|  |  |
| --- | --- |
|  | **2016** |
| Joshua Rising  Haleigh Robbins  Christian Smith | Group 17 is: |

|  |
| --- |
| **[Database Design of an Online music marketplace]** |
| Phase 1: Entity-Relationship Modeling, Diagraming, and Conversion to the Relational Model Schema |

**Database Design for an**

**Online Music Marketplace**

**Domain:** Online Music Marketplace

Our project focuses on the concept of an online music marketplace. Essentially this is a catalog of artists, albums, and individual tracks with the addition of a buying and selling component. Users of the website are able to create accounts that will allow them to interact with other uses and the database itself. For instance, users are able to list sales from their own collection of albums which other users can inquire about and potentially purchase should the buyer and seller agree on a price. In addition to this user interaction, users alter the database by being able to add comments, rate albums/songs and also rate other users they’ve had transactions with. The result is a user focused and user promoted marketplace that builds a community of music lovers and consumers.

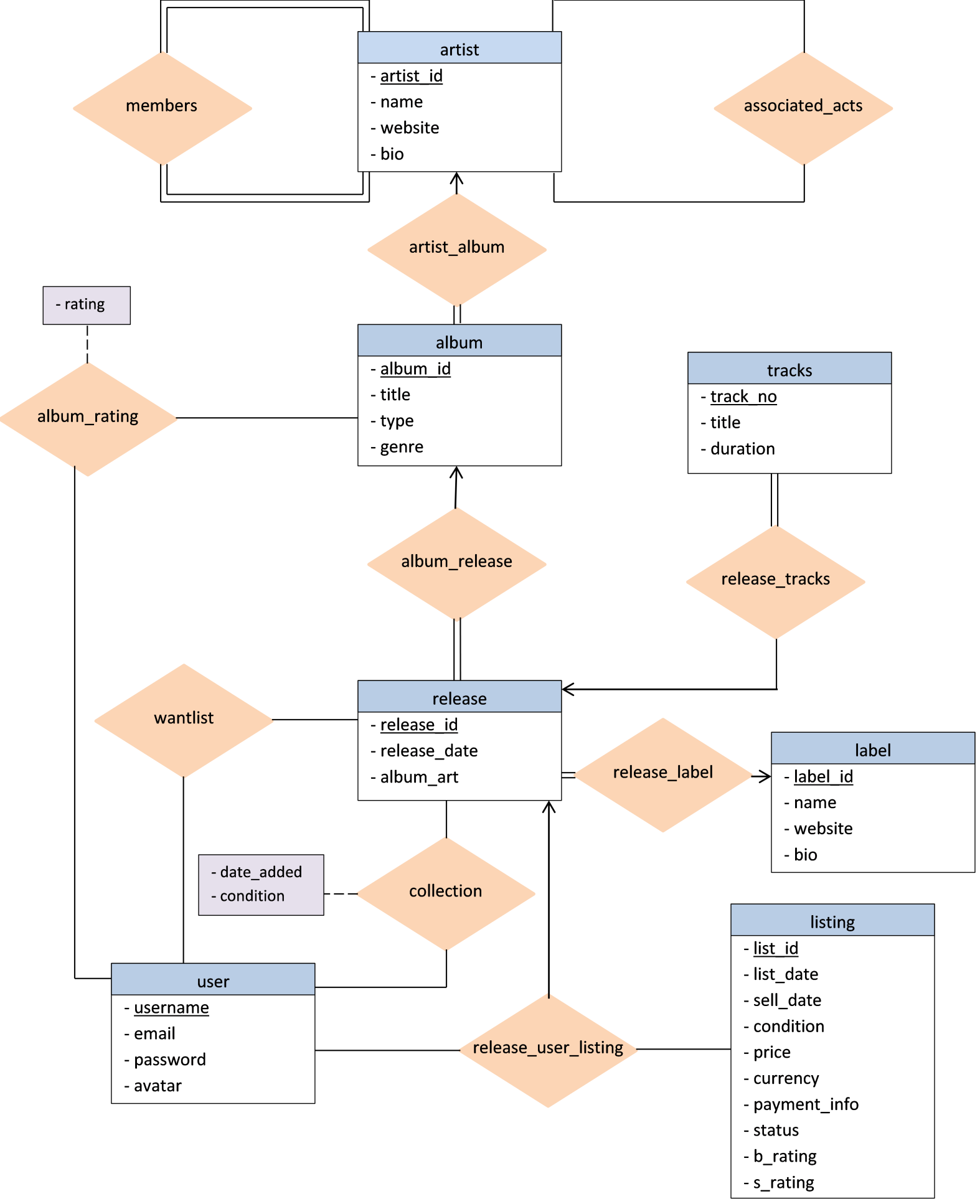
The scope is fairly all encompassing for this project. Little is left out aside from features not directly relevant to the marketplace component. These would be things such as personal user information beyond their usernames and passwords, for example. Generally speaking though, we intend to implement all the functional components necessary for a user oriented online marketplace. One exception to this would be the use of third parties for handling the actual flow of money between customers. Suffice it to say, Paypal or some similar service is assumed to be responsible for this particular component.

**User Requirements:**

* A music marketplace must include a catalog of the music that will be for sale. As such, the logical first step is to divide music into artists. Every artist has a unique id (*artist\_id*) that is used to differentiate it from others. Additionally this unique ID allows for multiple artists that have the same name to exist in the database. In addition to their IDs, artists have an associated *name* as well as a list of members and associated acts. Some other information such as their *website* and a *bio* are included as well.
* Artists have albums associated with them. Albums have a unique id (*album\_id*) to differentiate them from one another. Again this has the effect of preventing name collisions in the event an artist releases two or more albums with the same name. A *title* is also attached in addition to some characteristic information such as *genre* and *album\_rating*. Also important is the *type*, which differentiates between LPs, EPs, singles, and other types of albums.
* An important distinction to make is the difference between an album and an individual release of that album. This is important because albums are often reissued, pressed by different labels, or have other differences that are relevant for buyers and sellers to be aware of. Every release has a unique ID (*release\_id*). Other important information about the release such as its *release\_date* and issuing label are also included. The *medium* of the release is also included to differentiate between physical formats. Finally, *album\_art* is attached to each release because it potentially provides an easy way to identify different releases when users browse the marketplace for some item.
* Releases are made up of individual tracks. Tracks are associated with releases rather than albums because different releases may have different tracklists or different ordering for the tracks, yet still are, technically, the same album. Tracks have a track number (*track\_no*), *title* and *duration*.
* To help with the categorization of releases, it is also necessary to group labels. Each label has an ID (*label\_id*) and *name*. A list of *sites* and a *bio* are used to give users more information about the labels which can be useful for users looking to find labels specializing in certain types of music.
* Users are what allow for the community. Each user has a unique *username* as well as a *password*. There is also an *avatar* for some personalization.
* Listings are created when a user lists an item for sale on the marketplace. Each sale item must refer back to the seller as well as including release information. Specific to the sale item are its *condition* (mint, used, etc.), the list *price*, the type of *currency* (so that users can sell globally), and its *list\_date*. Also needed is a block of text that explains *payment\_info* which will allow sellers to discuss shipping information as well as methods of payment. When a customer purchases a listing, the listing is updated to contain the *buyer\_name* and the *sell\_date*. Buyers and sellers can also rate each other since the listing is now a transaction.

**Relationships:**

* Listings are relationships between two users and a release. Therefore this relationship has ternary cardinality. A release can have multiple sellers and a seller can sell multiple releases, which means this is a many-to-many relationship. Additionally, buyers can buy multiple listings. Full participation is not required since some items may not currently be up for sale.
* Members connect two artist IDs so that individual band members can be searched. The advantage of the relationship is that it allows users to find other bands a particular member they’re interested in is also a part of. It can be combined with associated acts to achieve this functionality.
* Associated acts provide a way of finding other bands an artist is a part of or otherwise involved with. It’s an excellent way for users to find similar music and broaden their horizons.
* Album ratings are a range given to an album by users. This allows for users to rate albums for their own use as well as making it possible to find the average rating for an album and the total number of people who have rated an album. Users may want to browse the marketplace for albums that scored above a certain rating and this relationship lets this functionality exist.
* Wantlists are a way for user to catalogue the items that they are interested in purchasing. They are able to be browsed by sellers so that sellers are able to contact users with sale offers should they want to take a more direct sales approach.
* Finally there are the collections. Much like wantlists, these are user catalogues, only this time they contain owned items. Potential buyers can view seller collections to find items a seller owns that aren’t yet listed on the marketplace. This provides an opportunity for buyers to take the initiative. Additionally, sellers browsing user collections may be able to recommend buyers new albums based on the kinds of music a user has in their collection. All in all, wantlists and collections provide an excellent way to broaden the avenues for sales on the marketplace.

****

**­­­**

**Entity-Relationship Model:**

Converting the diagram to the entity-relationship model gives this schema

artist (*artist\_id, name, website, bio*)

members (*artist\_id, member\_id*)

associated\_acts (*artist\_id, associated\_act\_id*)

artist\_album (*artist\_id, album\_id*)

album (*album\_id, title, type, genre*)

album\_rating (*username, album\_id, rating*)

album\_release (*album\_id, release\_id*)

release (*release\_id, release\_date, medium, album\_art*)

release\_tracks (*release\_id, track\_no*)

tracks (*track\_no, title, duration*)

release\_label (*release\_id, release\_date, medium, album\_art, label\_id*, name, website, bio)

label (*label\_id, name, website, bio*)

user (*username, email, password, avatar*)

wantlist (*username, release\_id*)

collection (*username, release\_id, date\_added, condition*)

listing (*list\_id, list\_date, sell\_date, condition, price, currency, payment\_info, status, b\_rating, s\_rating*)

All many-to-one and one-to-many relationships are redundant by nature. Therefore, artist\_album, album\_release, release\_tracks, and release\_label are all redundant. By simply adding the primary key from the one side to the many side we can remove this extra relationships.

Wantlist and collection seem to be redundant to each other, and while they are in some senses, they represent different information and will be kept because of this distinction.

The entity-relationship set release\_user\_listing will keep the same functionality by giving listing foreign keys from release and two users.

Finally, members and associated\_acts will also stay, despite being redundant, because they are conversions of multi-valued attributes that are related to artist. There is no other way to keep this information without some redundancy, unfortunately.

**Conversion to the Relational Model:**

artist (*artist\_id, name, website, bio*)

* *artist\_id* is an integer and cannot be null
* *name* is a varchar of 100 and cannot be null
* *website* is a single value varchar of 100 and can be null
* *bio* is a clob of 10 Kb and can be null

Keys:

* primary key = (*artist\_id*)

members (*artist\_id, member\_id*)

* artist\_id is an integer and cannot be null
* member\_id is an int and cannot be null

Keys:

* primary key = (*artist\_id, member\_id*)
* *artist\_id* is a foreign key that references artist
* *member\_id* is a foreign key that references artist

associated\_acts (*artist\_id*, *associated\_act\_id*)

* artist\_id is an integer and cannot be null
* associated\_act\_id is an integer and cannot be null

Keys:

* primary key = (*artist\_id*, *associated\_act\_id*)
* *artist\_id* is a foreign key that references artist
* *associated\_act\_id* is a foreign key that references artist

album\_rating (*username, album\_id, rating*)

* username is a varchar of 30 and cannot be null
* album\_id is an int and cannot be null
* rating is an int and cannot be null

Keys:

* primary key = (*username, album\_id*)
* *username* is a foreign key that references user
* *album\_id* is a foreign key that references album

album (*album\_id, artist\_id, title, type, genre*)

* *album\_id* is an integer and cannot be null
* *artist\_id* is an integer and cannot be null
* *title* is a varchar of 50 and cannot be null
* *type* is a varchar of 10 and cannot be null
* *genre* is a single value varchar of 20 and cannot be null

Keys:

* primary key = (*album\_id*)
* *artist\_id* is a foreign key that references artist

release (*release\_id, album\_id, release\_date, label\_id, medium, album\_art*)

* *release\_id* is an integer and cannot be null
* *album\_id* is an integer and cannot be null
* *release\_date* is a type date and cannot be null
* *label\_id* is an integer and cannot be null
* *medium* is a varchar of 20 and cannot be null
* *album\_art* is a varchar of 100 containing a relative path to an image file

Keys:

* primary key = (*release\_id­­­­*)
* *album\_id* is a foreign key that references album

track (*release\_id, track\_no, title, duration*)

* *release\_id* is an integer and cannot be null
* *track\_no* is an integer and cannot be null
* *title* is a varchar of 100 and can be null (null defaults to ‘-‘)
* *duration* is an integer (time in seconds)

Keys:

* primary key = (*release\_id, track\_no*)
* *release\_id* is a foreign key that references release

label (*label\_id, name, website, bio*)

* *label\_id* is an integer and cannot be null
* *name* is a varchar of 100 and cannot be null
* *website* is a varchar of 50 and can be null
* *bio* is a clob 10 Kb

Keys:

* primary key = (*label\_id*)

user (*username, email, password, avatar*)

* *username* is a varchar of 30 and cannot be null
* *email* is a varchar of 30 and cannot be null
* *password* is a varchar of 25 and cannot be null
* *avatar* is a varchar of 100 containing a relative path to an image file

Keys:

* primary key = (*username*)

listing (*list\_id, release\_id, seller, buyer, list\_date, sell\_date, condition, price, currency, payment\_info, status, b\_rating, s\_rating*)

* *list\_id* is an integer and cannot be null
* *release\_id* is an integer and cannot be null
* *seller* is a varchar of 30 and cannot be null
* *buyer* is a varchar of 30 and can be null
* *list\_date* is a type date and cannot be null
* *sell\_date* is a type date and can be null
* *condition* is a varchar of 15 and cannot be null
* *price* is a numeric (6,2) and cannot be null
* *currency* is a char of 3 and cannot be null
* *payment\_info* is a clob of 10kb
* *status* is a varchar of 6 and can be either “active” or “sold”
* *b\_rating* is an integer representing the buyer’s rating of the seller
* *s\_rating* is an integer representing the seller’s rating of the buyer

Keys:

* primary key = (*list\_id*)
* *release\_id* is a foreign key that references release
* *seller* is a foreign key that references user
* *buyer* is a foreign key that references user

wantlist (*username, release\_id*)

* *user\_id* is a varchar of 30 and cannot be null
* *release\_id* is an integer and cannot be null

Keys:

* primary key = (*username, release\_id*)
* *username* is a foreign key that references user
* *release\_id* is a foreign key that references release

collection (*username, release\_id, date\_added, condition*)

* *user\_id* is a varchar of 30 and cannot be null
* *release\_id* is an integer and cannot be null
* *date\_added* is type date and cannot be null
* *condition* is a varchar

Keys:

* primary key = (*username, release\_id, date\_added*)
* *username* is a foreign key that references user
* *release\_id* is a foreign key that references release

**Example Operations:**

Queries:

* Find all albums on the marketplace with a rating of 4 or above
* List all albums released by a particular label
* List all sale items from a particular seller
* Find all sale items for a particular release

Updates:

* Add a new artist to the database
* Change the album art for a release
* Change the list price of a sale item
* Register a new user

**Normalization:**

artist (*artist\_id, name, website, bio*)

* Primary key: (*artist\_id*)
* Functional dependencies: *artist\_id* ―› *name, website, bio*
* Because there is only one functional dependency and it is a primary key artist is in BCNF

members (*artist\_id, member\_id*)

* Primary key: (*artist\_id, member\_id*)
* Functional dependencies: *artist\_id, member\_id* ―› *artist\_id, member\_id*
* The only functional dependency is a trivial dependency so members is in BCNF

associated\_acts (*artist\_id, associated\_act\_id*)

* Primary key: (*artist\_id, associated\_act\_id*)
* Functional dependencies: *artist\_id, associated\_acts\_id* ―› *artist\_id, associated\_acts\_id*
* The only functional dependency is a trivial dependency so associated\_acts is in BCNF

album\_rating (*username, album\_id, rating*)

* Primary key: (*username, album\_id*)
* Functional dependencies: *username, album\_id* ―› *rating*
* The only functional dependency is a primary key, therefore album\_rating is in BCNF

album (*album\_id, artist\_id, title, type, genre*)

* Primary key: (*album\_id*)
* Functional dependencies: *album\_id* ―› *title, type, genre*
* The only functional dependency is a primary key, therefore album is in BCNF

release (*release\_id, album\_id, release\_date, label\_id, medium, album\_art*)

* Primary key: (*release\_id*)
* Functional dependencies: *release\_id* ―› *album\_id, release\_date, label\_id, album\_art*
* The only functional dependency is a primary key, therefore release is in BCNF

tracks (*release\_id, track\_no, title, duration*)

* Primary key: (*release\_id, track\_no*)
* Functional dependencies: *release\_id, track\_no* ―› *title, duration*
* The only functional dependency is a primary key, therefore tracks is in BCNF

label (*label\_id, name, website, bio*)

* Primary key: (*label\_id*)
* Functional dependencies: *label\_id*  ―› *name, website, bio*
* The only functional dependency is a primary key, therefore label is in BCNF

user (*username, email, password, avatar*)

* Primary key: (*username*)
* Functional dependencies: *username* ―› *email, password, avatar*

*email* ―› *username, password, avatar*

* Both functional dependencies are super keys, therefore user is in BCNF

wantlist (*username, release\_id*)

* Primary key: (*username, release\_id*)
* Functional dependencies: *username, release\_id* ―› *username, release\_id*
* The only functional dependency is trivial, therefore wantlist is in BCNF

collection (*username, release\_id, date\_added, condition*)

* Primary key: (*username, release\_id, date\_added*)
* Functional dependencies: *username, release\_id, date\_added* ―› *condition*
* The only functional dependency is a primary key, therefore collection is in BCNF

listing (*list\_id, release\_id, seller, buyer, list\_date, sell\_date, condition, price, currency, payment\_info, status, b\_rating, s\_rating*)

* Primary key: (*list\_id*)
* Functional dependencies: *list\_id* ―› *release\_id, seller, buyer, list\_date, sell\_date, condition,*

*price, currency, payment\_info, status, b\_rating, s\_rating*

* The only functional dependency is a primary key, therefore listing is in BCNF

**View Definitions:**

This view will display all transactions a particular user has been the buyer in.

CREATE VIEW user\_transactions AS

SELECT transaction\_id, seller\_name, date, cost, buyer\_rating

FROM transaction

WHERE buyer\_name = (some specified username)

This view will display all the items in a particular user’s collection.

CREATE VIEW user\_collection AS

SELECT title, type, album\_art, date\_added, condition

FROM collection NATURAL JOIN release NATURAL JOIN album

WHERE username = (some specified username)

**Query Definitions:**

Changes I’m considering:

1. Transaction should include either a foreign key to a release\_id ***Done***
2. Maybe give listing an id since it would be easier to refer to for transactions
3. Release should have a medium attribute which determines things like CD, vinyl, tape ***Done***
4. Type in album should only refer to LP, EP, single, etc rather than include medium ***Done***
5. Add album\_rating relation to add some more functionality to the database for users ***Done***
6. Storing images in databases seems to be frowned upon. So all images will be stored as relative file paths to the image itself. ***Done***
7. Fix E-R diagram to reflect changes
8. E-R diagram should not display foreign keys
9. Possible listing/transaction changes. Listings replaces transaction entirely. Each listing has an ID, release\_id, seller, buyer, list\_date, sell\_date, condition, price, currency, payment\_info, and status. Buyer, and sell\_date default to NULL on the initial listing. These fields are updated when a listing is sold. Ratings will get moved to a separate relation sale\_ratings which will just be a listing ID and a buyer\_rating and seller\_rating attributes. Rating columns can be NULL and can only be updated by the appropriate users. i.e. if username == buyer for an ID they only have access to updating the buyer rating column. Likewise for sellers.

Mockup below:

listing(*list\_id, release\_id, seller, buyer, list\_date, sell\_date, condition, price, currency, payment\_info, status*)

Where list\_id is the primary key and completely unique for every listing.

Release\_id is a foreign key referencing release  
list\_id is an integer, release\_id is an integer, seller is a varchar, buyer is a varchar, list\_date is a date, buy\_date is a date, condition is a varchar, price is a numeric (6,2), currency is a char(3), payment\_info is a clob, status is a varchar.