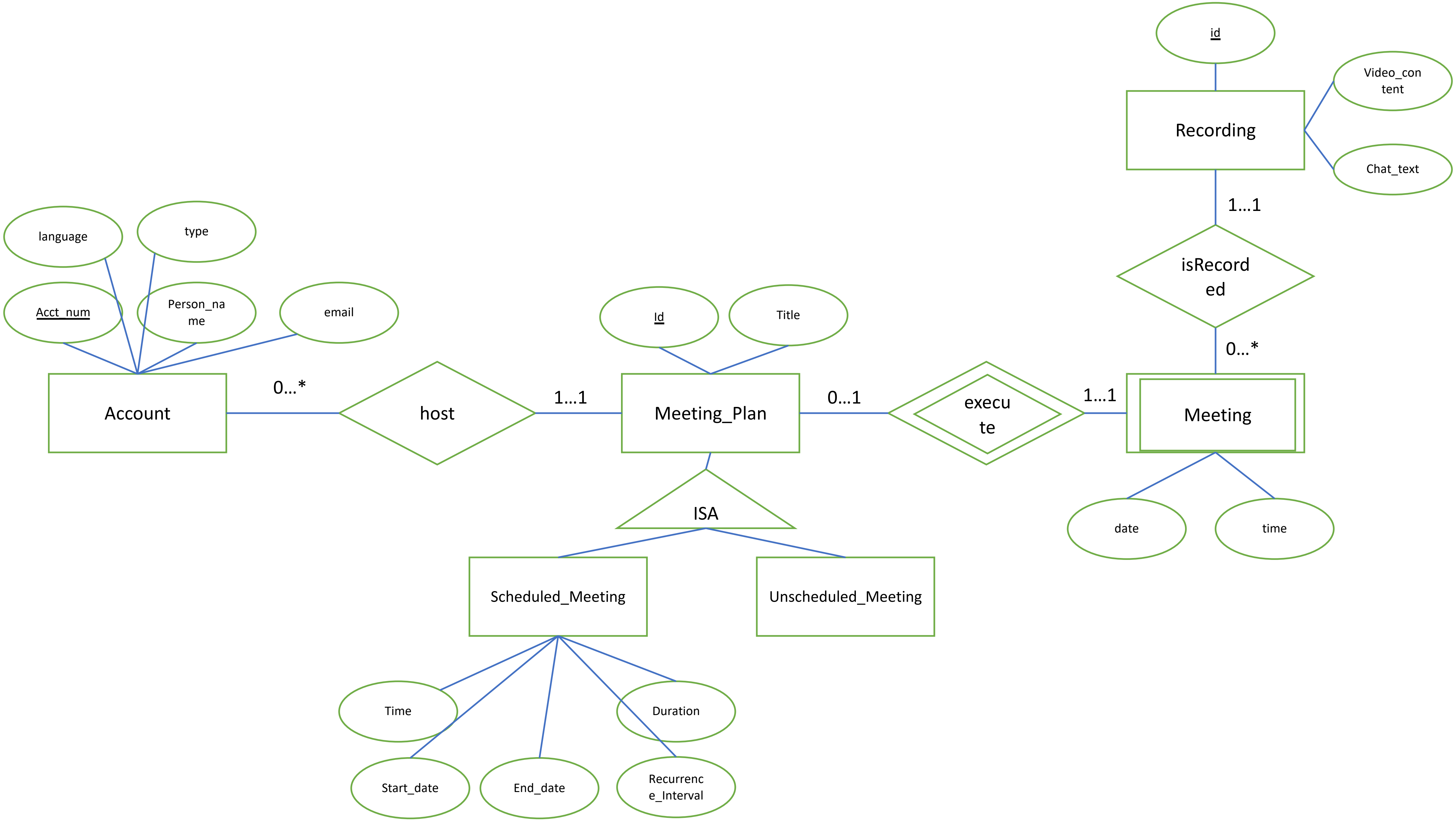

id
Video_content
Recording
Chat_text
1...1
isRecorded
0...*
language
type
Acct_num
Person_name
email
Id
Title
Account
0...*
host
1...1
Meeting_Plan
0...1
execute
1...1
Meeting
ISA
date
time
Scheduled_Meeting
Unscheduled_Meeting
Time
Duration
Start_date
End_date
Recurrence_Interval

Q1-B

- Not it doesn’t capture the fact that these should be correct.
- This can be guaranteed, we can create a relationship set between scheduled meeting and meeting instance to check if meeting datetime falls under the scheduled datetime sets.

Q2

Guest (Id, Name, Email)

Property (Id, Type, Address, MaxGuests, OwnerId)
OwnerId FOREIGN KEY REFERENCES Host (Id)
OwnerId NOT NULL

Stay (Date, PropertyId, PaymentInfo, NumGuests, GuestId, GuestComments, GuestScore, HostId, HostComments, HostScore)
PropertyId FOREIGN KEY REFERENCES Property (Id)
GuestId FOREIGN KEY REFERENCES Guest (Id)
GuestId NOT NULL
HostId FOREIGN KEY REFERENCES Host (Id)

Host (Id, Name, Email, SuperHost)

Actually, in Stay, the HostId is also NOT NULL, but this has been handled through Property + Stay.

Q3

A: Yes, the participation of hosts in the owns relationship is 0..*, and therefore it can owns multiple properties with different property ids

B: No, the stay is uniquely represented by Id + Date, and the participation of stay in books relationship is a 1..1, so therefore only a single guest can book the unique stay.

C: Yes. The participation of stay in G_Rate relationship is a 0..1, and therefore a stay could be rated by either 0 or 1 guest. And the participation of stay in books relationship is a 1..1, so there has to be a guest who booked this specific stay. There is no restriction to prohibit such scenarios as long as the comment/score for a stay is at most 1 record.

So interesting! It seems there can be the same person, how to prevent this from happening????

Q4

- A: Yes
- B: Yes
- C: No
- D: No, it can’t. We can simply use a counter example to prove this is incorrect. Take B as an constant value while A and C are not constant variable, and A->C. In this case, AB->C always satisfies but B->C won’t be.

Q5

- A

(A,B), (A,D), (B,C), (C,D), because (A,C), (B,D) cannot functionally determines all attributes and no single attributes can functionally determines all attributes as well. So since these four pairs could, there are the candidate keys.

- B

Yes, for each of the X->Y in F, X is a superkey for R.

- C

Yes, for each of the X->Y in F, Y is a member of some key.

Q6

- A

(C,D,A), (C,D,E), (C,D,B), since C,D is not in a right side of any dependency relation, so C,D must present in the keys. However, only C and D cannot do anything.

For (C,D,A), A->B, BC->E;

For (C,D,E), ED->A, A->B;

For (C,D,B), BC->E, ED->A;

- B

Yes, because for every X->Y in F, Y is a member of some key.

- C

Fr1 = {A->B}

Fr2 = {A,C->E}

Fr3 = {}

- D

No, ED->A is lost.

Q7

- A

(A,E), since E doesn't show up in F, and A never appears in the right side of any relation in F. So A and E are must-have in the candidate keys. A->B, A->C, and C->D, so A->D, which means A, E can cover all attributes.

- B

1. Lift C->D into a table of its own, R1(C,D), Fr1 = {C->D}, keep C in original table, remove D, now original table is: R(A,B,C,E)
2. Lift A->C into a table of its own, R2(A,C), Fr2 = {A->C}, , keep A in original table, remove C, now original table is: R(A,B,E)
3. Lift A->B into a table of its own, R3(A,B), Fr3 = {A->B}, , keep A in original table, remove B, now original table is: R(A,E)

We get the following decomposition, R1, R2, R3, and R. This way, it preserves dependencies since we lift all dependencies in the process.

- C

First we can confirm that this is a minimal equivalent set. We can output a relation for every dependency. AB, AC, CD, together with AE. And the relation AE contains a key (A,E), so we are done.