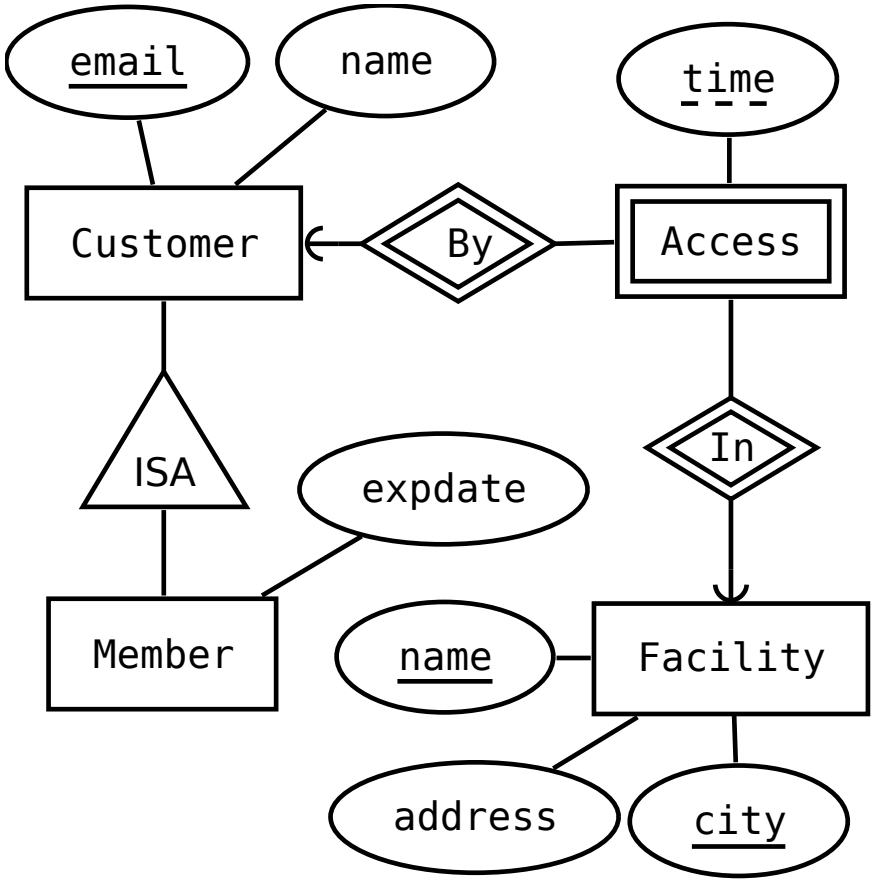
1 a)



Note (not part of answer): Assumes customers have unique emails (which seems reasonable). Adding a synthetic key (like an ID-number) is OK.

1b)

```
Courses(code, name)

Instances(course, semester, capacity)
  course -> Courses.code

Teachers(ssn, name)

Teaches(teacher, course, semester)
  teacher -> Teachers.ssn
  (course, semester) -> Instances(course, semester)

Examiner(examiner, course, semester)
  (examiner,course,semester) -> Teaches(teacher,course,semester)
```

Note (not part of answer): Additional references in Examiner are allowed but not required.

2 a) {a,c,e} {b,c,e} {d,c,e}
Note (not part of answer): Set notation ({,}) is not required as long as it is clear which attributes are in each key.

2 b) {b,c,d} {a,b,c,d} {a,b,d,e}

2 c) Example:

a	b	c	d	e
0	0	0	0	0
0	0	0	1	1

Note (not part of answer): b and c must be the same in both rows, and d differ to violate the functional dependency. Column a is essentially irrelevant and e must be different to respect the keys. Having additional rows is OK as long as the keys are respected.

Another correct solution where values are "as unique as possible":

a	b	c	d	e
0	1	2	3	4
5	1	2	6	7

3 a)

```
SELECT username, email, COUNT(follower) AS total_followers
FROM Users LEFT OUTER JOIN Follows ON username=follows
GROUP BY username, email
```

3 b) One simple solution:

```
SELECT (follower, follows) FROM Follows
EXCEPT
SELECT (follows, follower) FROM Follows
```

3 c)

```
WITH JonasFollowers AS SELECT follower FROM Follows WHERE follows='jonas'

SELECT follower FROM JonasFollowers
UNION
SELECT follower FROM Follows WHERE follows IN (SELECT follower FROM JonasFollowers)
```

Note (not part of answer): Union deletes duplicates automatically. A self-join could be used instead of the subquery. The WITH-clause is not required, but reduces code size slightly.

4 a)

$$\delta(\pi_{(name,\ idnr)}(\sigma_{(minute > 90)}(Players\ X_{(player=idnr)}\ Goals)))$$

Note (not part of answer): The sigma could also be placed on Goals or in the join condition (but the latter is discouraged)

4 b)

$$\gamma_{(birthmonth,\ COUNT(*))}(Players\ X_{(player=idnr)}\ Goals)$$

4 c) One solution:

$$\pi_{G1.idnr}(\rho_{G1}(Goals)\ X_{G1.player=G2.player\ AND\ G1.game=G2.game\ AND\ G1.minute=G2.minute\ AND\ G1.goalNumber\ !=\ G2.goalNumber}\ \rho_{G2}(Goals))$$

Note (not part of answer): An aggregate + select works as well.

Note (not part of the answer): Since the question was slightly unclear, solutions that allow goals from different games are also OK if done correctly ($G1.goalNumber\ !=\ G2.goalNumber$ OR $G1.game\ !=\ G2.game$)

5)

```
CREATE TABLE Customers(
  id INT PRIMARY KEY, -- Any numeric type is OK
  name TEXT,
  isPrivate BOOLEAN,
  UNIQUE (id, isPrivate) -- c - Not needed for full points
  );

CREATE TABLE Subscriptions
  (number INT PRIMARY KEY,
   customer INT,
   isPrivate BOOLEAN,                                    -- c
   plan TEXT,
   fee INT,
   balance INT,
   FOREIGN KEY (customer, isPrivate)                     -- c
     REFERENCES Customers(id,isPrivate)
     ON DELETE CASCADE,                                  -- e
   CHECK (plan IN ('prepaid','corporate','flatrate')),   -- a
   CHECK (plan='prepaid' OR balance=0),                  -- b
   CHECK (plan!='corporate' OR NOT isPrivate),           -- c
   CHECK (fee >= 0)                                      -- d
   );

CREATE VIEW CustomerView AS                              -- d
  SELECT id, name, Customers.isPrivate, SUM(fee)
  FROM Customers JOIN Subscriptions ON id=customer
  GROUP BY id,name,Customers.isPrivate;


CREATE FUNCTION deleteEmpty() RETURNS trigger AS $$
BEGIN
  IF NOT EXISTS (SELECT *
                 FROM Subscriptions
                 WHERE customer = OLD.customer) THEN
    DELETE FROM Customers WHERE id=OLD.customer;
  END IF;
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER deleteEmpty                               -- f
  AFTER DELETE ON Subscriptions
  FOR EACH ROW
  EXECUTE PROCEDURE deleteEmpty();
```

Note (not part of answer): Using an assertion is acceptable for c)

6

a)

```
[
  {"category":"Starters",
   "contents":[
     {"dish":"Calamari", "price":8.50}
   ]
  },
  {"category":"Salads",
   "contents":[
     {"dish":"Caesar",  "price":8.50},
     {"dish":"Chicken", "price":9.25}
   ]
  },
  {"category":"Burgers",
   "contents":[
     {"dish":"Standard", "price":9},
     {"dish":"Bacon",    "price":10},
     {"category":"Vegetarian Burgers",
       "contents":[
         {"dish":"Haloumi",  "price":12},
         {"dish":"Mushroom", "price":10}
       ]
     }
   ]
  }
]
```

b)

```
{"type":"array",
 "items":{
   "type":"object",
   "oneOf":[
     { "properties":{
         "category":{"type":"string"},
         "contents":{"$ref":"#"},
         "dish":false,
         "price":false
         },
       "required":["category","contents"]
     },
     { "properties":{
         "category":false,
         "contents":false,
         "dish":{"type":"string"},
         "price":{"type":"number"}
         },
         "required":["dish","price"]
     }]
 }
}
```

c)

```
$[?(@.category=="Burgers")]..price
```