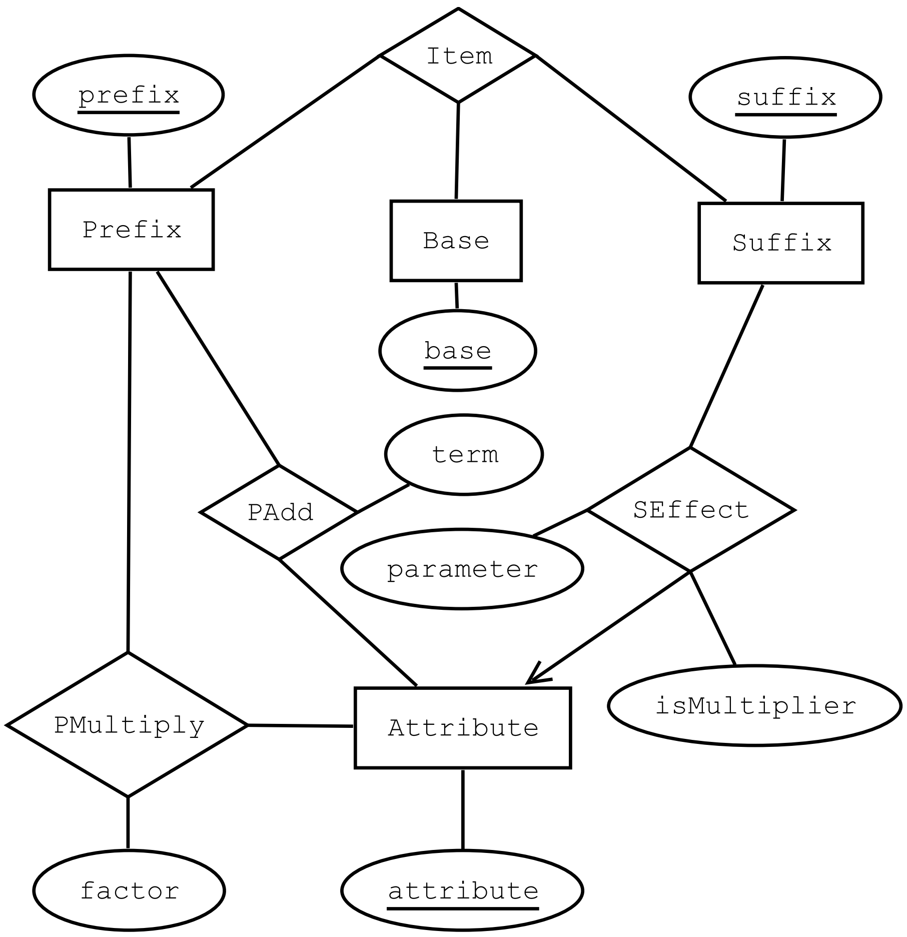
Question 1:

a) One possible solution, perhaps the simplest one that satisfies all criteria:



I use “ismultiplier” as to indicate if parameter is a factor or a term. It could be split up, but then suffixes could have more than one effect.

You could have an Effect entity with multiplier and/or addition as subentities.

b)

Prefix(prefix)
Base(base)
Suffix(suffix)
Attribute(attribute)

PMultiply(prefix, attribute, factor)
  prefix -> Prefix.prefix
  attribute -> Attribute.attribute

PAdd(prefix, attribute, term)
  prefix -> Prefix.prefix
  attribute -> Attribute.attribute

SEffect(suffix, attribute, isMultiplier, parameter)
  suffix -> Suffix.suffix
  attribute -> Attribute.attribute

c)

Prefix:
(Wonderful)

Base:
(Armor)

Suffix:
(The Rainbow)

Attribute:
(Wisdom)
(Charisma)

PMultiply:
(Wonderful, Wisdom, 1.03) – or 0.03 or 3 or really any value that makes sense

PAdd:
(Wonderful, Wisdom, 3)

SEffect:
(The Rainbow, Charisma, FALSE, 10)

Question 2:

a)

A	B	C	D
1	1	2	2
1	1	3	3
4	4	2	2
5	5	5	2

In the table above I mostly use new numbers except when there's a specific reason not to. A and B doesn't always have to be identicalLi though (e.g. I could have used (1,2), (2,3) and (3,4) instead of (1,1), (4,4) and (5,5)).

There are *a lot* of other correct solutions (this one will be fun grading…). The first two lines in this solution makes sure that there are no dependencies from A and/or B to C or D. The third line does the same in the other direction and the last just makes sure that D -> C doesn't hold.

b)

a c
b c

Question 3:

```
-- a)

SELECT student, SUM(COALESCE(filesize,0))
FROM Submission LEFT OUTER JOIN SubmittedFile ON submission=idnr
GROUP BY student;

-- b)
-- There are lots of ways to solve this, here I use a correlated query
--   to count the number of files named tables.sql and views.sql in each
--   submission (should be two only if both files are present since
--   duplicate filenames cannot exist within submissions)
-- There are also lots of incorrect ways of solving this, like most things
--   you can do with joins
SELECT *
FROM Submission AS Sub
WHERE
  course = 'TDA357' AND assignment='Lab 1'
  AND  2 > (SELECT COUNT(*) FROM SubmittedFile
            WHERE submission = Sub.idnr
                  AND (filename = 'tables.sql' OR filename = 'views.sql'))
  ;
```

Test data for your own solution:

```
CREATE TABLE Assignment
    ( course TEXT
    , name TEXT
    , description TEXT
    , deadline INT
    , PRIMARY KEY (course, name)
    );

CREATE TABLE Submission
    ( idnr INT PRIMARY KEY
    , student TEXT
    , course TEXT
    , assignment TEXT
    , stime INT
    , FOREIGN KEY (course, assignment) REFERENCES Assignment(course, name)
    );

CREATE TABLE SubmittedFile
    ( submission INT
    , filename TEXT
    , filesize INT
    , contents TEXT
    , FOREIGN KEY (submission) REFERENCES Submission
    , PRIMARY KEY (submission, filename)
    );

INSERT INTO Assignment VALUES ('TDA357', 'Lab 1', '...', 2500);
INSERT INTO Assignment VALUES ('TDA357', 'Lab 2', '...', 6000);
INSERT INTO Assignment VALUES ('TDA144', 'Lab 2', '...', 2000);

INSERT INTO Submission VALUES (0, 's1', 'TDA357', 'Lab 1', 1000);
INSERT INTO Submission VALUES (1, 's2', 'TDA357', 'Lab 2', 2000);
INSERT INTO Submission VALUES (2, 's1', 'TDA357', 'Lab 1', 3000);
INSERT INTO Submission VALUES (3, 's3', 'TDA144', 'Lab 2', 1000);

INSERT INTO SubmittedFile VALUES (1, 'tables.sql',  1300, '...');
INSERT INTO SubmittedFile VALUES (1, 'comment.txt', 100,  '...');
INSERT INTO SubmittedFile VALUES (2, 'views.sql',   800,  '...');
INSERT INTO SubmittedFile VALUES (2, 'tables.sql',  900,  '...');
```

Question 4:

a)

$\sigma_{\text{stime > deadline AND Submissions.course = Assignment.course AND assignment = name}}$
(Submissions × Assignment)

b)

$\sigma_{\text{A1.idnr > A2.idnr AND A1.stime < A2.stime}}$
$(\rho_{A1}(\text{Assignment}) \times \rho_{A2}(\text{Assignment}))$

c)

$\gamma_{\text{student, course, assignment, MAX(idnr) -> idnr}}(\text{Submissions})$

Question 5:

```
CREATE TABLE Registration
  ( student TEXT
  , course TEXT
  , PRIMARY KEY (student, course)
  );

CREATE TABLE Assignment
    ( course TEXT
    , name TEXT
    , description TEXT
    , deadline INT -- Should probably be a TIMESTAMP or similar
    , PRIMARY KEY (course, name)
    );

CREATE TABLE Submission
    ( idnr INT PRIMARY KEY -- Should probably be a SERIAL
    , student TEXT
    , course TEXT
    , assignment TEXT
    , stime INT
    , FOREIGN KEY (course, assignment) REFERENCES Assignment(course, name)
    , FOREIGN KEY (course, student) REFERENCES Registration(course, 
student) -- b)
    , UNIQUE (student, stime) -- f)
    );

CREATE TABLE SubmittedFileTable
    ( submission INT
    , filename TEXT
    , contents TEXT
    , FOREIGN KEY (submission) REFERENCES Submission ON DELETE CASCADE-- d)
    , PRIMARY KEY (submission, filename)
    );


-- c) Using a view, an alternative is to use DEFAULT + a check constraint 
or something like that.
CREATE VIEW SubmittedFile AS
  SELECT *, length(contents) AS filesize FROM SubmittedFileTable;

-- a) needs a trigger on INSERT OR UPDATE on Submission that either
--    automatically adjusts the idnr or rejects updates that violate the
--    rule. Alternatively, a) can be solved using an assertion with a
--    NOT EXISTS query corresponding to the RA expression in 4b
-- e) needs to be done using a trigger AFTER DELETE for
--    SubmittedFile(Table) that checks if the last file was deleted and if
--    so run DELETE FROM Submission WHERE idnr = OLD.submission
--    to be completely safe it probably needs to be AFTER DELETE OR UPDATE
```

Question 6:

```
{
    "splitAt": 24,
    "branch1": {
        "splitAt": 20,
        "branch1": {
            "splitAt": 17,
            "branch1": "Blue shark",
            "branch2": "Caribbean reef shark"
        },
        "branch2": {
            "splitAt": 10,
            "branch1": "Bull shark",
            "branch2": "Blacknose shark"
        }
    },
    "branch2": "Lemon shark"
}
```

This solution only includes positions for splits, thus assuming the length of the bar for the root of the tree does not need to be specified. Every branch is either just a String or or another split. This one requires splitAt to be decreasing further down the tree.