3.2 How many distinct tuples are in relation to an instance with cardinality 22?

- Cardinality is defined as the number of tuples in a relation instance.
- Since each tuple must have a primary key, this implies that the # of distinct tuples is exactly 22 as well.
    - Note that primary key means that, for a chosen set of fields in the relation, no two tuples may have the same values for those fields.
    - So there are 22 primary keys (and hence 22 distinct tuples) because this is enforced by the primary key rule.

3.4 What is the difference between a candidate key and the primary key for a given relation? What is a superkey?

- A candidate key is a minimal set of fields that uniquely identifies a tuple.
    - name, dob
    - ssn
    - name, address

^ These may be some possible minimal set of fields that uniquely identifies a tuple.

- A primary key means that in our relation instance, we pick one of the possible candidate keys (set of fields) to represent a distinct tuple.
- Here we may pick ssn to be the primary key, or we may pick (name, age) to be the primary key.

- Primary key is only one of the possible distinct identifiers, candidate key can be many such minimal set of fields to uniquely identify a tuple.

- A superkey on the other hand is NOT a minimal set of fields to uniquely identify a tuple.
- A superkey may still be used to identify a unique tuple such as Superkey (ssn, address).
    - But a superkey is a set of fields, in which some subset is also a candidate key.
    - In our example, superkey = {ssn, address}, but one subset {ssn} is a candidate key.
        - Hence the superkey is not a minimal set of fields to uniquely identify a tuple because there exists a minimal subset of the superkey which is a candidate key.

3.5

i) We can deduce with certainty that an attribute is not a candidate key, if two or more records contain the same value of that attribute in a legal instance.

- In Figure 3.1, we see that two records contain the age 18. If age was a candidate key, we could uniquely identify a tuple with age - 18.
- Since we know there is more than one record with the same age, we can't uniquely identify one over the other. Hence age is an attribute that isn't a candidate key.

ii) No, because even though there are fields in which no more than one record contains some value of the field, we cannot state that that field is a candidate key.

- Unless the relation instance was exhaustive as well as legal (meaning all the records we see are all the possible ones), then we cannot tell some field (or set of fields) is a candidate key

because there could always be the addition of another tuple that might duplicate the field which we claim is a candidate key (and it would be legal still since figure 3.1's records aren't exhaustive).

3.6

A foreign key constraint is an integrity check to link two relation instances: the foreign key set of fields in relation 1 should serve as a valid primary key of relation 2.

- This implies that the IC will check to make sure a foreign key is a valid primary key (i.e fk of r1 contains all required fields and values in those fields of the respective field domains from the pk of r2).

A referential integrity constraint is a type of IC that ensures that two linked relation instances are logically connected. Take r1 and r2, where r2 has a foreign key referencing r1.

Case1: Say r1's primary key is ssn, and r2 attempts to point a tuple using a fk_r1 with an ssn that does not exist in r1. Then r1 and r2 are not logically connected.

Case2: Say r1 attempts to delete a tuple with some ssn='777777777' that some other tuple in r2 (record #42) points to. Then a deleted ssn='777777777' in r1, means that record 42 in r2 has an invalid foreign key. This scenario points back to case 1.

A referential integrity constraint ensures that all r2's foreign keys point to tuples which exist in the referenced r1 relation instance and that any deletions or updates to the primary key fields of r1 tuples does not affect some referencing tuple in r2.

3.7

1) List all the foreign key constraints among these relations.
   - FK constraints on Enrolled are:
      - foreign key (sid) references Students
      - foreign key (cid) references Courses

   - FK constraints on Teaches are:
      - foreign key (fid) references Faculty
      - foreign key (cid) references Courses

   - FK constraints on Meets_In are:
      - foreign key (cid) references Courses
      - foreign key (rno) references Rooms

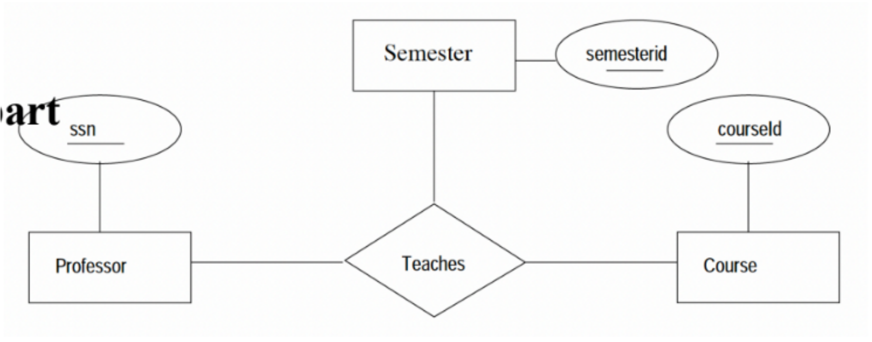
2) Here we give a referential integrity constraint.
   - Suppose a Student leaves the University half-way and his details need to be removed from the University's dbms.
   - Let's say that Student was enrolled in 3 courses as well.
      - Then 3 tuples in Enrolledin reference that Student.

We logically wish to delete those tuples that reference deleted students.
   - Here is the updated EnrolledIn tuple then:

```
- EnrolledIn ( sid:string, cid:string, grade:string,
               PrimaryKey (sid, cid),
               Foreign Key (cid) references Courses,
               Foreign Key (sid) references Students ON DELETE CASCADE )
```

   - The `ON DELETE CASCADE` is a referential integrity constraint that we added.

3.12
I assume for 1-5 that a course can only be taught by one professor in a given semester.
Lines that start with # are comments.

1. 

Professor (ssn, PK(ssn))
Course(courseId, PK(courseID))
Teaches (ssn, courseID, semesterID,
    PK (ssn, courseID, semesterID),
    FK (ssn) ref Professor,
    FK (semesterID) ref Semester
    FK(courseID) ref Course)

2.

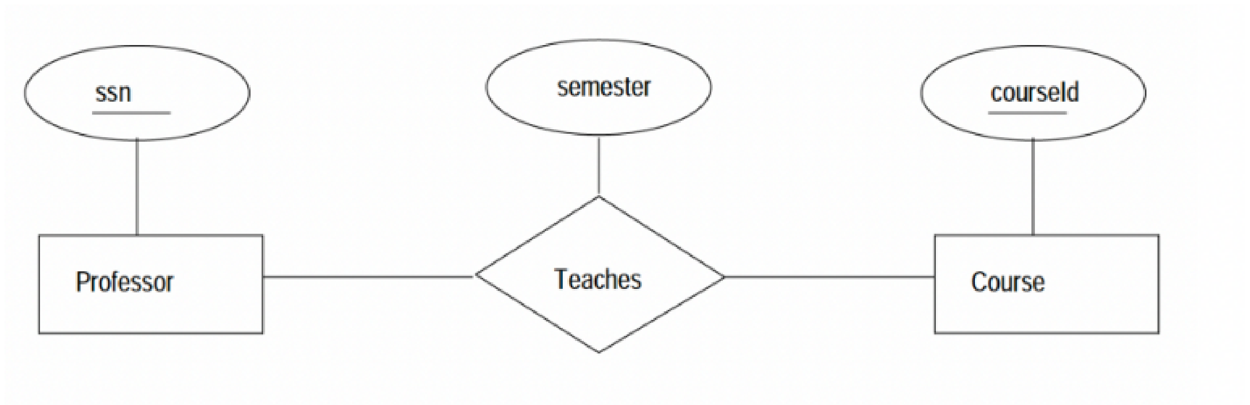


Figure 2.2 ER Diagram for Exercise 2.2, Part 2

Professor (ssn, PK(ssn))
Course(courseId, PK(courseID))
Teaches (ssn, courseID, semester,
    PK (ssn, courseID),
    FK (ssn) ref Professor,
    FK(courseID) ref Course)

3.

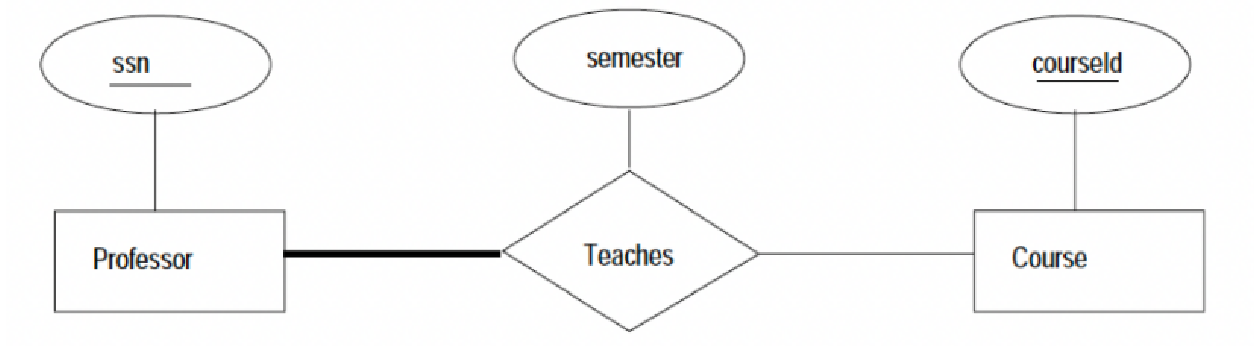


Figure 2.3 ER Diagram for Exercise 2.2, Part 3

Professor (ssn, PK(ssn))
Course(courseId, PK(courseID))
Teaches (ssn, courseID, semester,
    PK (ssn, courseID), #implicit NOT NULL
    FK (ssn) ref Professor,
    FK(courseID) ref Course)

We cannot express the participation constraint of Professor short of using an assertion. Saying ssn in Professor is a FK referencing Teaches is incorrect, since ssn isn't a candidate key for Teaches.

Since some Professors may teach more than one course, we cannot merge Professor and Teaches table either (sine there would be only one field for Course).
Since a course could have many Professors, we cannot apply Merge rule on Course and Teaches.

Our dbms would have to check that every ssn appears in Teaches.

4.

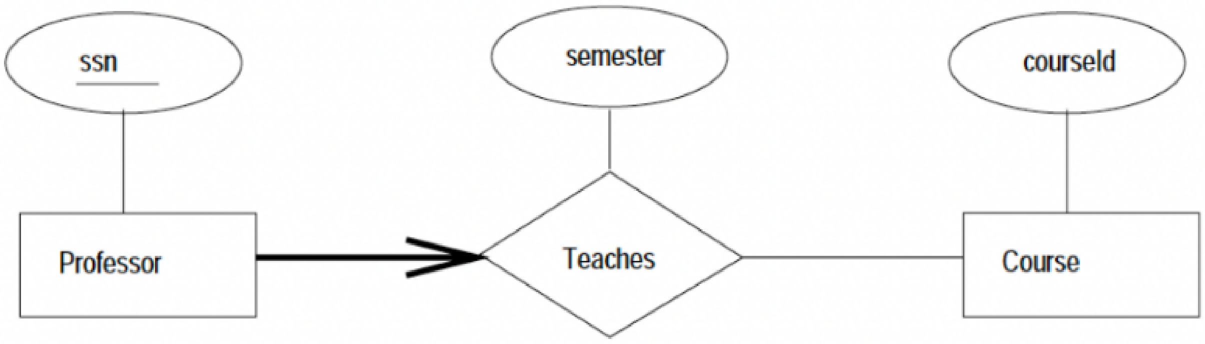


Figure 2.4 ER Diagram for Exercise 2.2, Part 4

Professor (ssn, PK(ssn))
Course(courseId, PK(courseID))
Teaches (ssn, courseID, semester,
    PK (ssn),   # since key/partipication constraint expressed on that side
    FK (ssn) ref Professor,
    FK(courseID) ref Course)

We apply merge rule for Professor and Teaches and get:

Professor (ssn,
    courseId NOT NULL,
    PK(ssn),
    FK (courseId) references Course)

Course(courseId, PK(courseID))

5

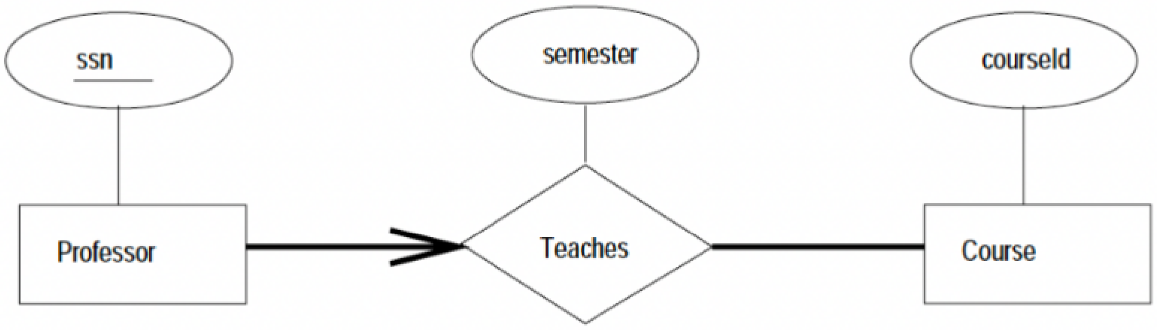
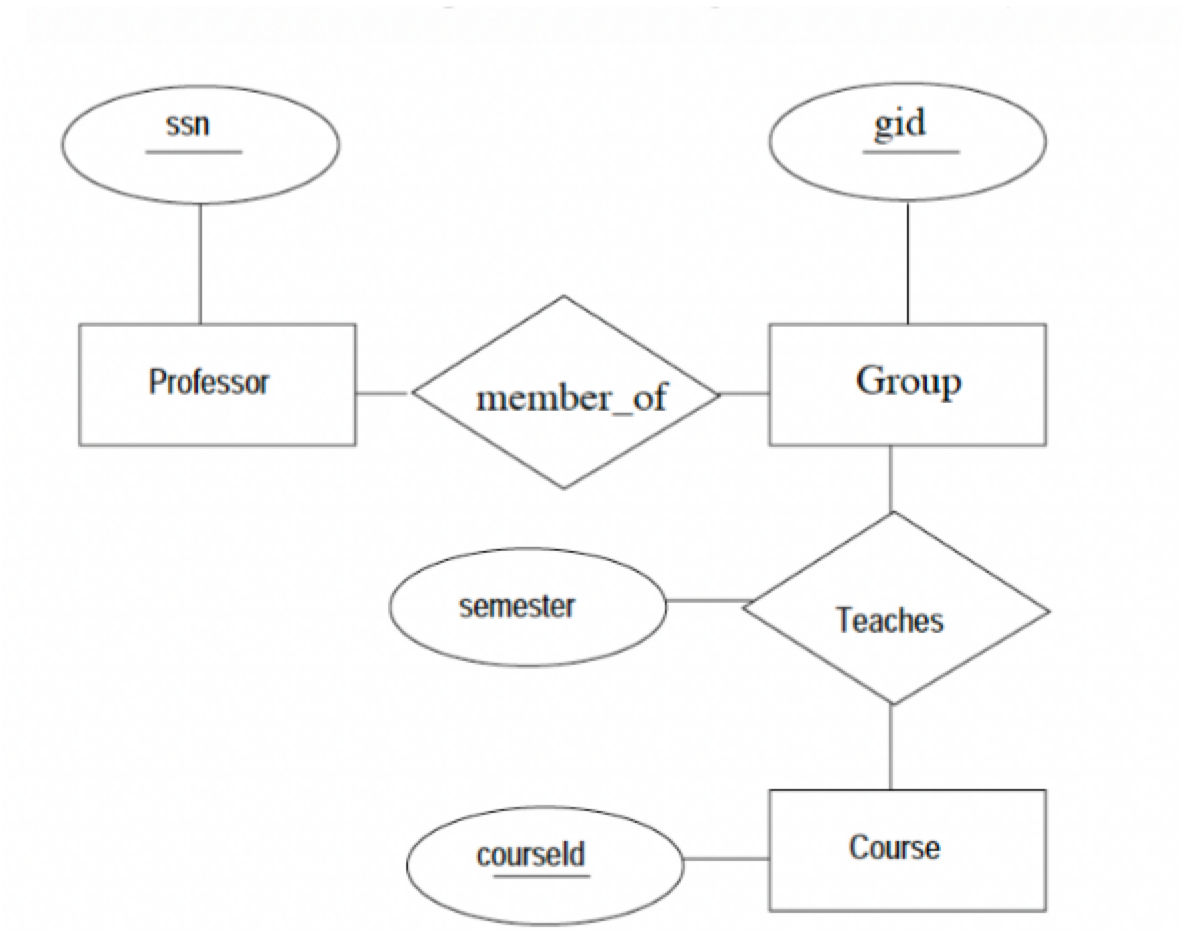


Figure 2.5 ER Diagram for Exercise 2.2, Part 5

Professor (ssn,
    courseId NOT NULL,
    PK(ssn),
    FK (courseId) references Course)

Course(courseId, PK(courseID))

We cannot express course's participation constraint without using an assertion.
Course cannot say courseID is a FK references Professor to guarantee that it appears in the relationship because courseID is not a valid candidate key for Professor.



6.

Professor (ssn,   PK (ssn))

Group (gid, PK (gid))

Member_of (ssn, gid, PK(ssn, gid), FK (ssn) ref Professor, FK(gid) ref Group)

Course(courseId, PK(courseId))

Teaches (gid, courseId, semester,
PK (courseId, gid),
FK (gid) ref Group,
FK (courseId) ref Course)

3.15

We are able to capture all constraints using the 2.5 answer image:

```
Place ( address : varchar(250),
        Primary key (address) )

TelephoneHome( phone_no : int,
        address : varchar(250) NOT NULL,
        Primary key (phone_no),
        Foreign Key (address) references Place )

Lives ( phone_no : int,
        Ssn : int,
        Primary key (phone_no, ssn),
        Foreign key (ssn) references Musician,
        Foreign key (phone_no) references TelephoneHome )

Musician ( ssn: int,
        name : varchar(250),
        Primary key (ssn))

Instrument ( instrumentID : int, dname : varchar(100), key : char(1),
        Primary key (instrumentID) )

Plays ( ssn : int, instrumentID : int,
        Primary key (ssn, instrumentID),
        Foreign key (ssn) references Musician,
        Foreign key (instrumentId) references Instrument )

Album ( albumIdentifier : int,
        copyrightDate : date,
        Title : varchar(250),
        speed : int,
        ssn : int NOT NULL
        Primary key (albumIdentifier)
        Foreign key (ssn) references Musician )

Songs ( songId, int,
        Songauthor : varchar(250),
        Title : varchar(250),
        albumIdentifier : int NOT NULL,
        Primary key (songId),
        Foreign Key (albumIdentifier) references Album )
```

```
Perform ( ssn : int,
          songId : int,
          Primary key (ssn, songId),
          FK (ssn) references Musician,
          FK (songId) references Songs )
```

3.16
I did this based on an answer given by Denson George in recitation.
Here I assume that a technician cannot be a traffic_controller as well (so disjoint, complete).

```
Traffic_Control (
          ssn : int,
          union_mem_no : int,
          exam_date : date,
          Primary key (ssn)
)

Technician (
          ssn : int,
          union_mem_no : int,
          name: varchar(150), salary : float, phone_num : int, address : int,
          Primary key (ssn)
)

Model (
          model_no : int,
          capacity : float,
          weight : int,
          Primary key (model_no)
)
```

```
Expert (
        ssn : int,
        model_no : int,
        Primary key (ssn, model_no),
        Foreign key (ssn) references Technician,
        Foreign key (model_no) references Model
)
```

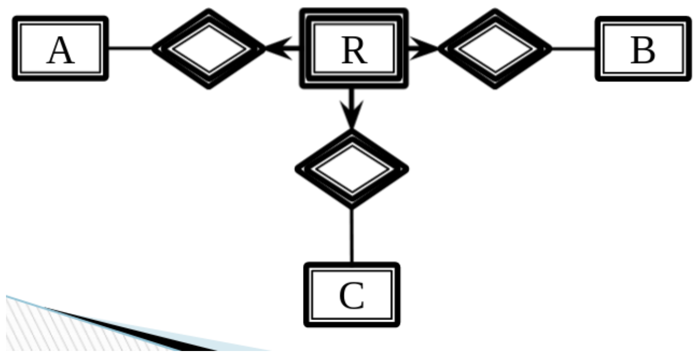
^ For the above we cannot represent the participation constraint of technician without using assertions. We cannot use a null constraint and put Expert with model, since not every model needs a technician, we cannot use a null constraint and put Expert with Technician, since one Technician may be an expert in many models but the tuple would only allow for one model foreign key but here there can be many. (When i say put expert with Technician, I mean merge the schema for the entity and the relationship set).
We cannot work around the assertion requirement by suggesting that ssn for a technician is also a foreign key for expert, since ssn is not a candidate key for expert.

```
Plane (
        reg_no : int,
        model_no : int NOT NULL    # TYPE r.s. - plane needs exactly one model_no
        Primary key (reg_no)
        Foreign key (model_no) references Model
)
Test (
        name : varchar (250),
        FAA_no : int,
        score : float,
        Primary key (FAA_no)
)

Test_info (
        reg_no : int
        FAA_no : int,
        ssn : int,
        score : float,
        hours : float,
        date : date,
        Primary key (FAA_no, reg_no, ssn),
        Foreign key (reg_no) references Plane,
        Foreign key (FAA_no) references Test,
        Foreign key (ssn) references Technician
)
```

2.6 part 2 - tests on a plane must be administered by an expert on that model. Can you modify the SQL statements defining the relations obtained by mapping the ER diagram to check this constraint?

Yes we can.



Note that we apply reification and get an ER diagram where Test_info is a weak entity with identifying parent entities: Expert, plane, and test. Expert can be treated as an entity in the ER diagram (if you apply aggregation). Here we refer to it from its schema.

In the picture above, R would be the Test_Info. A,B,C would be Plane, Test, Expert (in any order you fashion).

The new Test_info relation schema would look like this:

```
Test_info (
        reg_no : int, FAA_no : int, score : float, hours : float,
        date : date,    weak entity key (in conjunction with parent keys)
        ssn : int, model_no: int,
        Primary key (FAA_no, reg_no, ssn, model_no, date),
        Foreign key (reg_no) references Plane,
        Foreign key (FAA_no) references Test,
        Foreign key (ssn, model_no) references Expert
)
```

^ Note that reg_no, FAA_no, model_no, ssn are NOT NULL implicitly because they are in the primary key and primary key attributes cannot be null. So having applied reification, we simply make a schema for Test_info which is a weak entity.

Since the other sides of the identifying relationships have no key or participation constraints, we have nothing more to do (see the picture above).