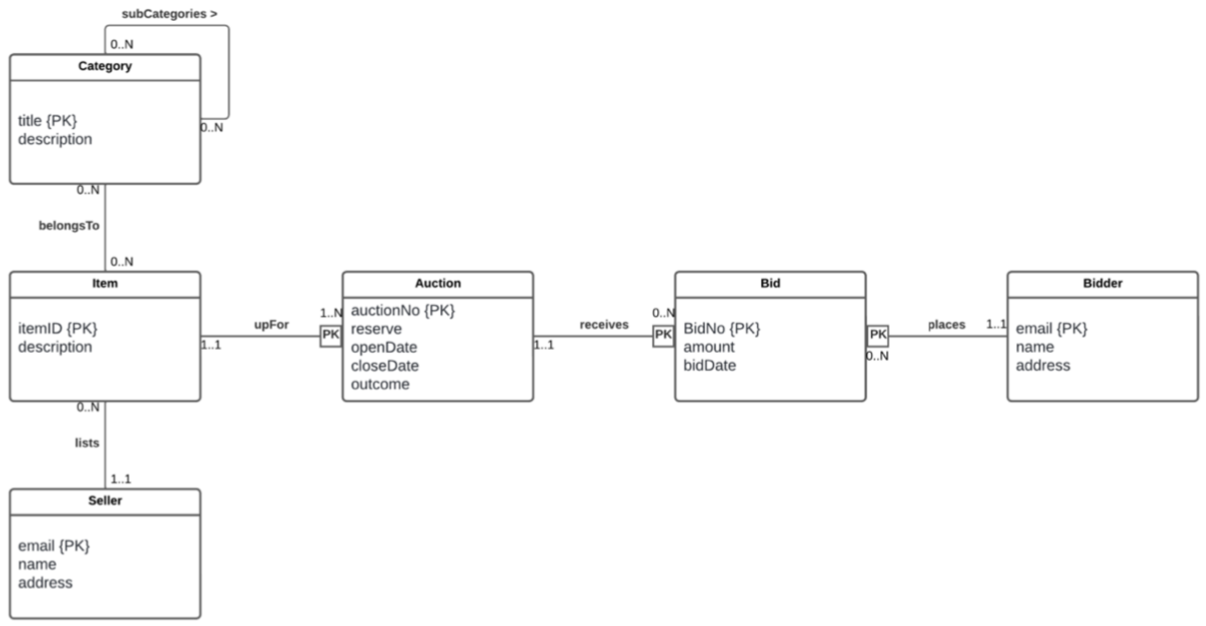


To hold multiple menu items in one order, we can add Menu_Item_Name from Menu Item (refer back to Name) to Order as a foreign key. With this we have two more entities added, which are Delivery and Driver. The relationship between Order and Delivery is one-to-one, while the relationship between Driver and Delivery is on-to-many. Driver in this case is a weak entity and depends on Delivery. If there is no order of delivery, we will not need a driver, and one driver can only deliver to one customer per delivery order.

Rates relationships are created between Customer and Driver and Customer with Order. While the relationship attribute of Rates between Customer and Driver is only score, those attributes of Customer and Order are Score and Menu_Item_Name to specify the score for each menu item.

To enforce ordered items to be from one restaurant, we limit the relationship between Restaurant and Menu Item to one-to-many, where one restaurant can prepare several menu items, however; each menu item can only be limited to one restaurant.

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



Step 1: Map Strong Entities
Category (<u>title</u>, description)
Item (<u>itemID</u>, description)
Seller (<u>email</u>, name, address)
Bidder (<u>email</u>, name, address)

Step 2: Map Weak Entities
Auction (<u>itemID*, auctionNo</u>, reserve, openDate, closeDate, outcome)
Bid (<u>itemID*, auctionNo*, email*, BidNo</u>, amount, bidDate)

Step 3: Map One-to-one Relationships
No one-to-one relationship

Step 4: Map One-to-many Relationships
Item (<u>itemID</u>, description, email*)

Step 5: Map Many-to-many Relationships
belongsTo (<u>title*, itemID*</u>)

Step 6: Multi-valued Attributes
No multi-valued attributes

Step 7: Map Higher-degree Relationships
No higher-degree relationship

Special Case: subCatagories
Step 5 is applied. Add subCatTitle as a foreign key in Category

Final Schema:
Category (title, description, subCatTitle*)
Item (itemID, description)
Seller (email, name, address)
Bidder (email, name, address)
Auction (itemID*, auctionNo, reserve, openDate, closeDate, outcome)
Bid (itemID*, auctionNo*, email*, BidNo, amount, bidDate)
Item (itemID, description, email*)
belongsTo (title*, itemID*)

Task 4: Relational Database Model

4.1
Yes, it does. Because Employee.empjob_id refers back to Job.job_id, which is a primary key of Job, and cannot be NULL.

4.2
Make Department.department_ID and Department.manager_ID a composite primary key. With that, you can have more managers in the same department, and can also assign different locations using Department.location_Id.

4.3
It almost fulfills the requirement since all departments are not expanding and divided into sub departments, so we can treat the additional departments as other departments. However, just adding more departments does not satisfy the fact that the three new departments' managers are reported to one director. Department.director_Id may need to be added to identify the role in this case.

In addition, by changing the definition of department_ID = 3 to the new department name, this will also apply to the current employee under this dapartment_ID. Adjusting the current HR department resource may need to be done later if this approach is implemented.

4.4
We have to be careful when *WHERE first_name='Adam'* is stated because employee_id = 12 is also using *'Adam'* as first_name. Better set the condition with employee_id as it is the primary key. Also, considering changing the job, We cannot be sure if there is anything to do with the current salary since it has been stated as a promotion. And also, since Jonny Deans is also a programmer and his department_id is 1, this should also be the same case for Adam Smith.

I am not sure if *contracts* relate to the Job History entity. If that is the case, then it can be done no matter if Adam Smith's job has been changed or not. This is because we can select the record using employee_id, which has nothing to do with updating Adam Smith's job. However, we may not find any records since the entity only contains records for Jonny Deans and Adam Jones.

4.5
This will affect the Departments entity as it uses location_id as its foreign key. Row 2 of Locations entity will be deleted, and the value of location_id in row 2 of the Departments entity will be NULL.

4.6

```
CREATE TABLE Employees (
                        employee_id INTEGER PRIMARY KEY,
                        first_name VARCHAR(10) NOT NULL,
                        last_name VARCHAR(20) NOT NULL,
                        phone_number INTEGER NOT NULL,
                        hire_date INTEGER NOT NULL,
                        empjob_id INTEGER NOT NULL,
                        salary INTEGER NOT NULL,
                        department_id INTEGER NOT NULL
                        );
```

4.7

```
CREATE TABLE Departments (
                        department_id INTEGER PRIMARY KEY,
                        department_name VARCHAR(20) NOT NULL,
                        employee_id INTEGER,
                        CONSTRAINT manager_id
                          FOREIGN KEY (employee_id)
                          REFERENCES Employee(employee_id)
                        location_id INTEGER,
                        CONSTRAINT location_id
                          FOREIGN KEY (location_id)
                          REFERENCES Locations(location_id)
                        );
```

4.8

```
INSERT INTO Departments (department_id, department_name, manager_id, location_id)
VALUES (4, ‘Cyber Security’, ‘Johnny Deans’, 30);

UPDATE Employees
SET job_id = 45, department_id = 4
WHERE employee_id = 10;

INSERT INTO Job History (employee_id, start_date, job_id, department_id)
VALUES (10, ‘01/01/2010’, 45, 4);
```