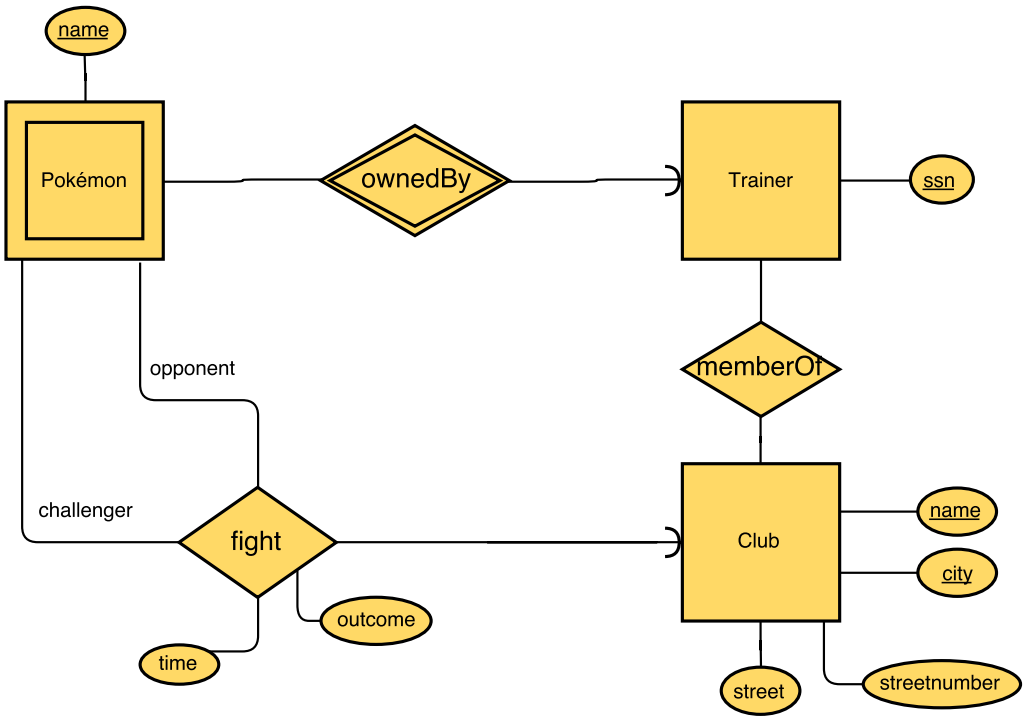
Databases Exam HT2016 Solution

Solution 1a



Solution 1b

```
Trainer(ssn)
Pokemon(ssn, name)
    ssn -> Trainer.ssn
Club(name, city, street, streetnumber)
MemberOf(ssn, name, city)
    ssn -> Trainer.ssn
    (name, city) -> Club.(name, city)
Fight(ssn1, name1, ssn2, name2, name, city, time, outcome)
    (ssn1, name1) -> Pokemon.(ssn, name)
    (ssn2, name2) -> Pokemon.(ssn, name)
    (name, city) -> Club.(name, city)
```

Solution 2a

```
(1)  star, name           → * (all other attributes)
(2)  star, position       → * (all other attributes)
(3)  star, distance       → * (all other attributes)

(4)  radius               → area
(5)  area                 → radius

(6)  water                → land
(7)  land                 → water

(8)  mass, radius         → gravity
(9)  mass, gravity        → radius
(10) gravity, radius      → mass

(11) atmosphere, oxygen   → otherGas
(12) atmosphere, otherGas → oxygen
(13) oxygen, otherGas     → atmosphere
```

Solution 2b

These 3 subsets of attributes of the Planets relation are keys:

```
{star, name}
{star, position}
{star, distance}
```

A key is a minimal superkey. Each of these subsets is a superkey of the relation Planets because their closure is the full set of attributes of Planets. In addition, each of these superkeys is minimal because there is no subset of attributes that is also a superkey.

Solution 2c

Functional dependencies 4-13 violate BCNF (all FDs except the first 3), because the left-hand side of each of these FDs is not a superkey of the Planets relation.

Solution 2d

Step 1

FD 4 violates BCNF, so we create a new relation **Areas** and remove the **area** attribute from the **Planets** relation. After this step, the violation of FD 5 is automatically resolved.

```
Planets(* - area)
    remaining FDs: 1-3, 6-13
Areas(radius, area)
    radius -> Planets.radius
    (4) radius → area
    (5) area   → radius
```

Step 2

FD 6 violates BCNF, so we create a new relation **Surfaces** and remove the **land** attribute from the **Planets** relation. After this step, the violation of FD 7 is automatically resolved.

```
Planets(* - area - land)
    remaining FDs: 1-3, 8-13
Areas(radius, area)
    radius -> Planets.radius
    (4) radius → area
    (5) area   → radius
Surfaces(water, land)
    water -> Planets.water
    (6) water → land
    (7) land  → water
```

Step 3

FD 8 violates BCNF, so we create a new relation **Gravities** and remove the **gravity** attribute from the **Planets** relation. After this step, the violation of FD 9 and 10 is automatically resolved.

```
Planets(* - area - land - gravity)
    remaining FDs: 1-3, 11-13
Areas(radius, area)
    radius -> Planets.radius
    (4) radius → area
    (5) area   → radius
Surfaces(water, land)
    water -> Planets.water
    (6) water → land
    (7) land  → water
Gravities(mass, radius, gravity)
    mass, radius -> Planets.(mass, radius)
    (8)  mass, radius    → gravity
    (9)  mass, gravity   → radius
    (10) gravity, radius → mass
```

Step 4

FD 11 violates BCNF, so we create a new relation **Atmospheres** and remove the **otherGas** attribute from the **Planets** relation. After this step, the violation of FD 11 and 12 is automatically resolved.

```
Planets(name, star, position, distance, radius, water, mass, atmosphere,
   oxygen)
    (1) star, name     → * (all other attributes)
    (2) star, position → * (all other attributes)
    (3) star, distance → * (all other attributes)
Areas(radius, area)
    radius -> Planets.radius
    (4) radius → area
    (5) area   → radius
Surfaces(water, land)
    water -> Planets.water
    (6) water → land
    (7) land  → water
Gravities(mass, radius, gravity)
    mass, radius -> Planets.(mass, radius)
    (8)  mass, radius    → gravity
    (9)  mass, gravity   → radius
    (10) gravity, radius → mass
Atmospheres(atmosphere, oxygen, otherGas)
    atmosphere, oxygen -> Planets.(atmosphere, oxygen)
    (11) atmosphere, oxygen   → otherGas
    (12) atmosphere, otherGas → oxygen
    (13) oxygen, otherGas     → atmosphere
```

Solution 3a

```
CREATE TABLE Planets(
    star TEXT NOT NULL,
    name TEXT NOT NULL,
    distance FLOAT NOT NULL CHECK(distance > 0),
    mass FLOAT NOT NULL CHECK(mass > 0),
    atmosphere BOOLEAN NOT NULL,
    oxygen FLOAT NOT NULL CHECK((oxygen = 0 and not atmosphere) OR (
       atmosphere AND oxygen >= 0 AND oxygen <= 100)),
    water FLOAT NOT NULL CHECK(water >= 0 AND water <= 100),
    PRIMARY KEY(star, name),
    UNIQUE(star, distance)
);
```

Solution 3b

```
SELECT COUNT(*) FROM Planets WHERE
        distance > (SELECT distance FROM Planets WHERE
                    star='Kerbol' AND name='Duna');
```

Solution 3c

```
(SELECT star, name, 'habitable' FROM Planets WHERE
        distance >= 100 AND distance <= 200 AND
        atmosphere AND oxygen >= 15 AND oxygen <= 25 AND
        water > 0)
UNION
(SELECT star, name, 'uninhabitable' FROM Planets WHERE NOT(
        distance >= 100 AND distance <= 200 AND
        atmosphere AND oxygen >= 15 AND oxygen <= 25 AND
        water > 0));
```

or

```
WITH habitables AS (SELECT star, name FROM planets WHERE
        distance >= 100 AND distance <= 200 AND
        atmosphere AND oxygen >= 15 AND oxygen <= 25 AND
        water > 0)
SELECT star, name, 'habitable' FROM Planets WHERE
        (star, name) IN (SELECT star, name from habitables)
UNION
SELECT star, name, 'unhabitable' FROM Planets WHERE
        (star, name) NOT IN (SELECT star, name from habitables);
```

or

```
SELECT star, name, CASE WHEN
        distance >= 100 AND distance <= 200 AND
        atmosphere AND oxygen >= 15 AND oxygen <= 25 AND
        water > 0
        THEN 'habitable' ELSE 'uninhabitable' END
    FROM Planets;
```

Solution 4a

The query in SQL:

```
SELECT star, SUM(mass) AS totalMass FROM Planets WHERE atmosphere GROUP BY
   star HAVING COUNT(*) > 5;
```

The query in relational algebra:

$\pi_{star,totalMass}(\sigma_{atmosphere\&planetcount>5}(\gamma_{star,COUNT(*)\rightarrow planetcount,SUM(mass)\rightarrow totalMass}(Planets)))$

Solution 4b

```
SELECT position, MAX(gravity) AS maxg
        FROM (P NATURAL JOIN G)
        GROUP BY position
        ORDER BY maxg;
```

Solution 5a

```
CREATE VIEW PromotionSummary AS
        SELECT category, MIN(price) AS minprice, MAX(price) AS maxprice FROM
           Books
        WHERE promoted
        GROUP BY category;
```

Solution 5b

```
CREATE OR REPLACE FUNCTION demoteBooks() RETURNS TRIGGER AS $$
BEGIN
        UPDATE Books SET promoted = False WHERE category = OLD.category;
END
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER demoteBooksTrigger INSTEAD OF DELETE ON PromotionSummary
    FOR EACH ROW
    EXECUTE PROCEDURE demoteBooks();
```

it is acceptable to shorten this to:

```
demoteBooks() → UPDATE Books SET promoted = False WHERE category = OLD.
   category;

CREATE TRIGGER demoteBooksTrigger INSTEAD OF DELETE ON PromotionSummary
    FOR EACH ROW
    EXECUTE PROCEDURE demoteBooks();
```

Solution 6a

Alice has too many privileges, since she does not need to read the password in the Users table, nor the LogBook entries. The minimally required set of permissions is:

```
GRANT SELECT(id, name) ON Users TO Alice;
GRANT SELECT(id, loggedin) ON UserStatus TO Alice;
GRANT INSERT(id, timestamp, name) ON LogBook TO Alice;
```

Solution 6b

Yes this code contains an SQL injection vulnerability.

The vulnerability can be removed by either correctly sanitizing or escaping the data in the `userinput` variable. A better solution is to use a `PreparedStatement` with placeholder:

```
...
String query = "SELECT * FROM UserStatus WHERE id = ?";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString(1, userinput);
ResultSet rs = stmt.executeQuery();
...
```

Solution 6c

The transaction is vulnerable to "non-repeatable read" and "phantom read" interferences, because the `READ COMMITTED` transaction isolation level does not protect against them. The stronger `REPEATABLE READ` isolation level is not sufficient because it still allows phantom reads. Only the `SERIALIZABLE` isolation level is sufficient, since it protects against dirty read, non-repeatable read and phantom read.