Practice work with the database design process

Topic 2 Lesson 9 Applying the database design process

Practice problem: university

A university consists of a number of departments. Each department offers several majors. A number of courses make up each major. Students declare a particular major and take courses towards the completion of that major. Each course is taught by a lecturer from the appropriate department, and each lecturer tutors a group of students

Example: entities

- A **university** consists of a number of **departments**. Each department offers several **majors**. A number of **courses** make up each **major. Students** declare a particular major and take courses towards the completion of that major. Each course is taught by a **lecturer** from the appropriate department, and each lecturer tutors a group of students

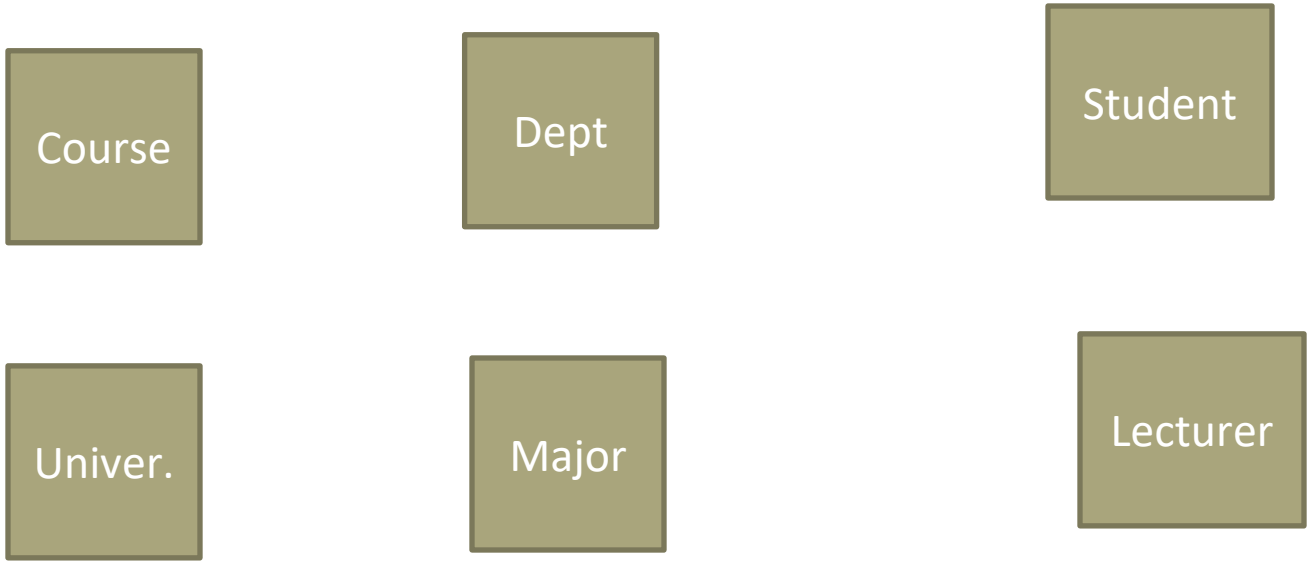
Example: relationships

- A **university** consists of a number of departments. Each department offers several **majors**. A number of **courses** make up each major. **Students** declare a particular major and take courses towards the completion of that major. Each course is taught by a **lecturer** from the appropriate department, and each lecturer tutors a group of students

Entities:

How do we add:

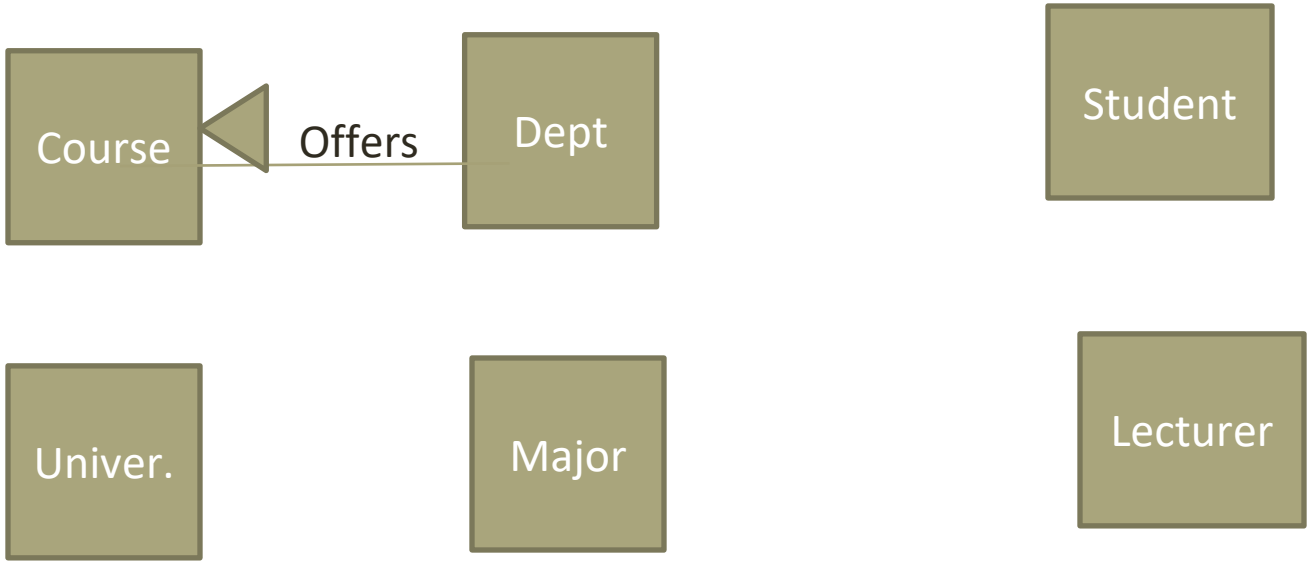
Department offers courses



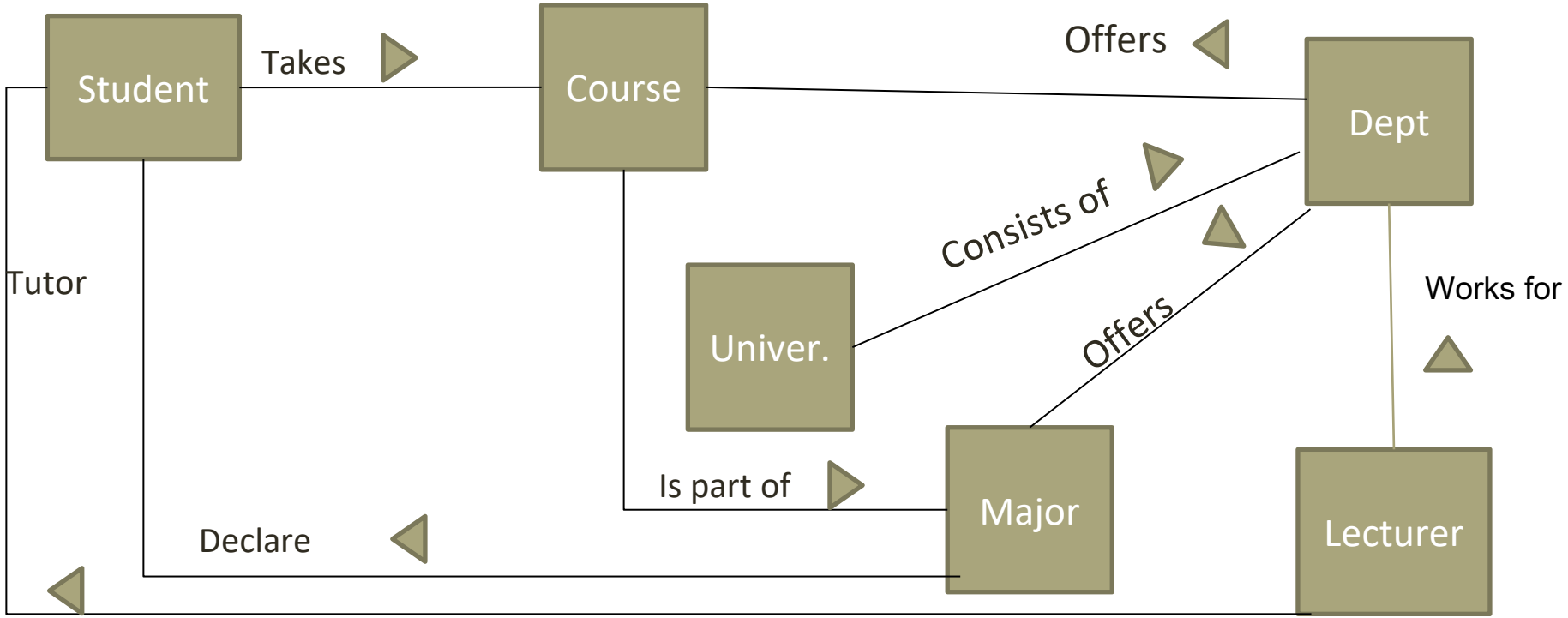
Relationships:

How do we add:

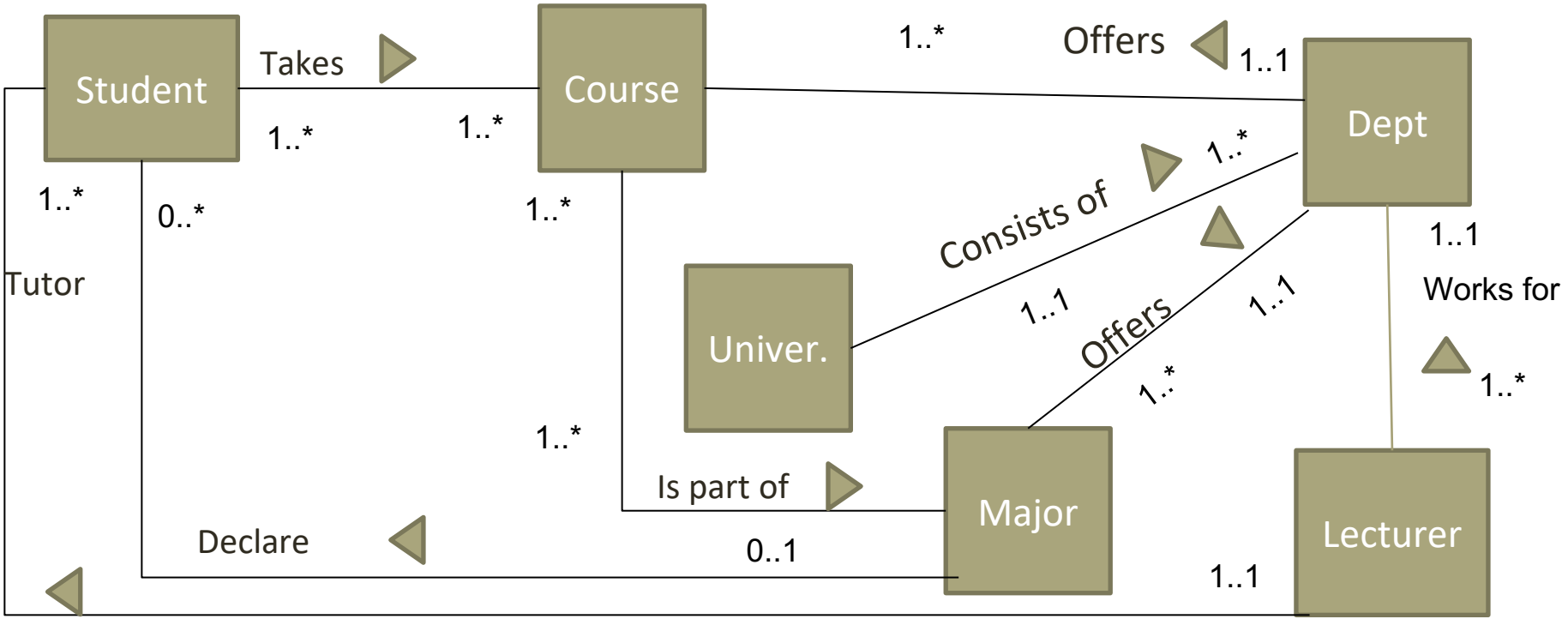
Department offers courses



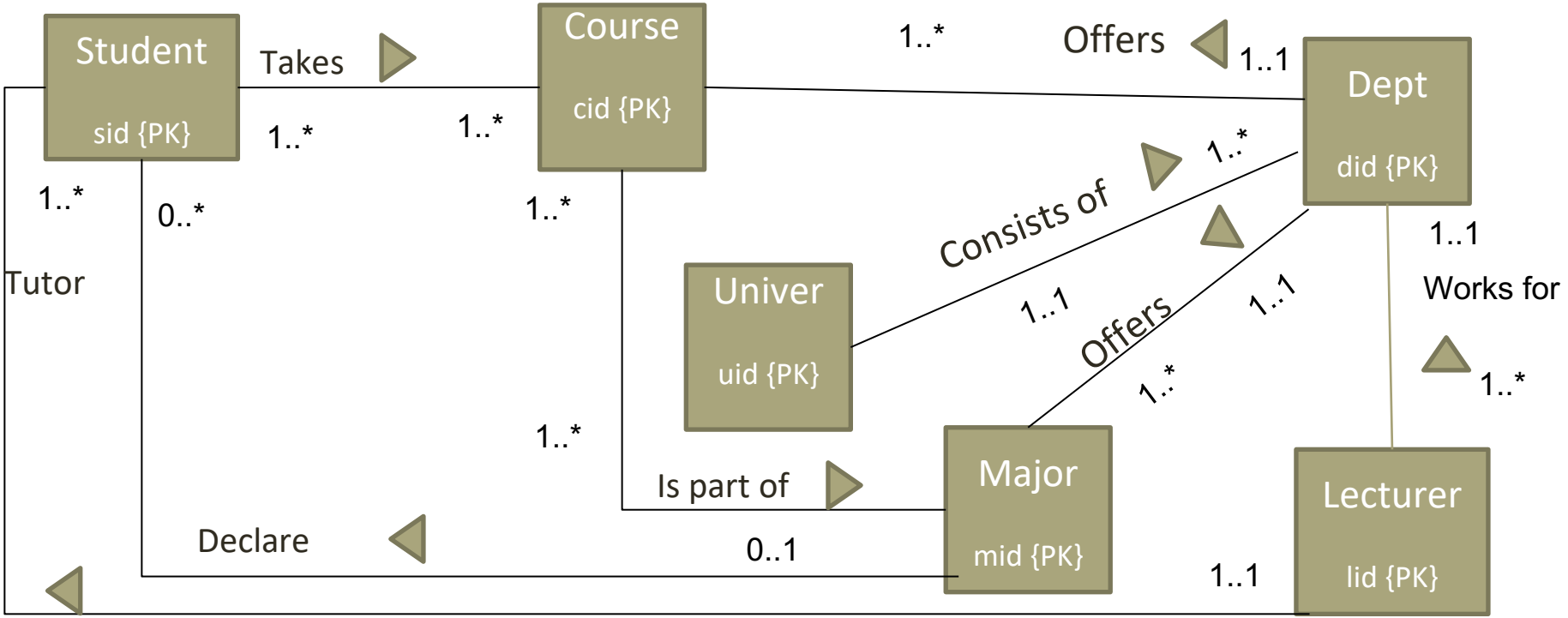
All relationships



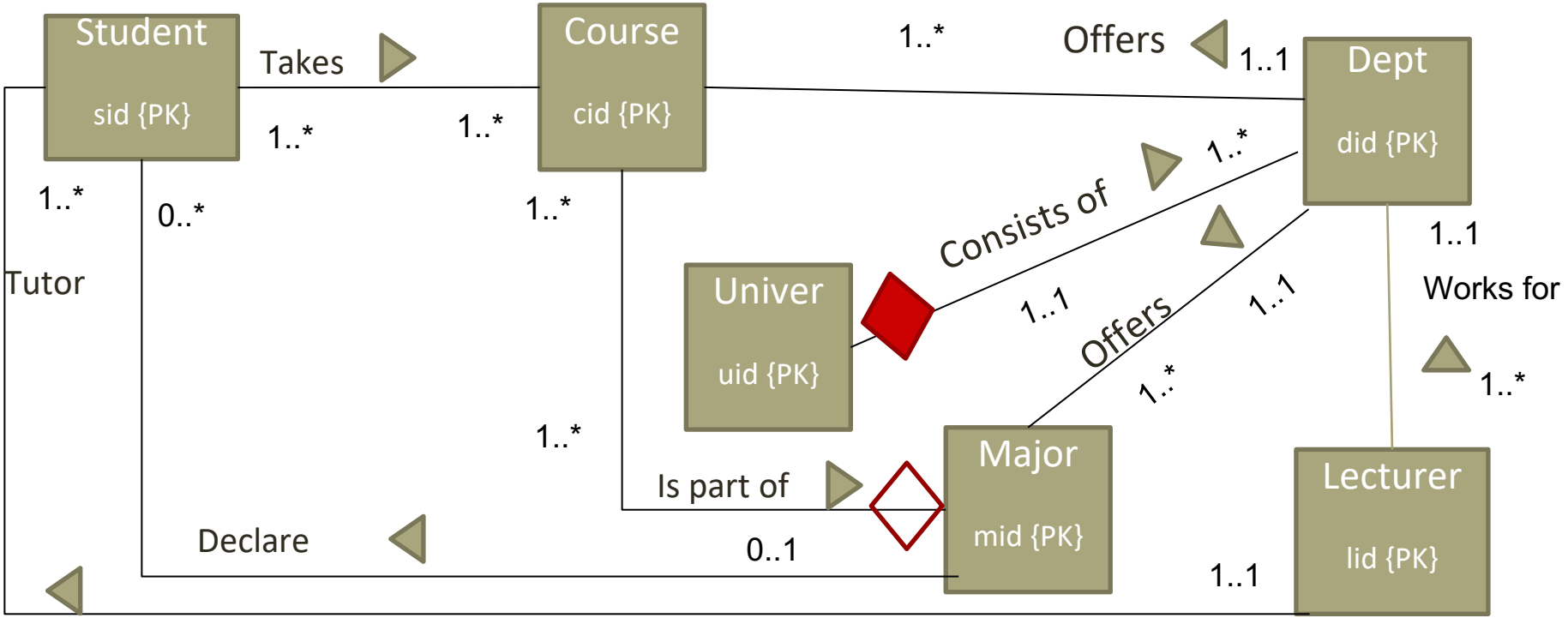
Multiplicities



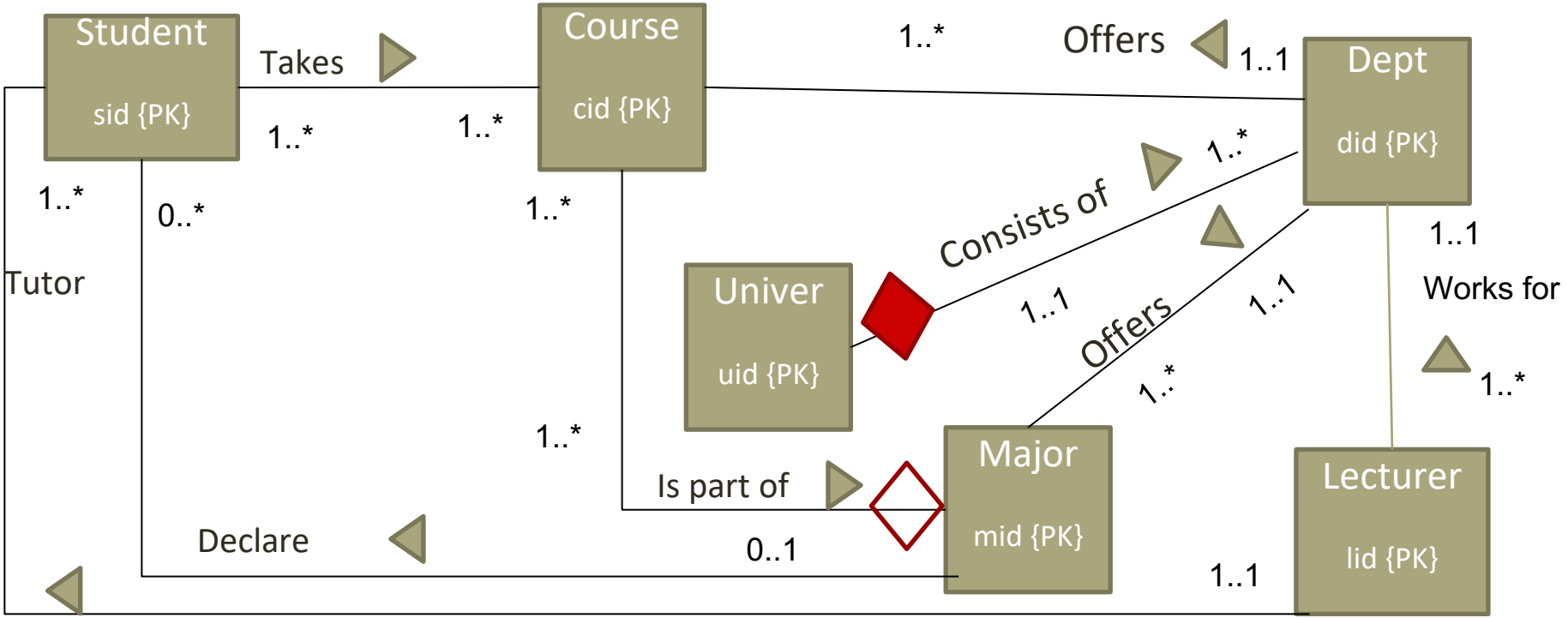
Primary keys are needed



Enhanced relationships



One solution



Practice problem: musicians

Notown Records has decided to store information about musicians who perform on its albums (as well as other company data) in a database.

Each musician that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians do not have cell phones, often share the same address, and no address has more than one landline phone. Given their limited use, cell phones are not tracked.

Each instrument used in songs recorded at Notown has a unique identification number, a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).

Each album recorded on the Notown label has a unique identification number, a title, a copyright date, a format (e.g., CD or MC), and an album identifier.

Each song recorded at Notown has a title and an author. The author of a song is a musician. There is 1 and only 1 author per song.

Each musician may play several instruments, and a given instrument may be played by several musicians.

Each album has a number of songs on it, but no song may appear on more than one album.

Each song is performed by one or more musicians, and a musician may perform a number of songs.

Each album has exactly one musician who acts as its producer. A musician may produce several albums, of course.

Design a conceptual schema for Notown and draw an UML diagram for your schema. Be sure to indicate all key and multiplicity constraints and any assumptions you make. Once you have created the diagram, create the necessary SQL CREATE TABLE commands necessary to support it.

Identify the entities

Each **musician** that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians do not have cell phones, often share the same **address**, and no address has more than one landline phone. Given their limited use, cell phones are not tracked.

Each i**nstrument** used in songs recorded at Notown has a unique identification number, a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).

Each **album** recorded on the Notown label has a unique identification number, a title, a copyright date, a format (e.g., CD or MC), and an album identifier.

Each **song** recorded at Notown has a title and an author. The author of a song is a musician. There is 1 and only 1 author per song.

Each musician may play several instruments, and a given instrument may be played by several musicians.

Each album has a number of songs on it, but no song may appear on more than one album.

Each song is performed by one or more musicians, and a musician may perform a number of songs.

Each album has exactly one musician who acts as its producer. A musician may produce several albums, of course.

Identify the relationships

Each musician that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians do not have cell phones, often **share** the same address, and no address has more than one landline phone. Given their limited use, cell phone numbers are not tracked.

Each instrument used in songs recorded at Notown has a unique identification number, a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).

Each album **recorded** on the Notown label by a musician has a unique identification number, a title, a copyright date, a format (e.g., CD or MC), and an album identifier.

Each song recorded at Notown has a title and an author. The author of a song is a musician. There is 1 and only 1 author per song.

Each musician may **play** several instruments, and a given instrument may be played by several musicians.

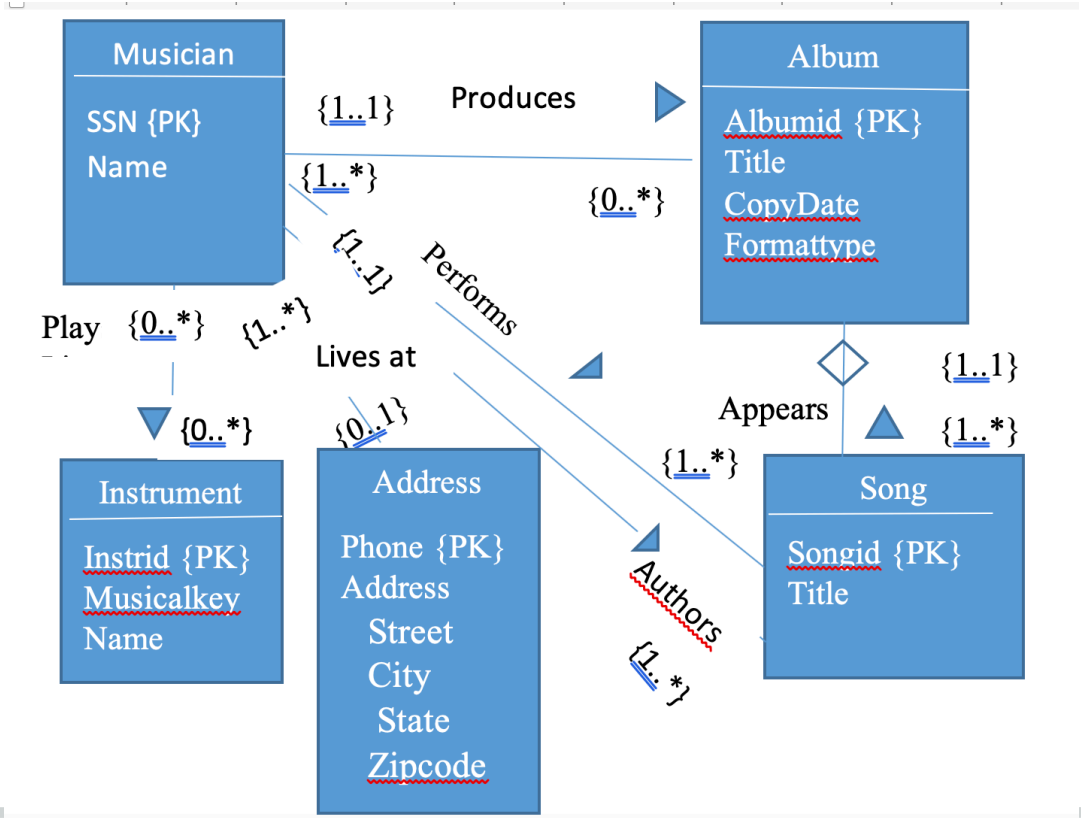
Each album **contains** a number of songs, but no song may appear on more than one album.

Each song is **performed** by one or more musicians, and a musician may perform a number of songs.

Each album has exactly one musician who acts as its **producer**. A musician may produce several albums, of course.

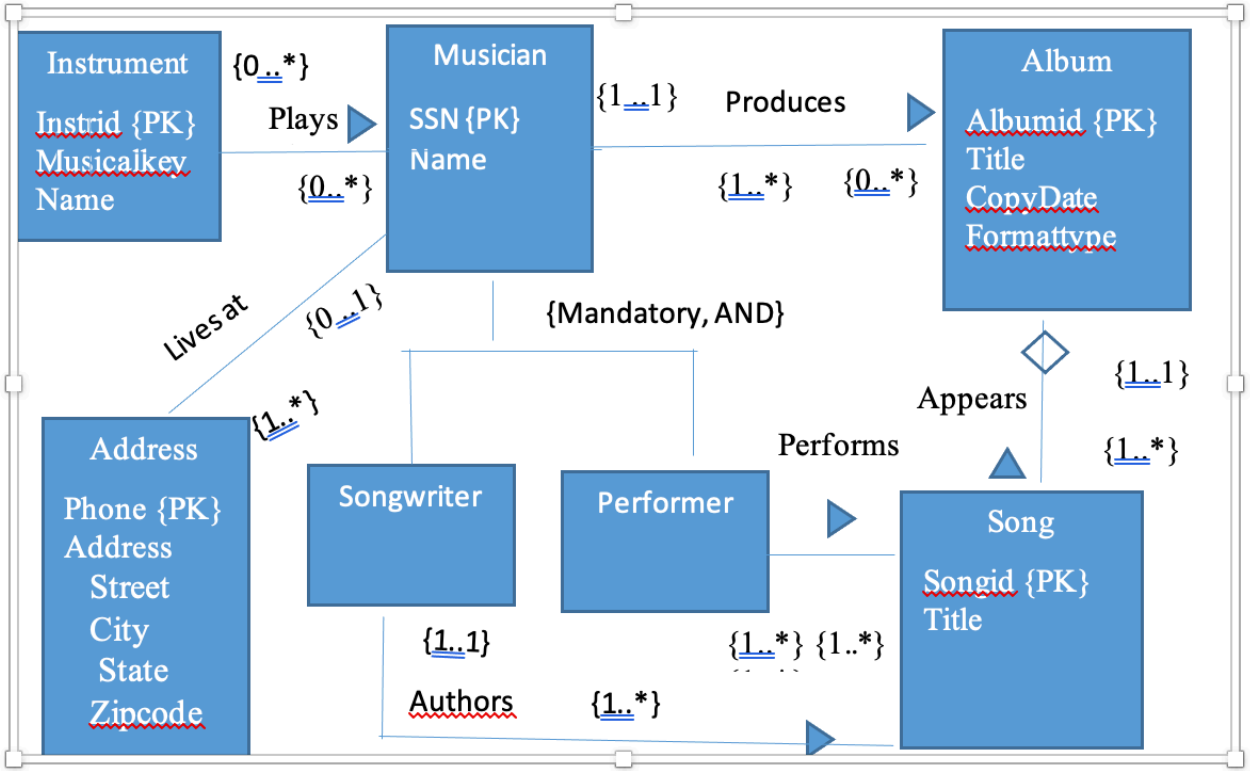
Conceptual design

- We need both authors and performs between musician and song to capture both relationships
- Alternatively authors and producers could have been a subclass of musician
- We chose to make address a separate entity since many musicians may live at the same address



Conceptual design solution 2

Using a hierarchy for musicians



Conceptual Model to SQL tables?

- Identify the tables for the entities
- Identify the tables for the relationships
- Identify the table name, field names and data types
- Identify the primary keys
- Identify the foreign keys
  - Determine behavior for DELETE/UPDATE operations
- Represent other column and table constraints
  - NULL allowed for field?
  - Default value for a field?

Tables for entities with foreign keys

```
CREATE TABLE address
      ( phone CHAR(11) PRIMARY KEY,
       Street VARCHAR(64) NOT NULL ,
        City VARCHAR(64) NOT NULL,
        State CHAR(2) NOT NULL,
      );

CREATE TABLE musician
( ssn INT PRIMARY KEY,
  name VARCHAR(64) NOT NULL,
   phone CHAR(11) DEFAULT “NOT KNOWN”,
   CONSTRAINT musician_address_fk FOREIGN KEY  (phone)
      REFERENCES address(phone)
      ON DELETE SET DEFAULT
      ON UPDATE SET DEFAULT
  );

  CREATE TABLE instrument
( instrumentid INT PRIMARY KEY,
  name VARCHAR(64) NOT NULL,
  musicalkey VARCHAR(64)
  );
```

```
CREATE TABLE album
( albumid INT AUTO_INCREMENT PRIMARY KEY,
  releasedate DATE NOT NULL,
  formattype char(8) NOT NULL,
 title VARCHAR(128) NOT NULL
  producer INT NOT NULL,
  FOREIGN KEY (producer) REFERENCES musician(ssn)
    ON UPDATE CASCADE ON DELETE CASCADE
  );

  CREATE TABLE song
( songid INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(128) NOT NULL,
  author INT NOT NULL,
  albumid INT NOT NULL,
  FOREIGN KEY (albumid) REFERENCES album(albumid)
    ON UPDATE RESTRICT ON DELETE RESTRICT,
  FOREIGN KEY (author) REFERENCES musician(ssn)
    ON UPDATE RESTRICT ON DELETE RESTRICT
  );
```

Tables for the many to many relationships

```
-- mapping tables to support the multiple artists
    on a song
 CREATE TABLE performs
 (artist INT,
  song INT,
  PRIMARY KEY (artist,song),
  FOREIGN KEY (artist) REFERENCES
    musician(ssn)
     ON UPDATE RESTRICT ON DELETE
    RESTRICT,
  FOREIGN KEY (song) REFERENCES
    song(songid)
     ON UPDATE RESTRICT ON DELETE
    RESTRICT
    );
```

```
 -- mapping table to support the multiple
    instruments a musician can play
CREATE TABLE musiciantoinstrument
( instrumentid INT,
     artist INT,
     PRIMARY KEY (instrumentid,artist),
     FOREIGN KEY (instrumentid)
    REFERENCES instrument(instrumentid)
         ON UPDATE RESTRICT ON DELETE
    RESTRICT,
    FOREIGN KEY (artist) REFERENCES
    musician(ssn)
         ON UPDATE RESTRICT ON DELETE
    RESTRICT
     );
```

Practice with normal form

Practice problem: normalization (1)

Determine if table is unnormalized, 1st, 2nd, 3rd normal form

BuildingNo	RoomNo	RoomCapacity	BuildingName
1	100	20	Behrakis
1	200	40	Behrakis
1	300	30	Behrakis
2	100	30	International Village

1st Normal Form

Practice problem: normalization (2)

Determine if table is unnormalized, 1st, 2nd, 3rd normal form

StudentNo	StudentName	Courses
1	Jane Smith	CS2500 CS3200 CS4100
2	Henry Wu	
3	Miles Standish	CS2500 CS2510 CS3000
4	Elizabeth Khan	CS2500 CS2510

Unnormalized

Practice problem: normalization (3)

Determine if table is unnormalized, 1st, 2nd, 3rd normal form

BuildingNo	Abbreviation	BuildingName
1	BRK	Behrakis
2	WVH	West Village H
3	NI	Nightingale Hall
4	RY	Ryder Hall

3rd Normal Form

Practice problem: normalization (4)

Determine if table is unnormalized, 1st, 2nd, 3rd normal form

StudentNo	FirstName	LastName
1	Jane	Smith
2	Henry	Wu
3	Miles	Standish
4	Elizabeth	Khan

3rd Normal Form

Practice problem: functional dependency (1)

Do you suspect a functional dependency that should not be in this table? If so which fields?

Item_id	Item_type	Color	Item_description
1	hoodie	white	Comfortable 100% cotton sweatshirt
2	hoodie	black	Comfortable 100% cotton sweatshirt
3	jeans	blue	Acid-washed, slim cut
4	jeans	black	Acid-washed, slim cut
5	hoodie	blue	Comfortable 100% cotton sweatshirt
6	jeans	blue	Acid-washed, slim cut

Item type → item_description

Practice problem: functional dependency (2)

Do you suspect a functional dependency that should not be in this table? If so which fields?

Item_id	Item_type	Color	Item_description
1	hoodie	white	Comfortable 100% cotton sweatshirt
2	hoodie	white	Comfortable 100% cotton sweatshirt
3	jeans	blue	Acid-washed, boot cut
4	jeans	blue	Acid-washed, slim cut
5	hoodie	white	Comfortable 100% cotton sweatshirt
6	jeans	blue	Acid-washed, slim cut

Item_type → color

Practice problem: functional dependency (3)

Do you suspect a functional dependency that should not be in this table? If so which fields?

Item_id	Item_type	Color	Item_description
1	hoodie	white	Comfortable 100% cotton sweatshirt, medium weight
2	hoodie	red	Comfortable 100% cotton sweatshirt, heavy weight
3	jeans	black	Acid-washed, boot cut
4	jeans	blue	Acid-washed, slim cut
5	hoodie	red	Comfortable 100% cotton sweatshirt, heavy weight
6	jeans	blue	Acid-washed, slim cut

Item_type, color → item_description