Coursework Submission

Part A & B (Full)

Tomasz Wasowski – w1684891

Tutors: Francois Roubert, Tasos Ptohos

GROUP: 5CS05; 5CS06 – 4:00pm to 6:00pm (Tuesday)

5COSC002W – Database Systems

# FOODTOOYOO - Conceptual ERD

# Data Dictionaries

## Entities

|  |  |
| --- | --- |
| Entity Name | Description |
| w1684891\_Member | A ‘Member’ is a customer that is registered with the FOODTOOYOO business. |
| w1684891\_Store | A ‘Store’ represents a retailer that is registered with the business and offers products for members to purchase. |
| w1684891\_Product | A ‘Product’ is an item offered by a retailer that is available to be purchased by members. |
| w1684891\_Basket | A ‘Basket’ is what a member creates in order to shop with a specific store. It is what will contain the members chosen products for that particular store. |
| w1684891\_Order | An ‘Order’ is what a customer places once they have confirmed and finalised their basket and would like to check out. |
| w1684891\_Payment | A ‘Payment’ represents the transaction a member makes to pay for their shopping order. |
| w1684891\_Driver | A ‘Driver’ is a person contracted by FOODTOOYOO to collect orders and deliver them to members. |
| w1684891\_Vehicle | A ‘Vehicle’ is what is used by the driver to make collections and deliver orders. |
| w1684891\_Collection | A ‘Collection’ is the process of a driver picking up an order (or multiple) from a store. |
| w1684891\_Delivery | A ‘Delivery’ represents the process of a driver delivering an order to a member. |

|  |  |  |
| --- | --- | --- |
| General Entity | Specialised Entity | Description |
| w1684891\_Product | w1684891\_Perishable | A ‘Perishable’ product is one that has an expiry date and cannot be stored indefinitely. |
| w1684891\_Perishable | w1684891\_Food | A ‘Food’ item represents a perishable product of solid substance; one for which it is necessary to store the ingredient description and allergy information. |
| w1684891\_Perishable | w1684891\_Drink | A ‘Drink’ item represents a perishable product of liquid substance; one with a specified type and sugar content. |
| w1684891\_Drink | w1684891\_Alcoholic | An ‘Alcoholic’ item represents a drink which contains some percentage of alcohol. |
| w1684891\_Payment | w1684891\_CardPayment | A ‘Card Payment’ is a payment made specifically with a credit/debit card. |
| w1684891\_Payment | w1684891\_MobilePayment | A ‘Mobile Payment’ represents a digital payment made via a mobile wallet. |
| w1684891\_Vehicle | w1684891\_TwoWheel | A ‘Two Wheel’ vehicle is one that has a two wheels and is only capable of making smaller collections. |
| w1684891\_Vehicle | w1684891\_FourWheel | A ‘Four Wheel’ vehicle is one that has a four wheels; better suited to making larger collections. |
| w1684891\_Collection | w1684891\_SmallCollection | A ‘Small Collection’ is a collection for an order (or multiple) small enough to be picked up by a two-wheeled driver. |
| w1684891\_Collection | w1684891\_LargeCollection | A ‘Large Collection’ is a collection for an order (or multiple) that requires a four-wheeled driver to collect it. |

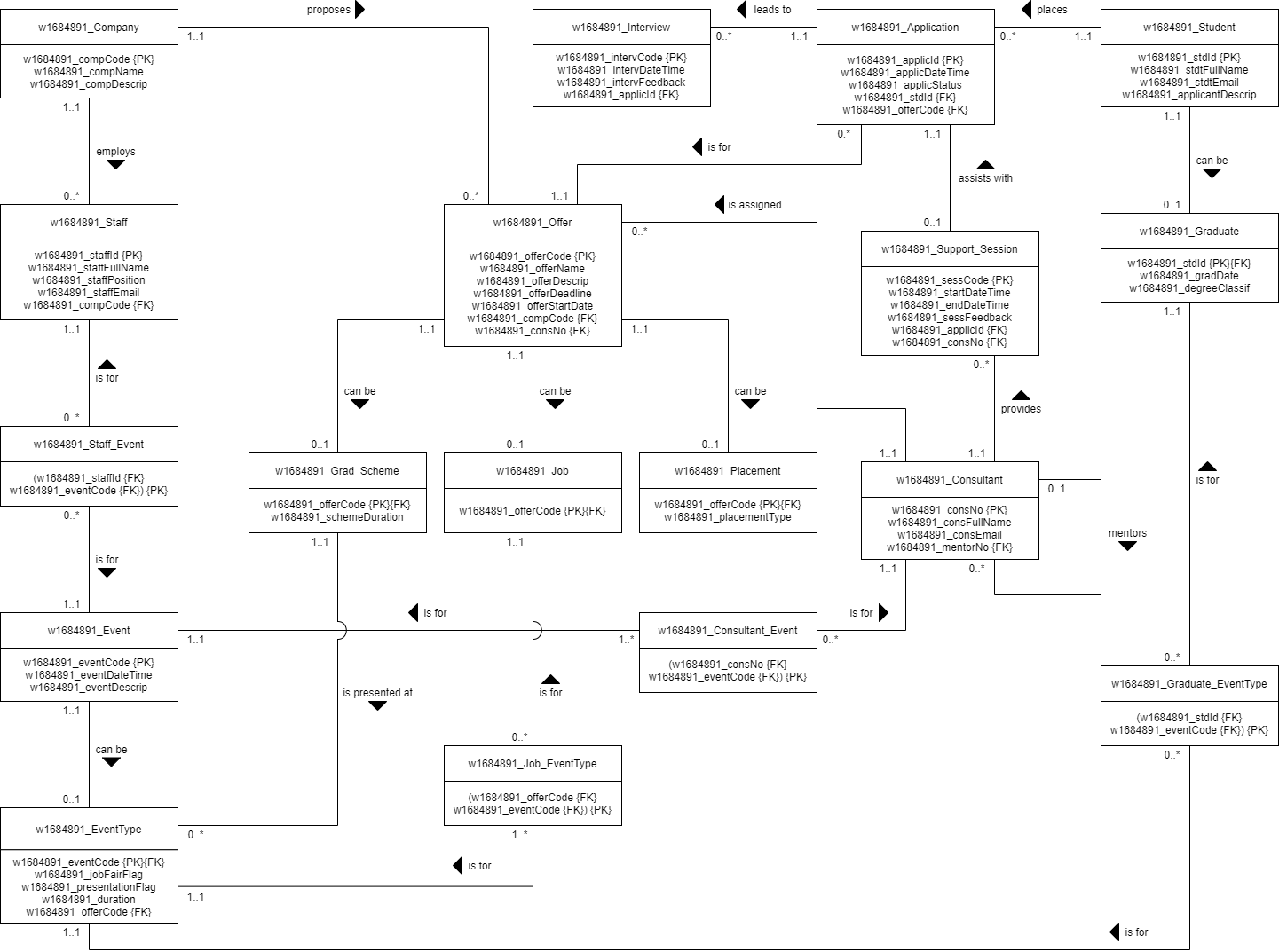
## Relationships and Multiplicities

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Entity | Multiplicity | Relationship | Multiplicity | Entity | Justification |
| w1684891\_Member | 1..1 | has | 0..\* | w1684891\_Basket | A single member might not necessarily have a basket. |
| A single member could have many baskets. |
| A single basket must belong to a member, as basket that does not belong to a client would be redundant. |
| A single basket cannot belong to more than one registered member, as each member creates their own baskets. |
| w1684891\_Basket | 0..\* | created with | 1..1 | w1684891\_Store | A single basket will be created for at least one store. A basket created for no store would have no purpose. |
| A single basket can only be created for one store. The brief specifies that a basket cannot be used between different stores. |
| A single store might not have any baskets created with it; it could be a newly registered store. |
| A single store might have many baskets created with it. |
| w1684891\_Store | 1..1 | offers | 0..\* | w1684891\_Product | A single store might not offer any products yet. It could be only just registered or even recently closed. |
| A single store might be offering many products for sale. |
| A single product has to be offered by at least one store; if a product were not offered by anyone then there would be no way to sell it. |
| A single product can only be offered by a single store, as the brief specifies that the stores sell unique, branded products. |
| w1684891\_Order | 0..1 | placed for | 1..1 | w1684891\_Basket | A single order cannot be placed for 0 baskets. If there was no basket, there would be no need to place an order in the first place. |
| A single order can only be placed for one basket; as specified by the managing director. |
| A single basket might not necessarily have on order placed for it yet. It might not be finalised yet, as the member is still in the process of shopping. |
| A single basket can only have one order placed for it; as specified by the managing director. |
| w1684891\_Order | 1..1 | dispatched for | 0..1 | w1684891\_Delivery | A single order might not necessarily be dispatched for delivery yet. It could still be pending for confirmation. |
| A single order might be dispatched for only one delivery. A single order cannot be delivered twice. |
| A single delivery will consist of at least one order. It makes no sense to record a delivery of nothing. |
| A single delivery will be recorded for a maximum of one order, as it is a record of the delivery of that specific order. |
| w1684891\_Payment | 0..1 | processed for | 1..1 | w1684891\_Order | A single payment that is made must be processed for at least one order. A payment cannot be processed without an order to pay for. |
| A single payment cannot be processed for more than one order. As specified by the managing director, each payment can only be made for a single order. |
| A single order might not necessarily be paid for yet. The payment could still be pending. |
| A single order will only have one payment placed for it. As specified by the managing director, the payment cannot be split and must be made in full for the order. |
| w1684891\_TwoWheel | 1..1 | makes | 0..\* | w1684891\_SmallCollection | A single two-wheeled driver might not make any small collections yet. Could be a new driver. |
| A single two-wheeled driver could make many small collections; they are contracted for that purpose. |
| A single small collection will be made by at least one two-wheeled driver. A collection cannot be made by nobody. |
| A single small collection can only be made by one two-wheeled driver. A single collection cannot be made by multiple drivers. |
| w1684891\_FourWheel | 1..1 | makes | 0..\* | w1684891\_LargeColleciton | A single four-wheeled driver might not make any collections yet. Similarly to above, could be a new driver. |
| A single four-wheeled driver might make many collections as they are contracted to do just that. |
| A single large collection must be made by at least one four-wheel driver. A collection cannot be picked up by no one. |
| A single large collection will be made by only one four-wheeled driver. A collection cannot be made by multiple drivers. |
| w1684891\_Collection | 0..1 | made for | 1..\* | w1684891\_Order | A single collection is made for at least one order. If there were no orders to pick up, there would be no need for a collection. |
| A single collection can handle many orders, as specified in the FOODTOOYOO brief. |
| A single order might not necessarily have a collection made for it yet. It could still be a pending order. |
| A single order will only have one collection made for it. It can only be picked up from the store once. |
| w1684891\_Collection | 0..\* | picked up at | 1..1 | w1684891\_Store | A single collection will be picked up at at least one store. You cannot make a collection from nowhere. |
| A single collection will not be picked up at more than one store; that would require multiple different collections. |
| A single store might have no collections picked up at it. It could be a brand-new store, or one that has not managed to sell anything yet. |
| A single store might have many collections made at it for the many orders that customers will place while buying from it. |
| w1684891\_Driver | 1..1 | owns | 1..\* | w1684891\_Vehicle | A single driver has to own at least one vehicle to make his collections/deliveries. A collection or delivery cannot be made without a vehicle. |
| A single driver could own many different vehicle which they might use to make collections/deliveries. |
| A single vehicle will be owned by at least one driver. It makes no sense for the business to store details of a vehicle owned by nobody. |
| A single vehicle will only be owned by a maximum of one driver, as drivers sign up to FOODTOOYOO with their own vehicles. |

## Attributes & Primary Keys

|  |  |  |
| --- | --- | --- |
| Entity Name | Attributes for this Entity | Justification |
| w1684891\_Member | w1684891\_memberID {PK} | An identification number assigned to each registered member that will help to uniquely identify them. This would be the primary key for this entity. |
| w1684891\_Member | w1684891\_memberFullName | A composite attribute made from the members first name and last name. It may be important to store their names separately, as well as together, for different levels of formality in different types of correspondences. |
| w1684891\_Member | w1684891\_memberFirstName | One of the attributes that makes up the full name. This will store just a member’s first name. |
| w1684891\_Member | w1684891\_memberLastName | Another attribute that makes up a member’s full name. This will store the member’s last name. |
| w1684891\_Member | w1684891\_memberDOB | This attribute will hold the members date of birth. Necessary to store as some of the products that can be purchased may require an age check. |
| w1684891\_Member | /w1684891\_memberAge | An attribute derived from the memberDOB which is a more convenient way to store the member age for above mentioned checks. |
| w1684891\_Member | w1684891\_memberAddress | A composite attribute containing the members full home address. It may be important to keep the full address for ease of deliveries, and to have it split for efficiently assigning drivers to make these deliveries. |
| w1684891\_Member | w1684891\_memberStreet | One of the attributes that make up the full address. This attribute stores the street name that the user lives on. |
| w1684891\_Member | w1684891\_memberCity | Another attribute that makes up the full address. This attribute stores the use’s home city. |
| w1684891\_Member | w1684891\_memberPostcode | Another attribute that makes up the full address. This one stores the member’s home postcode. |
| w1684891\_Member | w1684891\_memberEmailAddress | This attribute stores the email address that a member has registered with. |
| w1684891\_Member | w1684891\_memberPhoneNo | This attribute stores a member’s preferred phone number, in case there is need for direct contact. |
| w1684891\_Member | w1684891\_memberRegDate | This attribute stores a member’s date and time of registration with the FOODTOOYOO business. |
| w1684891\_Member | w1684891\_memberRegStatus | This attribute will store a member’s registration status, like whether their account has been created, their email address confirmed, or whether they have deactivated their account. |
| w1684891\_Store | w1684891\_storeID {PK} | This attribute is an identification number generated for each registered retail store, used to uniquely identify them within the FOODTOOYOO system. This attribute is the primary key for the store entity. |
| w1684891\_Store | w1684891\_storeName | This attribute contains the name of the retailing store. |
| w1684891\_Store | w1684891\_storeDescription | This attribute contains the description text for a store and could be used to identify what the store specialises in, or what sort of products it might offer. |
| w1684891\_Store | w1684891\_storeMgrFullName | This is a composite attribute that contains the store manager’s full name. Similarly to member, it may be a good idea to store the name in full, or separately, for different levels of correspondence formality. |
| w1684891\_Store | w1684891\_storeMgrFirstName | Attribute containing the store manager’s first name. One of the attributes that makes up the store manager’s full name. |
| w1684891\_Store | w1684891\_storeMgrLastName | Attribute containing the store manager’s last name. One of the attributes that makes up the store manager’s full name. |
| w1684891\_Store | w1684891\_storeAddress | Similar to a member’s full address. This is an attribute that is composite and stores a store’s full address together. |
| w1684891\_Store | w1684891\_storeStreet | This attribute contains the street name that the store is located on. It is one of the attributes that makes up the full address. |
| w1684891\_Store | w1684891\_storeCity | This attribute contains the city that the store is located in. It is one of the attributes that makes up the full address. |
| w1684891\_Store | w1684891\_storePostcode | This attribute contains the store’s post code. It is one of the attributes that makes up the full address. |
| w1684891\_Store | w1684891\_storeEmailAddress | The email address that the store is registered with. |
| w1684891\_Store | w1684891\_storePhoneNo | This is the main phone number that the registered store operates under. |
| w1684891\_Store | w1684891\_storeWebsite | This attribute contains a link to the store’s website address. |
| w1684891\_Store | w1684891\_storeRegDate | This attribute stores the date and time at which the retailer has registered with the FOODTOOYOO business. |
| w1684891\_Store | w1684891\_storeRegStatus | This attribute contains the registration status of the retailing store. For example, if the store is only just signed up, if it is certified and confirmer or if it has been deactivated. |
| w1684891\_Product | w1684891\_productID {PK} | This attribute is the primary key for this entity. It is a unique identification number for each product. |
| w1684891\_Product | w1684891\_productName | This attribute stores the name of this branded product. |
| w1684891\_Product | w1684891\_productDesc | This attribute stores the description of the product provided by the retailer. |
| w1684891\_Product | w1684891\_productWeight | This attribute stores the products weight. It is additional information about the product that might be crucial in determining the order size. |
| w1684891\_Product | w1684891\_productDimensions | This attribute stores the products physical dimensions. Another piece of additional information that may be crucial in determining the order size. |
| w1684891\_Product | w1684891\_productPrice | This attribute stores the sale price of a product. |
| w1684891\_Product | w1684891\_productStockLevel | This attribute stores the level of available stock for a particular product. |
| w1684891\_Product | w1684891\_productManuDate | This attribute stores the date of manufacture of a product. |
| w1684891\_Perishable | w1684891\_perishableExpiryDate | This attribute stores the expiry date of products which are perishable. |
| w1684891\_Food | w1684891\_foodIngredientDesc | This attribute stores the description of ingredients that a specific food consists of. |
| w1684891\_Food | w1684891\_foodAllergyInfo | This attribute stores information about the any allergens that this food might contain. |
| w1684891\_Drink | w1684891\_drinkType | This attribute stores the type of the drink product, as required by the brief. |
| w1684891\_Drink | w1684891\_drinkSugarContent | This attribute stores the sugar contents of a specific drink product. |
| w1684891\_Alcoholic | w1684891\_alcoholByVolume | This attribute stores the alcohol by volume contained in an alcoholic drink. |
| w1684891\_Basket | w1684891\_basketID {PK} | This attribute is the primary key for this entity. It stores a unique identification number for every basket. |
| w1684891\_Basket | w1684891\_basketCreationDate | This attribute stores a basket’s creation date and time. |
| w1684891\_Basket | w1684891\_basketStatus | This attribute stores the status of a basket. For example, whether the user is still in the process of shopping with this basket, or whether it is confirmed and ready for an order to be placed for it. |
| w1684891\_Order | w1684891\_orderID {PK} | This is the primary key for this entity. It stores a unique identification number for each order. |
| w1684891\_Order | w1684891\_orderTotal | This attribute stores the total price of the order that the member has to pay. |
| w1684891\_Order | w1684891\_orderPlacementDate | This attribute stores the date and time that the order was placed. |
| w1684891\_Order | w1684891\_orderConfirmationDate | This attribute stores the date and time of when the order is confirmed. |
| w1684891\_Order | w1684891\_orderStatus | This attribute stores the status of the order, such as pending, confirmed, in transit, delivered. |
| w1684891\_Payment | w1684891\_paymentID {PK} | This is the primary key for the payment entity. It is a unique identification number for each payment made. |
| w1684891\_Payment | w1684891\_paymentAmount | This is the total amount that this payment is made for. |
| w1684891\_Payment | w1684891\_paymentDate | This is the date and time at which the payment is processed. |
| w1684891\_Payment | w1684891\_paymentStatus | This attribute stores that status of the payment, such a pending, processed, or cancelled. |
| w1684891\_CardPayment | w1684891\_cardSecurityCode | This attribute stores the security of the credit/debit card that this payment is made with. |
| w1684891\_MobilePayment | w1684891\_mobileWallet | This attribute stores the information of the mobile wallet that was used to make this payment. |
| w1684891\_Delivery | w1684891\_deliveryID {PK} | This attribute is the primary key for this entity. A unique identification number given to each deliver made. |
| w1684891\_Delivery | /w1684891\_deliveryExpectedDate | This is the expected delivery date and time which is calculated upon order confirmation. |
| w1684891\_Delivery | w1684891\_deliveryActualDate | This attribute stores the actual date that the delivery took place. |
| w1684891\_Delivery | w1684891\_deliveryStatus | This attribute contains the status of the delivery i.e. in transit, delivered. |
| w1684891\_Driver | w1684891\_driverID {PK} | This is the primary key for this entity. This attribute stores the unique identification number created for each driver registered with FOODTOOYOO. |
| w1684891\_Driver | w1684891\_driverFullName | This is a composite attribute that stores the full name of the driver. |
| w1684891\_Driver | w1684891\_driverFirstName | This is one of the attributes that makes up the driver’s full name. It stores the driver’s first name. |
| w1684891\_Driver | w1684891\_driverLastName | This attribute stores the driver’s last name and is the other attribute that makes up the driver’s full name. |
| w1684891\_Driver | w1684891\_driverDOB | This attribute is the drivers date of birth. It is important to store this to ensure that the driver is of legal driving age. |
| w1684891\_Driver | /w1684891\_driverAge | This attribute is derived from the driverDOB and stores the driver’s age for the convenience of checking whether they fit the legal driving age. |
| w1684891\_Driver | w1684891\_driverAddress | This is a composite attribute that stores the driver’s full home address. It might be necessary to store this for the efficiency of driver assignment. |
| w1684891\_Driver | w1684891\_driverStreet | This attribute stores the driver’s home street and is used as one of the attributes to make up the full address. |
| w1684891\_Driver | w1684891\_driverCity | This attribute is the driver’s home city and is also an attribute used to comprise the full address. |
| w1684891\_Driver | w1684891\_driverPostcode | This attribute stores the driver’s postcode and is also an attribute which makes up the full address. |
| w1684891\_Driver | w1684891\_driverEmailAddress | This attribute stores the email address that the driver has registered with. |
| w1684891\_Driver | w1684891\_driverPhoneNo | This attribute stores the driver’s preferred phone number for any essential contact. |
| w1684891\_Driver | w1684891\_driverContractDate | This attribute stores the drivers contract date and time to track when the driver has started working with the business. |
| w1684891\_Driver | w1684891\_driverHourlyRate | This attribute stores the driver’s agreed-upon hourly rate. |
| w1684891\_Driver | w1684891\_driverLicenseNo {AK} | This attribute stores the driver’s license number. It could also be used as an alternate key to uniquely identify the driver. |
| w1684891\_Driver | w1684891\_driverLicenseExpDate | This attribute stores the expiry date of the driver’s license, to ensure that the driver can legally drive. |
| w1684891\_Vehicle | w1684891\_vehicleID {PK} | The primary key for the vehicle entity. It is a unique identification number given to each vehicle registered with the business. |
| w1684891\_Vehicle | w1684891\_licensePlateNo | If the vehicle has a license plate number, it will be stored here. Might be necessary for vehicles such as cars, vans or motor bikes. |
| w1684891\_Vehicle | w1684891\_vehicleManuDate | If relevant, the vehicles manufacture date will be stored here. Might be necessary for vehicles such as cars, vans, or motor bikes. |
| w1684891\_Vehicle | w1684891\_vehicleRegDate | This will store the date and time that the vehicle has been registered with the FOODTOOYOO business. |
| w1684891\_Vehicle | w1684891\_vehicleInsuranceNo | If applicable, this attribute will store the vehicles insurance number. |
| w1684891\_Colleciton | w1684891\_collectionID {PK} | This is the primary key for the collection entity. It is a unique identification number assigned to every collection that is made. |
| w1684891\_Colleciton | w1684891\_collectionDate | This attribute stores the date of when a collection is made. This is useful to track the status of orders. |
| w1684891\_Colleciton | w1684891\_collectionTime | This attribute stores the time of when a collection is made. |
| w1684891\_Colleciton | w1684891\_collectionStatus | This is the status of a collection; could be prepared or collected. |
| w1684891\_Colleciton | /w1684891\_amountOfOrders | This is an attribute derived from the orders linked to the collection. It specifies the amount of orders included in this collection. |

# Futuro Logical ERD



# Futuro Logical ERD Mapping – Step-by-Step Guide

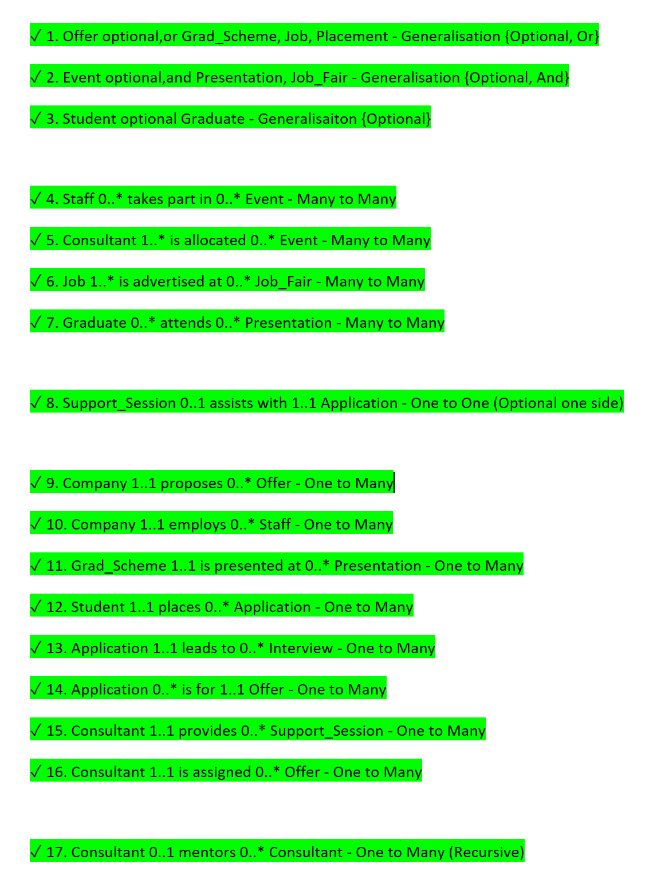


Figure 3 - List of relationships to resolve.

1. This is not one of the 10 mapping rules, but a short explanation of how I began the logical mapping process. Firstly, I’ve re-created all the entities from the conceptual ERD in draw.io (diagrams.net). Secondly, I’ve listed out all the relationships and specialisations/generalisations and ordered them by how disruptive they are (according to Lecture 4). Below I will list and describe each rule that I’ve implemented, in the order of implementation. As I applied the rules, I have re-created the resolved relationships between the recreated entities, using the Conceptual ERD as reference. As a result, there was no need to adjust any existing relationships between applying the rules.
2. **Rule 10 Generalisation with {Optional, Or}** – *Offer* specialises into: *Grad\_Scheme, Job, Placement* - I decided to resolve all the specialisations/generalisations first, as they are usually the most disruptive. Offer was specialised into the most sub-entities and had the most relationships, so I decided to resolve it first. I’ve applied rule 10, by making Offer the parent entity, and creating 3 child entities for Grad\_Scheme, Job and Placement. I’ve used the PK from Offer to act as the PK and FK for each of the child entities, while keeping their original attributes as they were. The children have been connected with a ‘can be’ relationship to the parent as follows: Offer 1..1 can be 0..1 ChildEntity.
3. **Rule 8 Generalisation with {Optional, And}** – *Event* specialises into: *Presentation, Job\_Fair* – As the specialisation/generalisation with the second most sub-entities and relationships, I decided to resolve this one next. I’ve applied rule 8, by making Event the parent entity, while creating a single child entity called EventType for both of the specialised sub-entities Presentation and Job\_Fair. I’ve used the parent PK as the PK and FK of the child entity, and created flag attributes for Presentation and Job\_Fair, while adding the attributes of both specialisation sub-entities into the child entity. Finally, I’ve created a ‘can be’ relationship between the parent and child entities as follows: Event 1..1 can be 0..1 EventType.
4. **Rule 10 Generalisation with {Optional, And}** – *Student* specialises into: *Graduate* – This specialisation had the least amount of sub-entities and relationships involved, so I chose to resolve it after all the other generalisations. As this generalisation is with {Optional} without and ‘or’ or an ‘and’, it doesn’t really matter whether we use rule 10 or rule 8, as they should both provide a similar resolution. As such, I’ve applied rule 10, making Student the parent entity and Graduate the child entity. As before, I’ve used the parent PK and the child PK and FK, while keeping all the original attributes in the child entity. As before, I linked these entities with a ‘can be’ relationship as follows: Student 1..1 can be 0..1 Graduate.
5. **Rule 5 Many-to-Many** – *Staff* 0..\* takes part in 0..\* *Event* – As there were no one-to-one mandatory or complex relationships, I moved directly to resolving all many-to-many relationships. I started in no particular order for these, and just began with the relationship between Staff and Event. I started by applying rule 5 to create an additional entity called Staff\_Event, making it the child entity to both Staff and Event. The child entity has both parent PKs as FKs, and a compound PK created from a combination of these FKs. I decided to go with a compound PK, as opposed to composite with an additional attribute, as I assume that a single staff member will either take part in an event or won’t, so there will not be any repeatable records. I’ve linked the child entity with a 1..1 to many (keeping the participations of this many from the original relationship) relationships ‘is for’ to the parent entities.
6. **Rule 5 Many-to-Many** – *Consultant* 1..\* is allocated 0..\* *Event –* Moving onto the next many-to-many relationship that links the Consultant and Event entities, I’ve once again applied rule 5 to resolve it similarly as in the step above. I created a new entity called Consultant\_Event which is a child to both Consultant and Event. I once again used a compound PK for the child that is made from a combination of it’s parents PKs, which act as FKs in the child. It makes sense for a consultant to either be allocated to an event or not, so I decided that a composite PK was not necessary. As above, I linked the child to it’s parents with ‘is for’ relationships as follows: Consultant\_Event (\_..\*) is for 1..1 Parent. (Many participations kept from original relationship).
7. **Rule 5 Many-to-Many** – *Job* 1..\* is advertised at 0..\* *Job\_Fair(EventType) –* Tackling the next many to many between Job and Job\_Fair (which has now become EventType within a previous step), I’ve once again applied rule 5. A new entity called Job\_EventType was created as the child of both Job and EventType. Once again, I chose a compound PK for the child, as a single Job either will or will not be advertised at a single event. As such, I see no reason to use a composite key. Relationships between the child and both parents were created in a similar fashion to those in the steps above.
8. **Rule 5 Many-to-Many** – *Graduate* 0..\* attends 0..\* Presentation*(EventType) –* Moving onto the final many to many relationship, between Graduate and Presentation (which has also now become EventType in a previous step), I am once again applying rule 5. In a similar fashion to the 3 steps above, a child entity called Graduate\_EventType is created in between the parents. This entity also uses a compound key as opposed to a composite one, as a single graduate student either attends a presentation or doesn’t.
9. **Rule 3 One-to-One Optional on One Side** – *Support\_Session* 0..1 assists with 1..1 *Application* – Moving on to 1:1 relationships that are optional or one or both sides, I am applying rule 3 to the above relationship. As such, I am simply making the entity on the ‘mandatory’ side of the relationship the parent, so in this case it is Application, and the entity on the ‘optional’ side the child, so in this case Support\_Session. As such, I am putting the parent entity’s PK as a FK in the child entity.
10. **Rule 1 One-to-Many** – Company 1..1 proposes 0..\* Offer – Having resolved all of the more disruptive relationships, we can move onto our One-to-Many relationships. Resolving these in no particular order, I begin with the one between Company and Offer. I apply the first rule and in this case Company, on the side of the one, is the parent and Offer, on the side of the many, is the child. As a result, I simply add the PK of the parent entity as a FK in the Child entity.
11. **Rule 1 One-to-Many** – Company 1..1 employs 0..\* Staff – Another one-to-many relationships for which I’m applying rule 1, Company being the parent and Staff being the child. As such, the PK of Company goes into Staff as a FK.
12. **Rule 1 One-to-Many** – Grad\_Scheme 1..1 is presented at 0..\* Presentation (EventType) – Once again using rule one to make Grad Scheme the parent and Presentation (now EventType) into the child. The PK from the parent becomes a FK in the child entity. In this case, since Grad\_Scheme was a specialisation, its PK is also the same as the Offer entity’s PK.
13. **Rule 1 One-to-Many** – Student 1..1 places 0..\* Application – Once again applying rule 1 to resolve this relationship. Student is the parent entity and Application is the child. The PK of the Student entity becomes a FK in the Application entity.
14. **Rule 1 One-to-Many** – Application 1..1 leads to 0..\* Interview – Another use of rule 1. Application is the parent entity while Interview, on the side of the many, is the child entity. Interview gains a FK referencing the PK in Application.
15. **Rule 1 One-to-Many** – Application 0..\* is for Offer 1..1 – Once again applying rule 1. Offer becomes the parent entity while Application is the child entity, gaining a FK attribute equivalent to the PK of the Offer entity.
16. **Rule 1 One-to-Many** – Consultant 1..1 provides 0..\* Support\_Session – Another application of rule 1. Consultant, on the side of the one, being the parent and Support\_Session, on the side of the many, being the child. The PK from Consultant becomes a FK for Support\_Session.
17. **Rule 1 One-to-Many** – Consultant 1..1 is assigned 0..\* Offer – Applying rule 1, I made Consultant the parent entity and Offer the child entity. The PK of Consultant becomes a FK in Offer.
18. **Rule 1 One-to-Many** – Consultant 0..1 mentors 0..\* Consultant – The final application of rule 1. In this case, the relationship is recursive and only consists of one entity. This entity simultaneously becomes the parent and the child. As such, its own PK will become a FK for it. I decided to name this FK as mentorNo, which essentially references consNo (the PK).