The first step to converting a diagram to 3NF is to create schemas. Therefore, first handle all strong entities.

File(FileID, FileName)
Song(SongID, Title)
Writer(WriterID, W_Name)
Artist(ArtistID, A_Name)
Producer(ProducerID, Prod_Name)
PQUser(PQUser, PQ_Fname, PQ_LName, Pledge)
FUser(FIUser, FI_FName, FI_LName)

Then handle all weak entities, which there are none; Therefore, we tread to the next step which is to handle all binary one-to-one relationships.

There currently is one binary one to one relationship between the entities **File** and **Song**. Regarding the relation **File**, there is one identifier called FileID which serves as the primary key and a single attribute called FileName that represents the single instance of a song's file name. Regarding the relation **Song**, there is an attribute called Title that represents the single instance of a given Title for a Song. There is also an identifier called SongID which serves as the primary key.

There's a catch because there are two ways we could create/represent this table. Since each instance of **File** will have one instance of a **Song**, and each instance of a **Song** will have one instance of **File**, we can represent this one-to-one relationship by putting a new foreign key into the entity for *either side.*

KaraokeFile(FileID, FileName, SongID†)

The following represents the current schema:

Song(SongID, Title)
Writer(WriterID, W_Name)
Artist(ArtistID, A_Name)
Producer(ProducerID, Prod_Name)
PQUser(PQUser, PQ_Fname, PQ_LName, Pledge)
FUser(FIUser, FI_FName, FI_LName)
KaraokeFile(FileID, FileName, SongID†)

The next step is to handle one-to-many relationships. There are two one to many relationships specifically between the entities PQUser and File as well as FUser and File. Therefore, the following is a description that details both relationships.

The first relationship regarding a one-to-many relationship is between the relations **FUser** and **File**. Regarding the relation **FUser**, there are two attributes FIFName and FILName. The attribute FI_FName represents a single instance of a user's first name. The attribute FI_LName represents a single instance of a user's last name. There is a single identifier called <u>FIUser</u>. The relation **File** has an attribute FileName that represents a single instance of a filename for a file and an identifier called <u>FileID</u>. This relationship requires a new relationship to be made called **FIFOQueue**. Within the relation **FIFOQueue**, this relation will be concatenation of the primary keys from the relations **FUser** and **File**. Any intersection data is put into this new relation as a non-prime attribute. The following is the schema:

FIFOQueue(<u>FIUser†</u>, <u>FileID†</u>, Time)

The second relationship regarding a one-to-many relationship is between the relations **PQUser** and **File**. Regarding the relation **PQUser**, there are three attributes Pledge, PQ_FName, and PQ_LName. There's a single identifier called <u>PQUser</u>. The attribute Pledge represents the single instance of a user's pledge amount. The attribute PQFName represents a single instance of a user's first name. The attribute PQ_LName represents a single instance of a user's last name. Regarding the relation **File**, there's one identifier called <u>FileID</u> and an attribute FileName which represents the single instance of a filename. This relationship requires a new relationship to be made called **PriorityQueue**. Within the relationship **PriorityQueue**, this relation will be a concatenation of the primary keys from the relations **PQUser** and **File**. Any intersection data is put into this new relation as a non-prime attribute. The following is the schema:

PriorityQueue(<u>PQID†</u>, <u>FileID†</u>, Time)

Therefore, we move on to handle binary many-to-many relationships. Thus, there are three binary one to many relationships.

The first relationship regarding a binary many-to-many relationship is between the entities **Song** and **Writer**. Regarding the relation **Song**, there is an attribute called Title that represents the single given instance of a song title. There is also an identifier associated with the relation **Song** called <u>SongID</u> which serves as the primary key. Regarding the relation **Writer**, there is an identifier called <u>WritterID</u> which serves as the primary key and an attribute called W_Name that represents the name of the writer. This relationship requires a new relation and its foreign key will be the concatenation of the primary keys of each entity relationship; Thus, which will be used as foreign keys to the corresponding tables. The following is the schema.

SongWriter(WriterID†, SongID†)

The second relationship regarding a binary many-to-many relationship is between the entities **Artist** and **Song**. Regarding the relation **Song**, there is an attribute called Title that represents the single given instance of a song title. There is also an identifier associated with the relation **Song** called SongID that serves as the primary key. Regarding the relation **Artist**, there is an attribute called A_Name that will store a single instance of an Artist's name and an identifier called ArtistID that serves as the primary key. This relationship requires a new relation and its foreign key will be the concatenation of the primary keys of each of the entity relationships; Thus, which will be used as foreign keys to the corresponding tables. The following is the schema.

SongArtist(SongID†, ArtistID†)

The third relationship regarding a binary many to many relationship is between the entities **Producer** and **Song**. Regarding the relation **Song**, there is an attribute called Title that represents the single given instance of a song title. There is also an identifier associated with the relation **Song** called SongID which serves as the primary key. Regarding the relation **Producer**, there is an attribute called Prod_Name that will store a single instance of a Producer's name and an identifier called ProducerID which serves as the primary key. This relationship requires a new relation and its foreign key will be the concatenation of the primary keys of each of the entity relationships; Thus, which will be used as foreign keys to the corresponding tables. The following is the schema.

SongProducer(SongID†, ProducerID†)

The following represents our current schema.

SongWriter(SongID†, WriterID†)
SongArtist(SongID†, ArtistID†)
SongProducer(SongID†, ProducerID†)
KaraokeFile(FileName, SongID†)
FIFOQueue(FIUser†, FileID†, Time)
PriorityQueue(PQID†, FileID†, Time)

Concluding forming schemas, that's it since relations greater than binary without any "ones" cease to exist and recursive relationships cease to exist as well. Therefore, The following describes which primary key's are foreign keys to which relation.

The primary key of the relation **Song** is a foreign key to the relation **KaraokeFile**.

The primary key of the relations **Song** and **Writer** are now foreign keys to the relation **SongWriter**.

The primary key of the relations **Song** and **Artist** are now foreign keys to the relation **SongArtist**.

The primary key of the relations **Song** and **Producer** are now foreign keys to the relation **SongProducer**.

The primary key of the relations **File** and **PQUser** now represent the foreign key in the relation **PriorityQueue**.

The primary key of the relations **File** and **FUser** now represent the foreign key in the relation **FIFOQueue**.

Now convert schemas to 3NF; Hence the following represents the schema converted to 1NF based on the newly created relations.

SongWriter(SongID†, WriterID†)
SongArtist(SongID†, ArtistID†)
SongProducer(SongID†, ProducerID†)
KaraokeFile(FileName, FileName, SongID†)

To convert the schema to 2NF, all of the non-prime attributes in a relation must be fully dependent upon the entire primary key. No non-prime attributes may be functionally determined by any subset of the primary key. No partial key dependencies.

Regarding the relation **SongWriter**, the primary keys from the relations **Song** and **Writer** are concatenated as the foreign key within the relation **SongWriter**. All of the non-prime attributes are fully dependent upon the entire primary key. No non-prime attributes functionally determine any subset of the primary key. No partial key dependencies. Therefore, this relation is in 2NF.

Regarding the relation **SongArtist**, the primary keys from the relations **Song** and **Artist** are concatenated as the foreign key within the relation **SongArtist**. All of the non-prime attributes are fully dependent upon the entire primary key. No non-prime attributes functionally determine any subset of the primary key. No partial key dependencies. Therefore, this relation is in 2NF.

Regarding the relation **SongProducer**, the primary keys from the relations **Song** and **Producer** are concatenated as the foreign key within the relation **SongProducer**. All of the non-prime attributes are fully dependent upon the entire primary key. No non-prime attributes functionally determine any subset of the primary key. No partial key dependencies. Therefore, this relation is in 2NF.

Regarding the relation **KaraokeFile**, since each instance of a **Song** will have a **File**, and each instance of **File** will have one of of **File** through the relationship "is stored in"; Thus, this one-to-one relationship is now represented as **KaraokeFile** by putting a new foreign key into the entity **File**. Thus, this relationship is represented as the new relation **KaraokeFile**. All of the non-prime attributes are fully dependent upon the entire primary key. No non-prime attributes functionally determine any subset of the primary key. No partial key dependencies. Therefore, this relation is in 2NF.

Regarding the relation **PriorityQueue**, since there are many instances of a **PQUser** that can only select one distinctive **File**; Thus creating instance data to the exact time they selected the song (i.e the attribute time), which validates the occurrence of this relationship between the relations **PQUser** and **File**. Therefore, this one-to-many relationship is represented as the new relation **PriorityQueue**. All of the non-prime attributes are fully dependent upon the entire primary key. No non-prime attributes functionally determine any subset of the primary key. No partial key dependencies. Therefore, this relation is in 2NF.

Regarding the relation **FIFOQueue**, since there are many instances of a **FUser** that can select only one distinctive **File**; Thus creating instance data to the exact time they selected the song (i.e the attribute time), which validates the occurrence of this relationship between the relations **FUser** and **File**. Therefore, this one-to-many relationship is represented as the new relation **FIFOQueue**. All of the non-prime attributes are fully dependent upon the entire primary key. No non-prime attributes functionally determine any subset of the primary key. No partial key dependencies. Therefore, this relation is in 2NF.

The next step is to process the schemas to 3NF. Therefore, since these relations qualify for 2NF, they are indeed in 3NF. For example, There are no transitive dependencies. For example, no non-prime attribute may functionally determine another non-prime attribute. Thus, tThe following is the schema in 3NF.

SongWriter(SongID†, WriterID†)
SongArtist(SongID†, ArtistID†)
SongProducer(SongID†, ProducerID†)
KaraokeFile(FileID, FileName, SongID†)
FIFOQueue(FIUser†, FileID†, Time)
PriorityQueue(PQID†, FileID†, Time)

The following represents all of the relations made:

File(FileID, FileName)
Song(SongID, Title)
Writer(WriterID, W_Name)
Artist(ArtistID, A_Name)
Producer(ProducerID, Prod_Name)
PQUser(PQUser, PQ_FName, PQ_LName, Pledge)
FUser(FIUser, FI_FName, FI_LName)
SongWriter(SongID†, WriterID†)
SongArtist(SongID†, ArtistID†)
SongProducer(SongID†, ProducerID†)
KaraokeFile(FileID, FileName, SongID†)
FIFOQueue(FIUser†, FileID†, Time)
PriorityQueue(PQID†, FileID†, Time)

The following relations we need for our database:

Song(SongID, Title)
Writer(WriterID, W_Name)
Artist(ArtistID, A_Name)
Producer(ProducerID, Prod_Name)
PQUser(PQUser, PQ_FName, PQ_LName, Pledge)

FUser(FIUser, FI_FName, FI_LName)
SongWriter(SongID†, WriterID†)
SongArtist(SongID†, ArtistID†)
SongProducer(SongID†, ProducerID†)
KaraokeFile(FileID, FileName, SongID†)
FIFOQueue(FIUser†, FileID†, Time)
PriorityQueue(PQID†, FileID†, Time)