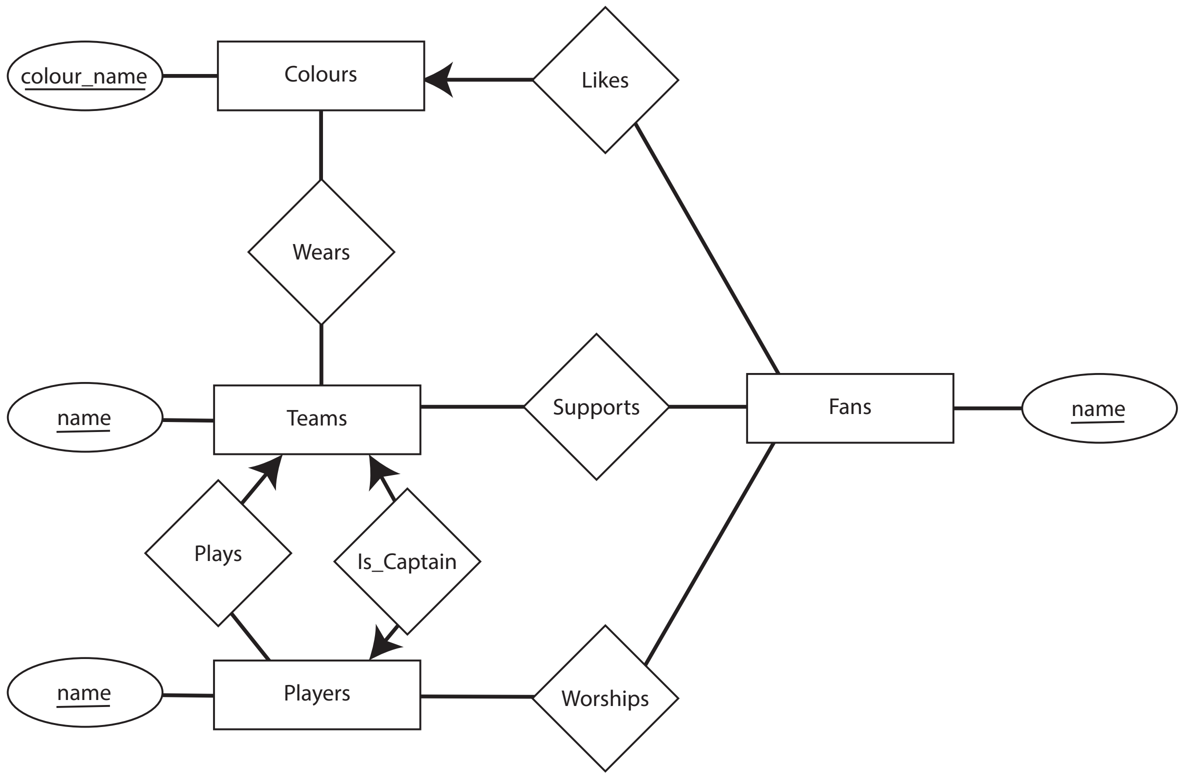
UNIVERSITY OF VICTORIA, DEPARTMENT OF COMPUTER SCIENCE

Assignment 2: Solution Set

CSC 370, Fall 2013

October 17, 2013

Question 1 (20 marks)



Note that in the diagram above, I have already put in keys for the entity sets. Also shown are some arities. Strictly speaking these are items requested in Question 3, and so they need not appear in the solution for Q1.

Question 2 (20 marks)

(a) *Redundancy*: The address of the owner appears twice – once in AccSets and another time in Addresses.

(b) *Simplicity*: The entity AccSets does not really seem to do anything useful here. The design can be redrawn such that we instead have a many-to-many relationship between Customers and Accounts.

(c) *Right kind of element*: The entity set Addresses has a single attribute, itself named *address*. As a customer cannot have more than one address, it makes more sense to have this as an attribute in the Customer entity set.

(d) *Model's fidelity to the real world*: Customers cannot be uniquely identified by their names. In the real world they would be identified by such unique attributes as a Social Insurance Number or by a customerID.

Question 3 (20 marks)

- Keys have already been indicated on the diagram of Question 1.
- Some additional constraints could be indicated by changing the arity of certain relationships. For example: A team must have a captain, and exactly one captain. This could be reflected by changing the filled arrow from *is_captain* to *Teams* with a hollow arrow.
- Other constraints are implied in the diagram from Question 1 (i.e, zero or one). For example, *Players* need not be associated with a team (i.e. free agents). Similarly teams (perversely) might not have players if the team is no longer in operation.

Question 4 (30 marks)

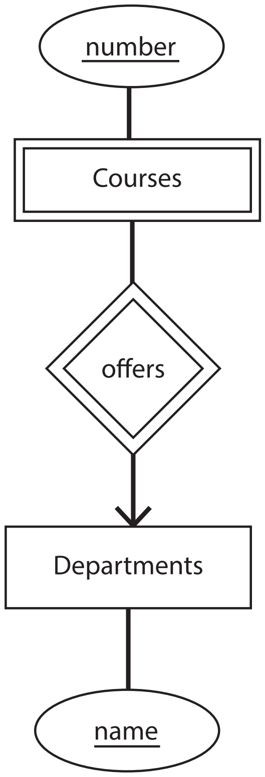
(a) The keys for both E_1 and E_2 are required in order to uniquely identify tuples in R.

(b) We need only the key for E_1 in order to determine the identity of the entity in set E_2.

(c) We need only the key for E_2 in order to determine the identity of the entity in set E_1.

(d) If we have the key to either of E_1 or E_2, we can always determine the identity of the other item from the other set. Therefore either key will help.

Question 5 (25 marks)



Question 6 (25 marks)

There may be other answers. Here is one solution.

- `Customers(SIN, name, addr, phone)`
- `Flights(number, day, aircraft)`
- `Bookings(customerSIN, flightNumber, flightDay, row, seat)`

Note that the relations for the weak entities (`toCust` and `toFlt`) are not required as the entity set `Bookings` already contains the keys for entities in sets `Customers` and `Flights`.

Question 7 (30 marks)

(a) Straight E-R method: Since `FatherOf` and `MotherOf` are many-to-one relationships from `Child`, there is no need for a separate relation for them. Similarly the one-to-one relationship `Married` can be included in `Father` (or `Mother`). `ChildOf` is a many-to-many relationship and therefore will need its own separate relation. *However*, the `ChildOf` relation is not actually required since the relationship can be deduced from `FatherOf` and `MotherOf` relationships contained in the `Child` relation itself.

```
Person(name,address)
ChildOf(personName,personAddress,childName,childAddress)
Child(name,address,fatherName,fatherAddress,motherName,motherAddresss)
Father(name,address,wifeName,wifeAddresss)
Mother(name,address)
```

(b) O-O method: The many-to-many `ChildOf` does require a relation here. Recall, also, that in the OO approach an entity belongs to one – and only one – class. Therefore the many-to-one relations `MotherOf` and `FatherOf` could be added as attributes to `PersonChild`, `PersonChildFather`, and `PersonChildMother` relations. (Note how the ISA path is indicated in the relation names.) In a similar way the `Married` relation can be added as attributes to `PersonChildMother` and `PersonMother` (or the corresponding father relations). *Your answer will depend upon many of these kinds of assumptions, and hence will not necessarily match the relations below.*

```
Person(name,address)
PersonChild(name,address)
PersonChildFather(name,address)
PersonChildMother(name,address)
PersonFather(name,address)
PersonMother(name,address)

ChildOf(personName,personAddress,childName,childAddress)
FatherOf(childName,childAddress,fatherName,fatherAddress)
MotherOf(childName,childAddress,motherName,motherAddress)
Married(husbandName,husbandAddress,wifeName,wifeAddress)
```

(c) For the `Person` relation, at least one of the husband and wife attributes may be null (depending upon the way you interpret legal definitions of marriage).

```
Person(personName,personAddress,fatherName,fatherAddress,
     motherName,motherAddresss,wifeName,wifeAddresss,
     husbandName,husbandAddress)
ChildOf(personName,personAddress,childName,childAddress)
```

Question 8 (30 marks)

```
class Colours(key(colourname)){
              attribute string colourname;
              relationship Set<Fans> LikedBy
                inverse Fans::Likes;
              relationship set<Teams> WornBy
                inverse Teams::Wears;
            };

class Teams(key(name)){
            attribute string name;
              relationship set<Colours> Wears
                inverse Colours::WornBy;
              relationship set<Players> PlayedBy
                inverse Players::Plays;
              relationship PLayers CaptainedBy
                inverse Platyers::Captains;
              relationship set<Fans> SupportedBy
                inverse Fans::Supports;
            };

class Players(key(name)){
              attribute string name;
              relationship Set<Teams> Plays
                inverse Teams::PlayedBy;
              relationship Teams Captains
                inverse Teams::CaptainedBy;
              relationship Set<Fans> WorshippedBy
                inverse Fans::Worships;
            };

class Fans(key(name)){
              attribute string name;
              relationship Colours Likes
                inverse Colours::LikedBy;
              relationship Set<Teams> SupportedBy
                inverse Teams::Supports;
              relationship Set<Players> Worships
                inverse Players::WorshippedBy;
            };
```

As collections are permitted in ODL, a Colours Set can become an attribute of Teams!