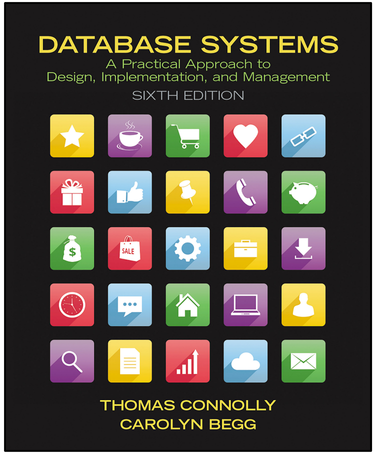
Data Redundancy & Normal Form

Topic 3, Lesson 8

Schema Refinement

Chapter 14 14.1-14.4 Connolly and Begg



Functional Dependencies

Reducing Data Redundancy

Major aim of relational database design is to group attributes into relations to minimize data redundancy.

Benefits:

- Updates to the data stored in the database are achieved with a minimal number of operations thus reducing the opportunities for **data inconsistencies**.
- Reduction in the file storage space required by the base relations thus **minimizing costs**.

What is wrong with this table?

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Data redundancy of the branch address

Data redundancy leads to update anomalies

Relations that contain redundant information may potentially suffer from **update anomalies.**

What are update anomalies?

- Insertion anomaly
  - Tuple being inserted may contain data fields that are inconsistent with data in other tuples in the table
- Deletion anomaly
  - Deleting a tuple leads to loss of information other than the tuple
- Modification anomaly
  - Modification of one tuple is dependent on the modifications of other tuples

Functional dependency

Important concept associated with normalization.

Functional dependency describes a relationship between attributes.

For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted $A \rightarrow B$), if each value of A in R is associated with **exactly one value** of B in R.

Types of Functional Dependencies

Full functional dependency indicates that if A and B are attributes of a relation, B is fully functionally dependent on A, if B is functionally dependent on A, but not on any proper subset of A.

Partial dependency if B is also functionally dependent on a subset of A.

Transitive dependency

Important to recognize a transitive dependency because its existence in a relation can potentially cause update anomalies.

Transitive dependency describes a condition where A, B, and C are attributes of a relation such that if $A \rightarrow B$ and $B \rightarrow C$, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).

Let’s remove the redundancy

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

A staff table and a branch table

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Reconfigure tables

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

A staff table and a branch table

Redundancy only for the foreign key

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Example of transitive dependency

Consider functional dependencies in the StaffBranch relation

StaffNo → sName, position, salary, **branchNo**, bAddress

branchNo → bAddress

Transitive dependency, StaffNo → branchNo

branchNo → bAddress

Transitive dependency staffNo to bAddress via branchNo.

Redundancy leads to anomalies

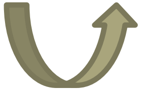
UPDATE ANOMALIES

Modification anomaly: Can we change W in just the first tuple?

Deletion anomaly: Can we delete tuple 3 and 4?

Insertion anomaly: What if we insert another tuple where the rating equals 8 but the wage is not equal to 10?
How do we track the wage associated with ratings not stored in the employee table?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40



There is a functional dependency between Rate and Wage. This functional dependency limits the operations I can do on my data if I want to keep my data consistent.

Identifying functional dependencies

Functional dependencies, can be used to identify schemas with such problems and to suggest schema refinements.

Each relation is dependent on the primary key since the primary key identifies the values for the other attributes

In our prior example, we had 2 functional dependencies (FDS)

S → {S,N,L,R,W,H}

R → {W} - each value of R is associated with exactly 1 value of W

Determinant on left hand side of →

If no attributes are dependent on another (not including the primary key) then there is no redundancy

How to remove a functional dependency?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Decompose the original relation into 2 relations.

R	W
8	10
5	7

Wage

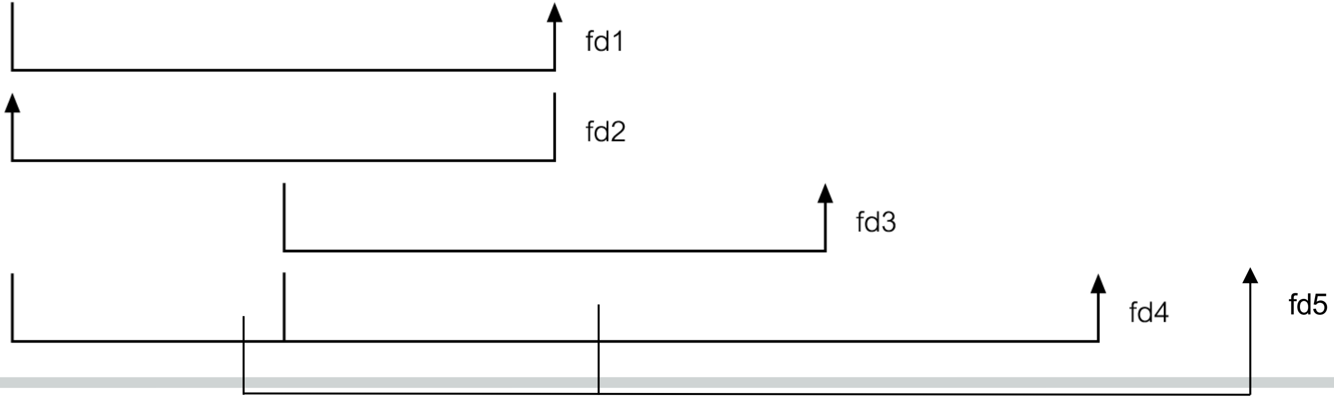
Example: Find the functional dependencies

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t

Given these values:

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t



Identified functional dependencies

Function dependencies between attributes A to E in the Sample relation.

$A \rightarrow C$	(fd1)
$C \rightarrow A$	(fd2)
$B \rightarrow D$	(fd3)
$A, B \rightarrow E$	(fd4)
$B, C \rightarrow E$	(fd5)

Candidate keys for sample relation

A candidate key must provide the irreducibility and uniqueness property. We do not have 1 attribute that can determine all attributes.

A,B determines C, D, and E

B,C determines A, D, and E

So we have 2 candidate keys.

Identifying functional dependencies

Main characteristics of functional dependencies used in normalization:

- There is a **one-to-one** relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency.
- Holds for **all** time.
- The determinant has the **minimal** number of attributes necessary to maintain the dependency with the attribute(s) on the right hand-side.

Decomposition properties

Decomposition process must ensure:

- **Lossless-join property** enables us to find any instance of the original relation from corresponding instances in the smaller relations.
  - We can still generate the original table from the decomposed tables (no loss of information)
- **Dependency preservation property** enables us to enforce a constraint on the original relation by enforcing some constraint on each of the smaller relations
  - No functional dependency is lost in the process

Lossless Join Property guarantees that

- The union of the attributes in the 2 smaller tables must be equal to the attributes in the larger table
- The intersection of the attributes in the 2 smaller tables must not be empty
- A common attribute in one of the relations must be a key in the other table

Summary

Functional dependency not involving a candidate key can lead to update anomalies

Update anomalies can lead to data inconsistencies

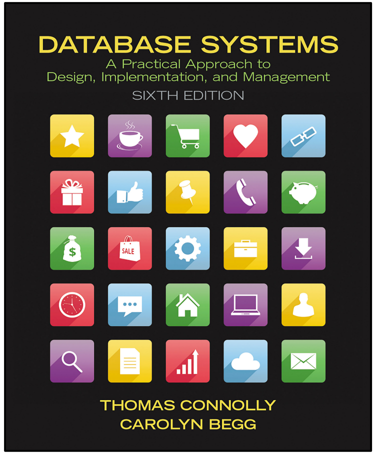
Schema refinement using the normalization process can resolve update anomalies

Normal Form

Topic 3 Lesson 9

Removing redundancy from a data schema

Chapter 14 14.5-14.9 Connolly and Begg



Normalization process

Formal technique for analysing a relation based on its primary key and the functional dependencies between the attributes of that relation.

We use the normalization process as a validation technique for the defined relations.

It is a crucial step in the **logical database design process.**

Normalization

Normalization identifies redundancy that leads to functional dependency.

The normalization process is a series of tests that help identify the optimal grouping of attributes to relations.

GOAL: reduce data redundancy

Normal form addresses dependencies

- GOAL: Free the collection of relations from undesirable insertion, modification and deletion **dependencies**
  - If schema has duplicated data in multiple rows
    - Forced to update/delete all copies of a piece of data
    - How do you know you got all copies of it?
- Address the flaws in the current design

Normal form leads to a cleaner schema

- Reduce the need for restructuring the collection of relations
  - Build an extensible design now as opposed to later
- Make the relational model more informative to users
  - Cleaner model should be easier to understand
- Make the collection of relations neutral to the query statistics
  - Designed for general purpose querying

Unnormalized form

- No primary key or NULL values in the primary key fields
- A table that contains an attribute with one or more repeating groups or contains a set
  - **A repeating group** is a set of logically related fields or values that occur multiple times in one record.
- Attributes need not be atomic

Unnormalized form

Table to track birth mother to child

Set of children to a mother

One field to represent all children

UNNORMALIZED FORM (UNF) – DUPLICATES ENTITIES		
Mother Id	Mother Name	Children
1	Elsa	Alex
1	Elsa	Mary Alice Tom Lou
2	Golda	George Fred
3	Viola	Ava
4	Iris	Kayla
5	Daisy	Harry

Does this solve the problem?

No, but it does not stop people from designing databases this way

Still unnormalized

Mother Id	Mother Name	Child1	Child2	Child3	Child4
1	Elsa	Alex	NULL	NULL	NULL
1	Elsa	Mary	Alice	Tom	Lou
2	Golda	George	Fred	NULL	NULL
3	Viola	Ava	NULL	NULL	NULL
4	Iris	Kayla	NULL	NULL	NULL
5	Daisy	Harry	NULL	NULL	NULL

First normal form

- Tuples in a relation must contain the same number of fields
- The domain of each attribute contains atomic values
- The value of each attribute contains only a single value
    - No attributes are sets or a repeating group.

Relational Model
1st normal form

UNF to 1NF

- Nominate an attribute or group of attributes to act as the key for the unnormalized table.
- Identify the repeating group(s) in the unnormalized table which repeats for the key attribute(s).
  - Remove the set by creating a separate table for the set or if there is an upper limit to the set you can flatten it into fields

1st Normal Form

Mother Id	Mother Name
1	Elsa
2	Golda
3	Viola
4	Iris
5	Daisy

Child Id	Name	Mother
11	Mary	1
12	Alice	1
13	George	2
14	Fred	2
15	Ava	3
16	Kayla	4
17	Harry	5
18	Alex	1
19	Tom	1
20	Lou	1

Decompose table, remove repeating attributes

Second normal form

- Requirement for tables that have a composite key
- Table must already be in first normal form
- Every non-primary key attribute is fully functionally dependent on the (entire) primary key
- A table in first normal form and having a primary key with only one field is also in 2nd normal form

Second normal form

Based on the concept of full functional dependency.

Full functional dependency indicates that if

- A and B are attributes of a relation,
- B is fully dependent on A if B is functionally dependent on A but not on any proper subset of A.

1NF to 2NF

Identify the primary key for the 1NF relation.
Identify the functional dependencies in the relation.
If partial dependencies exist on the primary key remove them by placing then in a new relation along with a copy of their determinant.

Example 2NF with a composite key

1st Normal Form but NOT 2nd NORMAL FORM

Mother Id	First Name	Last Name	Hospital	Hospital Address
1	Elsa	General	BIDMC	Boston
2	Golda	Major	MGH	Boston
3	Viola			
4	Iris			
5	Daisy			

2nd NORMAL FORM

Hospital_ID	Hospital	Hospital Address
1	BIDMC	Boston
2	MGH	Boston
3	TMC	Cambridge
4	Mayo	Allston

2nd NORMAL FORM

Mother Id	First Name	Last Name	Hospital Id
1	Elsa	General	1
2	Golda	Major	2
3	Viola	Funt	3
4	Iris	Batter	1
5	Daisy	Mae	4

Third normal form

- Table is in first and second normal form
- No dependencies between 2 non-key attributes
- No non-primary-key attribute is transitively dependent on the primary key
- Solution: decompose the table so that the offending attribute is in a separate table
- Attribute is fully functionally dependent on the primary key

Third normal form

Based on the concept of transitive dependency.

Transitive Dependency is a condition where

- A, B and C are attributes of a relation such that if $A \rightarrow B$ and $B \rightarrow C$,
- then C is transitively dependent on A through B. (Provided that A is not functionally dependent on B or C).

2NF to 3NF

- Identify the primary key in the 2NF relation.
- Identify functional dependencies in the relation.
- If transitive dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their dominant.

Example: to 3rd normal form

2nd NORMAL FORM				
Mother Id	First Name	Last Name	Hospital Id	Room Number
1	Elsa	General	1	36
2	Golda	Major	[illegible]	[illegible]
3	Viola	Funt	[illegible]	[illegible]
4	Iris	Batter	[illegible]	[illegible]
5	Daisy	Mae	[illegible]	[illegible]

3rd Normal Form		
Register Id	Hospital Id	Room Id
1	1	36
2	2	48
3	3	36
4	1	41
5	4	32

3rd NORMAL FORM			
Mother Id	First Name	Last Name	Register Id
1	Elsa	General	1
[illegible]	Golda	Major	2
[illegible]	Viola	Funt	3
[illegible]	Iris	Batter	4
[illegible]	Daisy	Mae	5

Bill Kent’s quote:

Every non-key attribute must provide a fact :

about the key,

the whole key

and nothing but the key

Summary

To remove unnecessary redundancy, a table needs to be decomposed and the redundancy should be broken out into a separate table.

Practice work with the database design process

Topic 3 Lesson 10 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

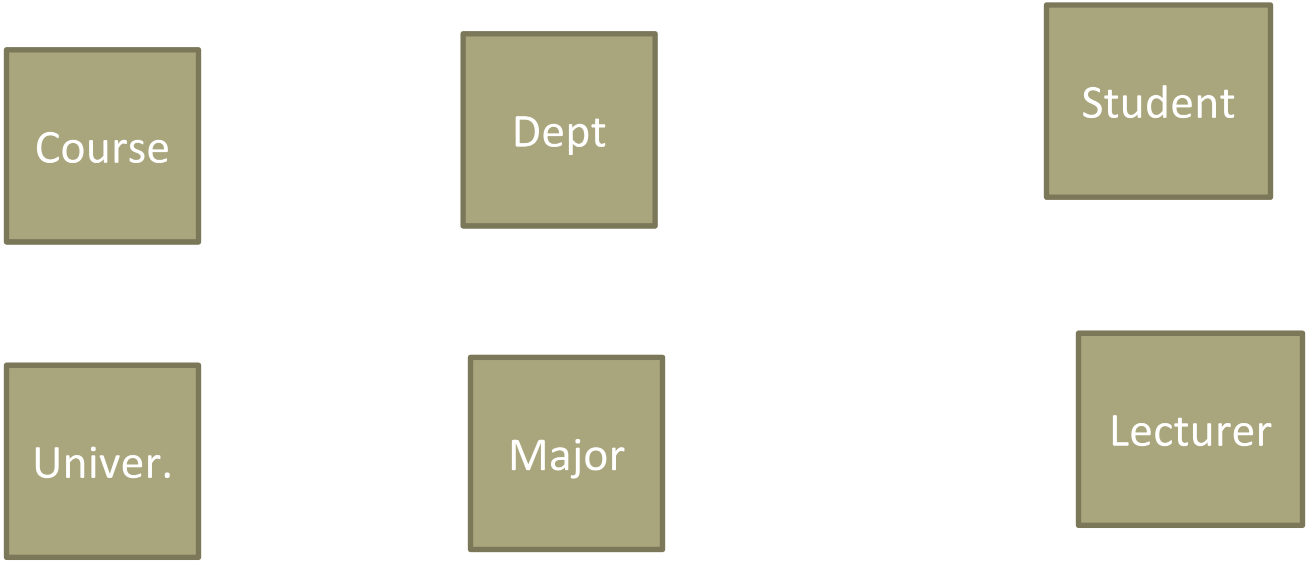
Example: relationships

- 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:

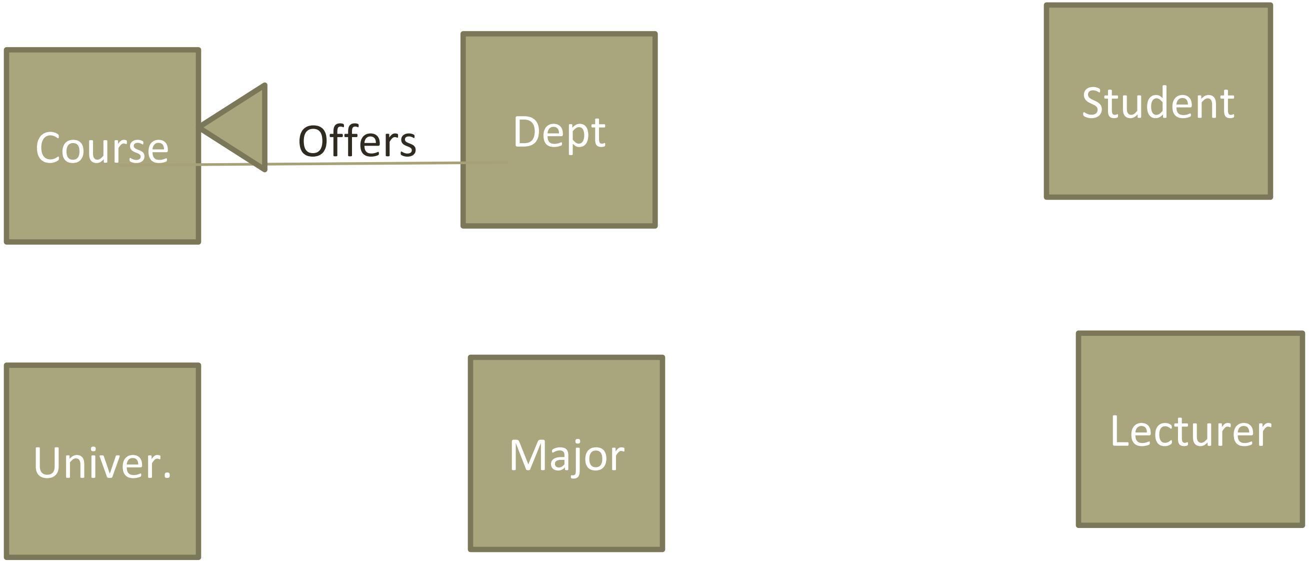
Department offers courses



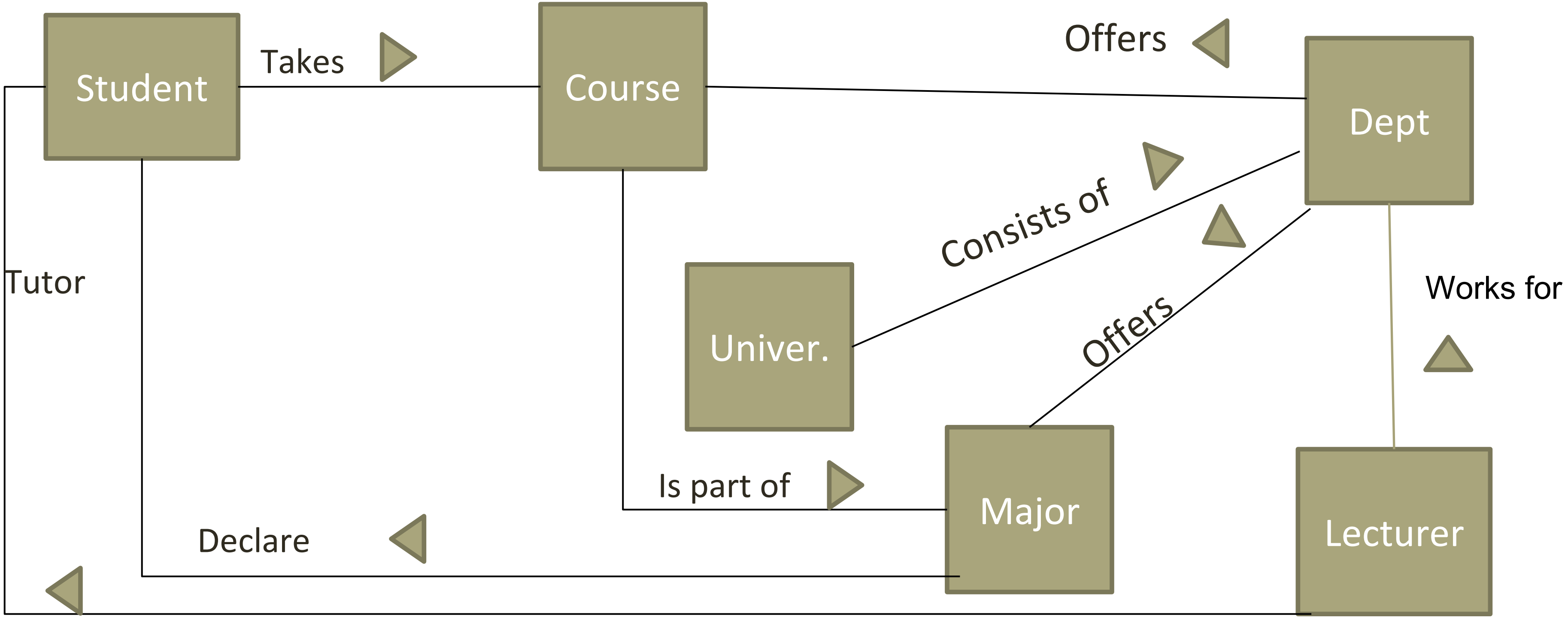
Relationships:

How do we add:

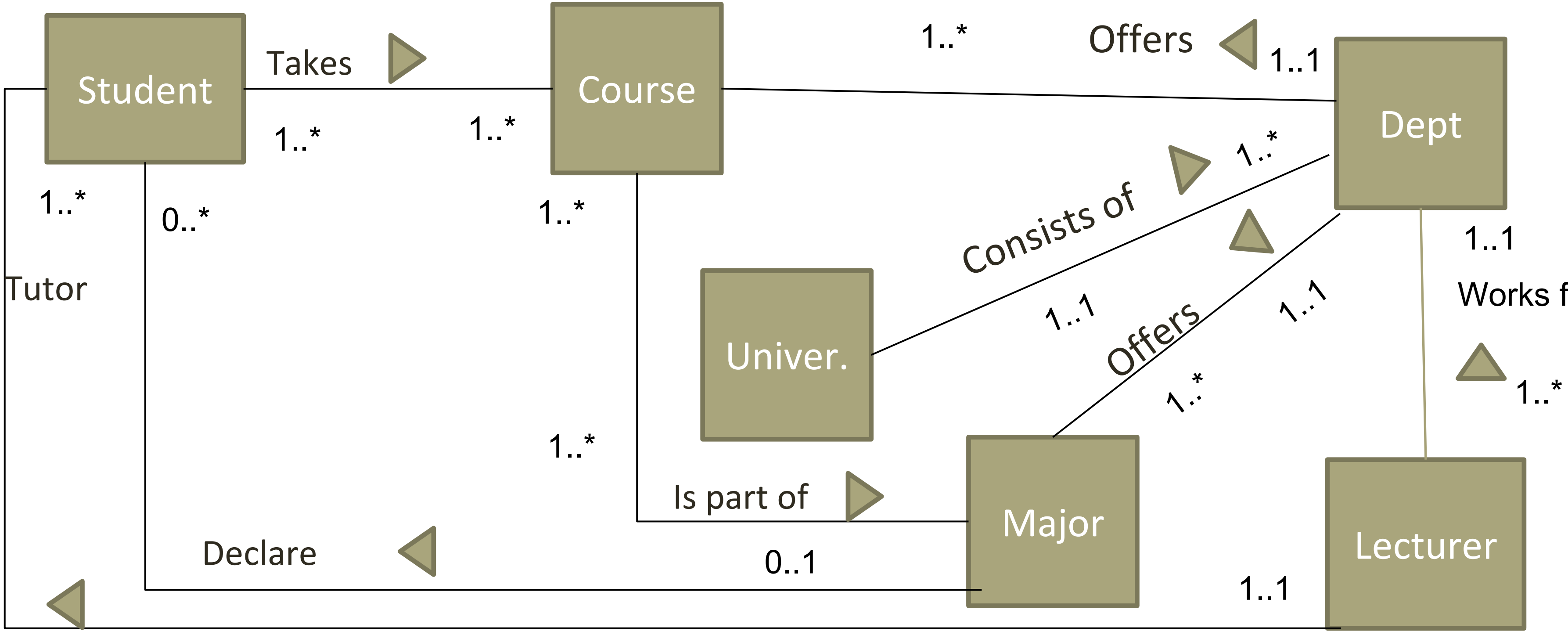
Department offers courses



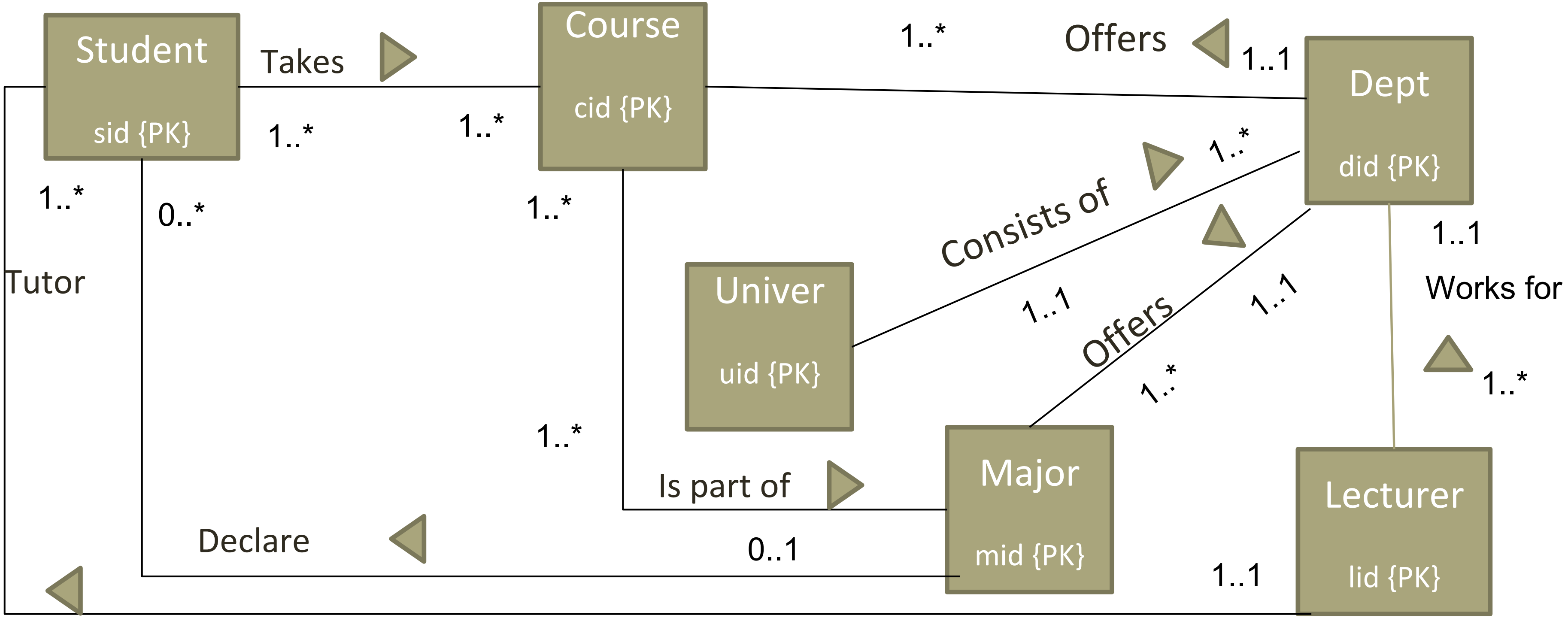
All relationships



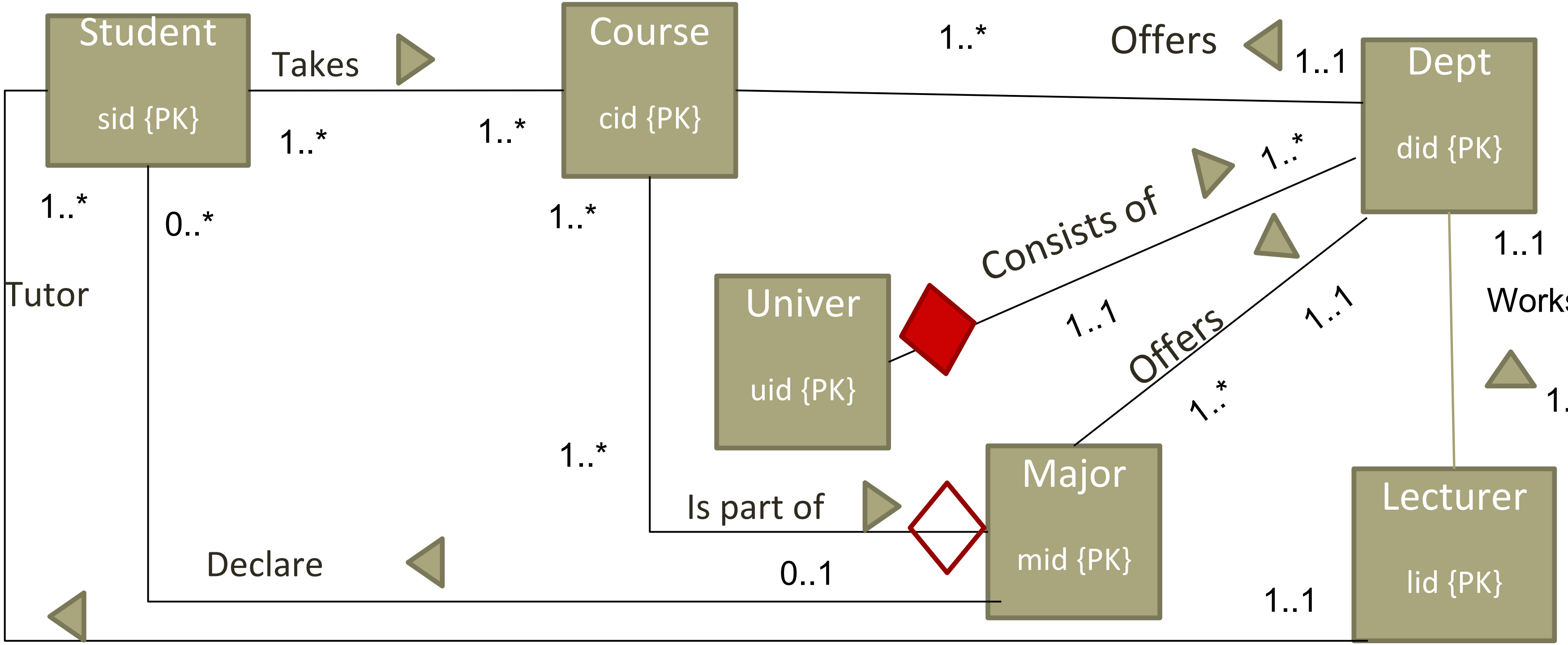
Multiplicities



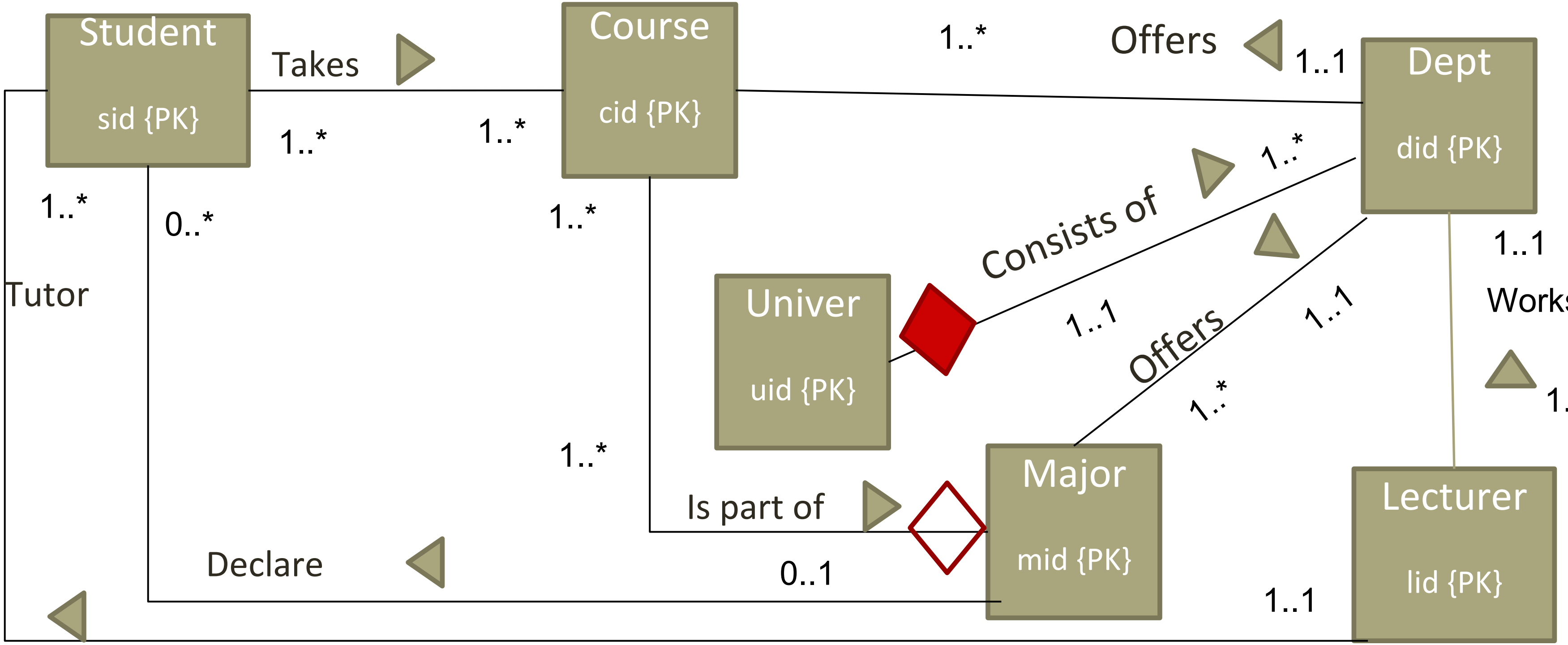
Primary keys are needed



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.

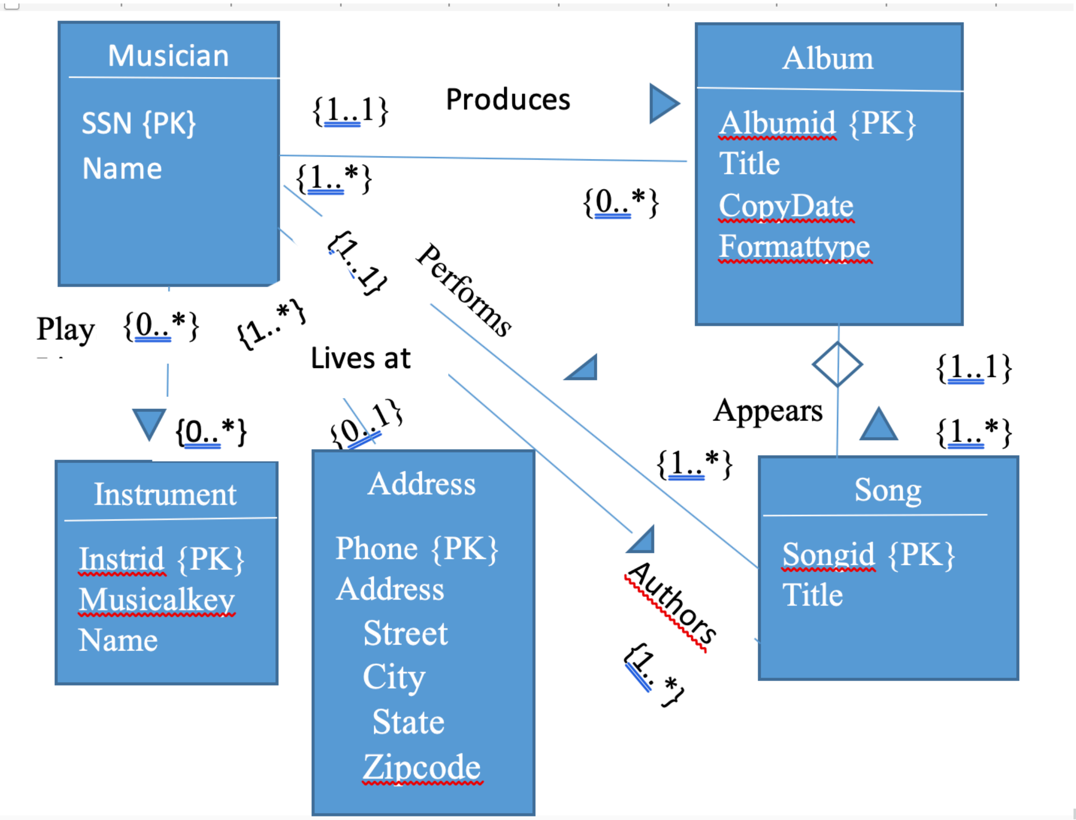
Each album has exactly one musician who acts as its **producer**. A musician may produce several albums, of course.

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 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,
  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, 1^{st}, 2^{nd}, 3^{rd} normal form

<u>StudentNo</u>	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
1	WVH	West Village H
1	NI	Nightingale Hall
2	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: normalization (5)

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

BuildingNo	RoomNo	Abbreviation	BuildingName
1	100	BRK	Behrakis
1	200	WVH	West Village H
1	300	NI	Nightingale Hall
2	100	RY	Ryder Hall

1rd 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