RECORD COMPANY DATABASE

CS342 Database Project – Fall 2015

Mark Armendariz and Andrew Lane

Table of Contents

Phase 1: Fact-Finding, Information Gathering, and Conceptual Database Design

1.1 Fact-Finding Techniques and Information Gathering

1.1.1 Introduction to Enterprise

The imaginary company our database will be created for is for a record company called Armenlane Records. Record companies, also known as “Record Labels”, are companies that hire musicians and artists by contract to record albums. These contracts have certain parameters, such as having a contract that is for a specific number of albums. After the albums are recorded, Armenlane Records releases the albums by selling them to major retailer distribution centers, such as Wal-Mart, Target, or Kmart. The main type of music our company focuses on is rock music and other similar types, or genres, of music. This company rents out a small number of independently-owned recording studios for each album to be recorded at. Each song also has a songwriter.

1.1.2 Description of Fact-Finding Techniques

For fact-finding, we used the internet and popular search websites in order for us to find and use information for our project. To find more specific information on record companies, we researched a few modern record companies, as well as legal information on contracts and the music business as a whole.

We were able to get into contact with Asthmatic Kitty Records, which is a small independent record company based in the United States. We emailed them for information on

data that is stored for their company. They weren't that specific in the information they provided, but they gave us the general idea of what data they store in their databases, such as artist contract information, as well as album information.

We were also able to get into contact with a musician who has recorded music at actual studios and he was able to give us more information on what information is stored, such as information on songs and writers.

From all of our research, we discovered that the music business is a very complex business. Every company's database may be different but they are probably very complex due to how complex the music business is, so for the purpose of this project, we will keep things simple.

Contracts can be very complicated, especially when it comes to financial information and income from music sales. But upon researching, we found that contracts often involve terms where an artist will have only a certain number of albums they can record under their current contract. We also discovered that record companies don't actually own recording studios anymore, but rent independent ones for use.

We also used the basic, and common structure of artists creating albums, which all have at least one song, with each song having a writer. Entities were easy to discover once we laid out what a record company does and what each artist does.

The reports that would need to be generated are numerous, but we came to identify a few important ones. For example, general artist and album information reports for all artists hired by the company would need to be generated, as well as financial reports and studio usage reports would be generated. To generate a report, the user will specify what items they would

like to see. For example, they could see all of the songs from a specific album, or view all the album transactions that occurred during a specific period of time.

1.1.3 Database Design Focus

Record companies can be very complicated, especially when dealing with financial information. So for this software, we decided to focus more on music part of the record company. Rather than covering the entire business, we will only focus on the following: The artist and their contract information, the album and their songs, studios and songwriters, as well as transactions for selling albums.

Keeping the main focus on artists and their music allows us to keep the project simple. For example, if we wanted to view information for an artist who is hired by contract by the company, we would include information based only on their role in music-making. But by including some financial information, we are able to make a more complete database.

1.1.4 Entity and Relationship Set Description

Our time in lecture helped us develop a more solid approach to how an entity should be created. During the lab period, we watched a few groups present their ideas for their proposed database. During each presentation, we discussed how each idea could possibly be improved, or if there was in issue with the current model. Using this experience, we tried to improve our own ideas and rough drafts of our entities.

So using the information we discovered from research, we decided to focus on the following entities: Contracts, Artists, Members, Albums, Songs, Studios, Writers, Transactions, and Buyers.

The Contract entity represent the conditions by which an Artist is hired by the record company. The Artist entity represents the group or individual that is hired by the record company under a contract with a certain number of albums as their terms. Each artist will have at least one member in the Member entity.

For example, if a band, called The Beatles, is hired by the company, and the band has four members, John, Paul, George, and Ringo, the Artist entity would contain an entry for The Beatles, but the Member entity would contain four entries, one for each member of The Beatles. Another example is if a single artist, called Bob Dylan, is hired by the company, then the Artist entity would contain a record for Bob Dylan and the Member entity would also contain just one entry for Bob Dylan.

Each Artist will have albums that they have recorded. Each of these Albums will have a Studio where it was recorded. Each Album will be sold to major retailer distribution centers. Each of these transactions will kept track of in the Transaction entity. Each album will also have a set of songs, and each song will have a writer.

All of these entities combine to create a database of hired artists without getting too complex for this quarter's project. It is a very general database, containing general artist and music information as well as some financial information.

1.2 Conceptual Database Design

1.2.1 Entity Set Description

Contract – Strong Entity

Our Record Company has hundreds of artists hired by contract to the record company, with each of these contracts having their own terms and a date when the contract started and ended. So a Contract entity was required.

Figure 1- Contract Entity

Primary Key: contract_ID

Attributes:

- contract_ID: Internal ID for organizing artist contracts
  - **String**; no nulls; unique; single-valued and simple.

- album_term: Number of albums artist has to record for their contract
  - **int**; nulls not allowed; not unique; single-valued and simple.

- start_date: Date artist signed contract
  - **date**; no nulls; not unique; single-valued and simple.

- end_date: Date artist contract ended
  - **date**; nulls allowed; not unique; single-valued and simple.

Artist – Strong Entity

Our Record Company has hundreds of artists hired by contract to the record company. So we needed an Artist Entity to represent these artists in our database.

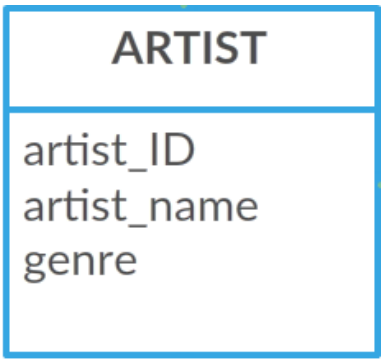


Figure 2- Artist Entity

Primary Key: artist_ID

Attributes:

- artist_ID: Internal ID for artist information
  - **string**; no nulls; unique; single-valued and simple.

- artist_name : Name used by artist
  - **string** with any value; no nulls, not unique; single-valued and simple.

- genre: Style of music classification
  - **string** with any value; nulls allowed; not unique; single-valued and simple.

Member – Strong Entity

Artists may have more than one member as part of their group, or they may just be a single person as the only member of the artist. For this reason, a Member Entity is required.

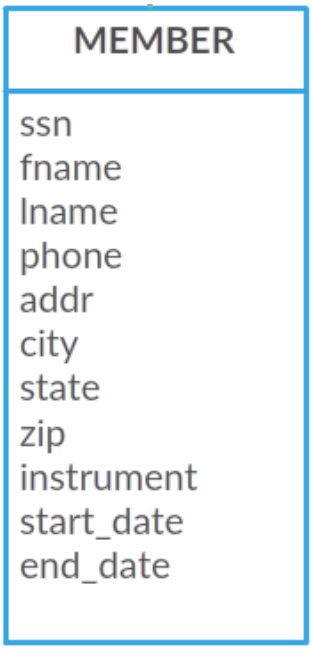


Figure 3 - Member Entity

Primary Key: ssn

Attributes:

- ssn: - Social Security Number unique to each person
    - **String** with length of 9 characters; no nulls; unique.
- fname : First name
    - **string** with any value; no nulls; not unique; single-valued and simple.
- lname: Last name
    - **string** with any value; no nulls; not unique.
- phone: Phone number
    - **string** with length of 10 chars; nulls allowed; not unique; single-valued and simple.
- addr: Street address
    - **string** of length 40 chars; nulls not allowed; not unique.
- city: Specific city in U.S. State
    - **string** of length 40 chars; nulls not allowed; not unique.
- state: State located in the U.S.

- string; nulls not allowed; not unique; single-valued and simple.

- zip: Valid U.S. Zip Code
  - string: nulls not allowed; not unique; single-valued and simple.

- instrument: Instrument that member plays
  - string; null not allowed; not unique; single-valued and simple.

- start_date: date member's contract began
  - Date type; null not allowed; not unique; single-valued and simple.

- end_date: date member's contract ended
  - Date type that is after starting date; nulls allowed; not unique. Single-valued and simple.

Album – Strong Entity

Each Artist will have a certain amount of Albums. So an Albums entity is required in order to store general information for each album recorded by the group.

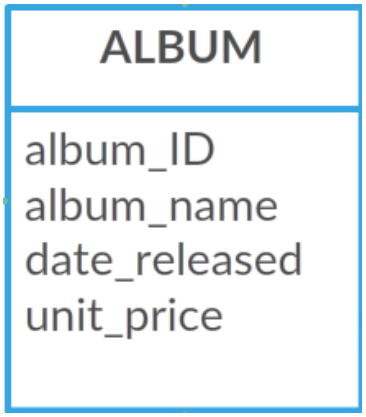


Figure 4 - Album Entity

Primary Key: album_ID

Attributes:

- album_ID: Unique album ID number
  - **string**; no nulls; unique; single-valued and simple.
- album_name : Name of album
  - **string**, no nulls, not unique; single-valued and simple.
- Date_released: Date album released
  - **date**; no nulls; no unique; single-valued and simple
- Unit_price: Price of album sold in stores
  - **float**; nulls allowed; not unique; single-valued and simple.

Song - Strong Entity

Every Album will have a set of songs listed in a particular order with a certain length. So for these reasons, a Song entity is required in order to store each of these songs that are associated with a particular album released by a particular artist.

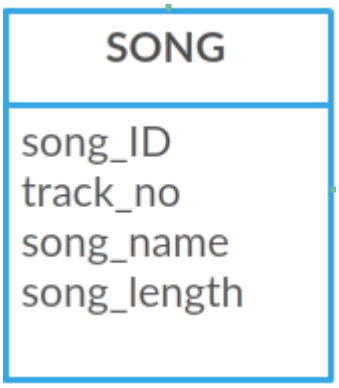


Figure 5 - Song Entity

Primary Key: song_ID

Attributes:

- song_ID: - Internal ID for song organization
  - string; no nulls; unique; single-valued and simple.
- track_no: Track number of song on album
  - int; nulls allowed; not unique; single-valued and simple.
- song_name : Name of the song
  - string; no nulls; not unique; single-valued and simple.
- song_length: Length of song in time
  - time; nulls allowed; not unique; single-valued and simple.

Studio – Strong Entity

This recording company rents out independently-owned studios for artists to use. In order to keep track of where each song was recorded and how much money was spent in recording this a particular albums or on an artist, a Studio entity is required to hold information for each studio.

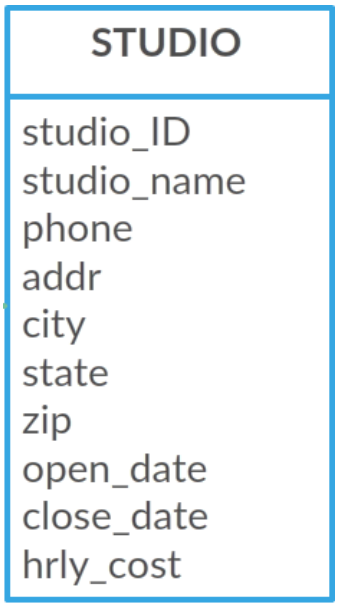


Figure 6 - Studio Entity

Primary Key: studio_ID

Attributes:

- studio_ID: - Internal ID for contract organization
  - **string**; no nulls; unique; single-valued and simple.
- studio_name: Name of studio
  - **string**, no nulls, not unique; single-valued and simple.
- phone: contact number of studio
  - **string**; nulls allowed; not unique; single-valued and simple.
- addr: street address where studio is located
  - **string**; nulls allowed; not unique; single-valued and simple.
- city: city where studio is located
  - **string**; nulls not allowed; not unique; single-valued and simple.

- state: Specific state where studio is located
    - string; nulls not allowed; not unique; single-valued and simple.

- zip: zip code of studio
    - string; nulls not allowed; not unique; single-valued and simple.

- open_date: Date studio opened
    - date; no nulls; not unique; single-valued and simple.

- close_date: Date studio closed
    - date that takes place after open_date; nulls allowed; not unique; single-valued and simple.

- hrly_cost: Hourly cost to rent studio for recording
    - float; nulls not allowed; not unique; single-valued and simple.

Writer – Strong Entity

The recording company will contract out writers to either compose lyrics or tracks for the artists to sing or use. A writer can work on many different songs and is not limited to the amount he can work on unless specified in the contract.

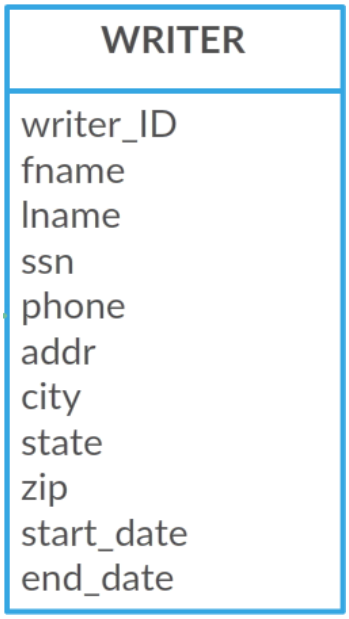


Figure 7 - Writer Entity

Primary Key: writer_ID

Attributes:

- writer_ID: Internal ID for contracted writer
  - **string**; no nulls; unique; single-valued and simple.
- fname: Name of writer
  - **string**, no nulls, not unique; single-valued and simple.
- lname: Last name of writer
  - **string**; nulls allowed; not unique; single-valued and simple.
- ssn: Social security number
  - **string**; nulls allowed; not unique; single-valued and simple.
  -
- addr: Street address of writer
  - **string**; nulls allowed; not unique; single-valued and simple.
- phone: Phone number of writer
  - **string**; nulls allowed; not unique; single-valued and simple.

- city: City where writer lives
    - string; nulls not allowed; not unique; single-valued and simple.

- state: Specific state where writer lives
    - string; nulls not allowed; not unique; single-valued and simple.

- zip: zip code of where writer lives
    - string; nulls not allowed; not unique; single-valued and simple.

- start_date: Date writer signed contract
    - date; no nulls; not unique; single-valued and simple.

- end_date: Date writer contract ended
    - date that takes place after start_date nulls allowed; not unique; single-valued and simple.

Transaction – Strong Entity

Albums will be sold to retail distribution centers through transactions, so a transaction entity is required to hold the date and unique ID for each transaction.

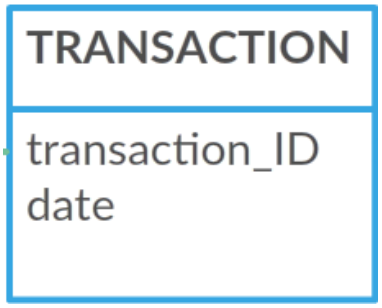


Figure 8- Transaction Entity

Primary Key: transaction_ID

Attributes:

- transaction_ID: Internal ID for transactions
  - **string**; no nulls; unique; single-valued and simple.

- Date: Date when batch of albums was purchased
  - **date**; nulls allowed; not unique; single-valued and simple.

Buyer – Strong Entity

Information for retail distribution centers is required to be stored for keeping transaction information. So a buyer entity is required.

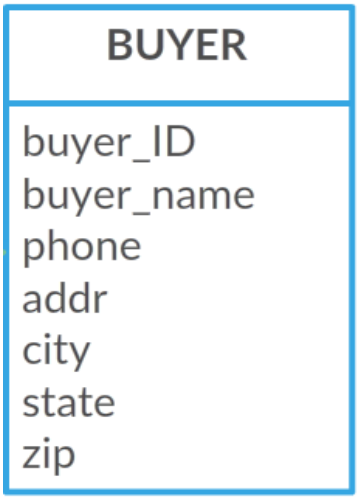


Figure 9- Buyer Entity

Primary Key: buyer_ID

Attributes:

- buyer_ID: Unique buyer ID
  - **string**; no nulls; unique; single-valued and simple.
- buyer_name: Name of major buyer
  - **string**, no nulls, not unique; single-valued and simple.
- phone: phone number
  - **string**; nulls allowed; not unique; single-valued and simple.
- city: City where buyer is located
  - **string**; nulls not allowed; not unique; single-valued and simple.
- state: Specific U.S. state
  - **string**; nulls not allowed; not unique; single-valued and simple.
- zip: zip code
  - **string**; nulls not allowed; not unique; single-valued and simple.

1.2.2 Relationship Set Description

Contract hires Artists

This relationship links each artist to the contract by which they were hired. Each artist has one contract, and each contract has one artist, so it is a One to One relationship. This relationship utilizes the Contract and Artist entities.

Artist is composed of Members

This relationship links each artist to the members to make up that artist. Single artists will have only member in the member entity, and bands will have more than one member in the member entity.

Using the same example from earlier, if the band The Beatles were hired, there would be a single entry in the Artist entity for “The Beatles”, but there would be four entries for the band members John, Paul, George, and Ringo, in the Member entity.

However, this relationship is a Many to Many relationship with any number of artists having any number of members and any number of members have any number artists. So a member can be a part of more than one group or single artist. This relationship utilizes the Artist and Member entities.

Artist records Albums

This relationship links each artist to all the albums they have recorded. It is a One to Many relationship with one artist having any number of albums and any number of albums

having one artist. This relationship utilizes the Artist and Album entities.

Album is composed of Songs

This relationship links each album recorded to all of the songs that are on that album. It is a **One to Many** relationship with one album have any number of songs and any number of songs having one album. This relationship utilizes the Album and Song entities.

Albums sold through Transactions

This relationship links each album to the transactions the recording company makes in order to sell an album to a major retailer. It is a **Many to Many** relationship with any number of albums having any number of transactions, and any number of transactions having any number of albums. This relationship utilizes the Album and Transaction entities.

Transactions purchased by Buyer

This relationship links each transaction to the buyer who purchased the batch of albums. It is a **Many to One** relationship, with any number of transactions being purchased by a single buyer, and one buyer having any number of transactions. This relationship utilizes the Transaction and Buyer entities.

Albums recorded at Studios

This relationship links each album to the recording studio it was recorded at. It is a **Many to One** relationship with any number of albums being recorded at one studio and one

studio having any number of albums. This relationship utilizes the Album and Studio entities.

Songs written by Writers

This relationship links each song to the writer who wrote the song. It is a Many to One relationship with any number of songs written by one writer and one writer writing any number of songs. This relationship utilizes the Song and Writer entities.

1.2.3 Related Entity Set

Generalization is when entities are grouped together to show a more general view. With generalization, we bring together multiple entities into one entity to create a larger entity based on their similar traits and characteristics.

On the other hand, specialization is the opposite of generalization. With specialization, entities are divided into sub classes based on their traits. You can take an entity such as artist, and split it into subclasses like Band or Solo. These are both very similar to the entity Artist.

We thought about possibly keeping Bands and Solo artists separate with two distinct entities, but they have very similar attributes. So we decided to generalize them into a single entity called Artist. What will eventually separate the two separate ideas is the number of members this entity is related to in the Member entity. An artist with one entry in the member entity will be a solo artist, and an artist with more than one entry in the member entity will be considered a group or band.

1.2.4 E-R Diagram

An E-R model, or entity-relationship model, is a visualization of entities and how they are related to other entities. This includes the relationships between entities and their cardinalities. This model allows for entities to be organized in meaningful way.

Referring back to all previous information and development, we created the following E-R Model for this database. It is a very basic and high level understanding of what our company would require for this database. This is tentative, as this design may change as we progress with this project. But, we can now use this model to create a relational database.

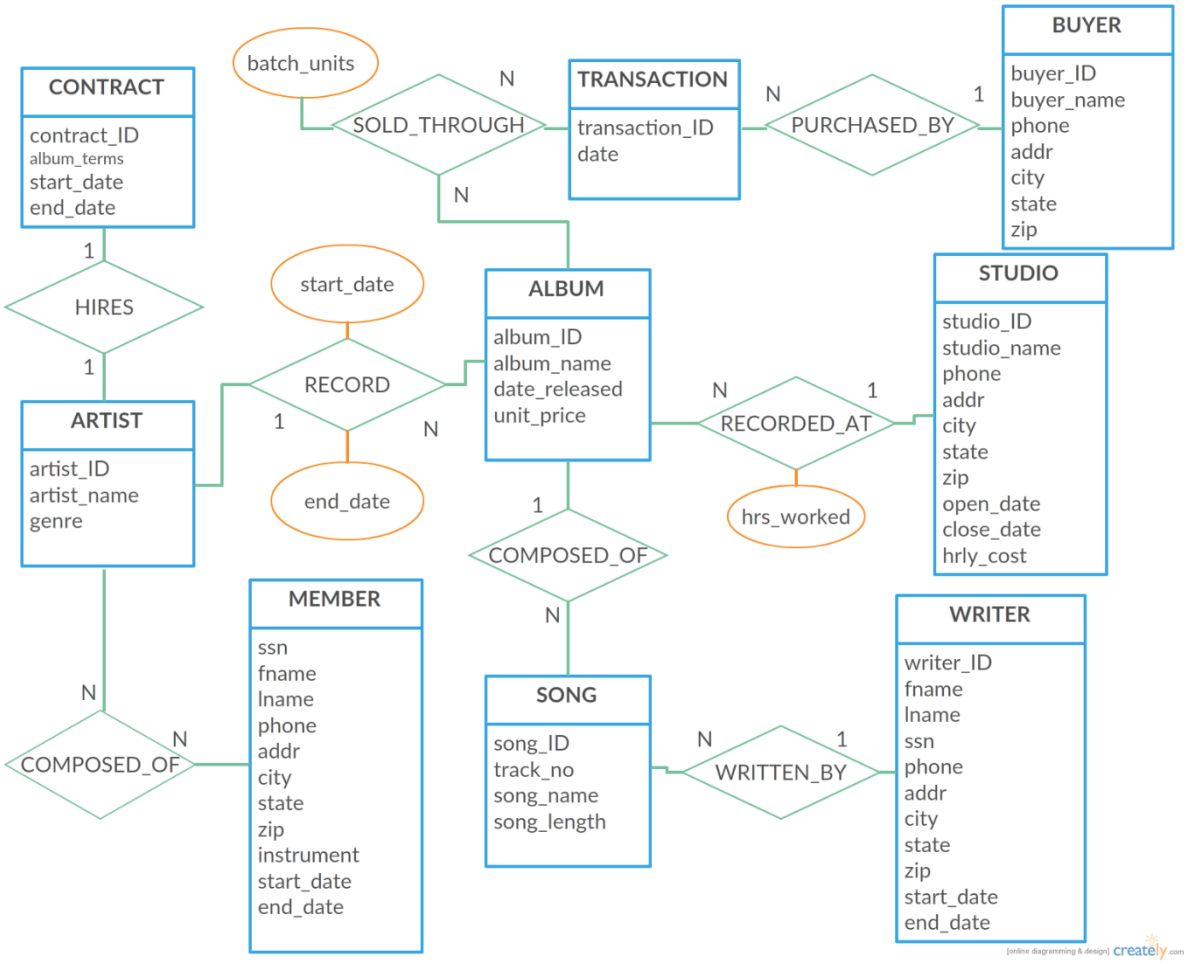


***Figure 10**- E-R Model of Record Company Database*

Phase 2: From E-R Model to Relational Model

2.1 Conceptual Database and Logical Database

2.1.1 E-R Model and Relational Model

E-R Model

The Entity-Relationship model was created by Peter Chen, a Computer Science and Applied Mathematics graduate from Harvard. Chen created the model to create a formal approach on data modeling in order to develop a database.

The E-R model has a few major features that assist in the development and visualization process of creating a database. E-R modeling allows the developer to create a graphical representation of how their database will look. Creating Entities which hold the place of real-life conceptual or physical objects with independent existence, a data model can be created by adding attributes to the Entity in order to fully describe it, and then by linking Entities together through relationships with real-world meaning. For example, if entities EMPLOYEE and PROJECT are created, they may be linked together by EMPLOYEE WORKS_ON PROJECT. In this way, a data model can be created which is a vital step towards developing a database.

Relational Model

The Relational Model, created by IBM researcher Ted Codd in 1970, is a data model rooted in set theory and first-order predicate logic that is widely used in developing modern database systems. Organized into tables, or more formally called “relations”, the two main methods of representation are through the tuple (row) and by attribute (column). Through the

use of unique primary keys, data requests, or queries, can be used to for extracting data from the database. Through the use of what are called foreign keys, relations can be linked in a similar fashion to the way the E-R model links entities together, allowing for a simplistic means of storing data, and yet a powerful means of organizing data for retrieval.

2.1.2 Comparison of Two Different Models

Differences and Similarities

The E-R and Relational Model both help to visualize and conceptualize what the actual database design will eventually be. Developers use the E-R model to visualize the main conceptual or physical entities of their database, as well as the attributes for each entity, and each relationship between other entities. The Relational Model is used to assist the developer to finalize their proposed database layout, as well as to create relations from proposed entities and link them together through foreign keys or relation tables, but the relational model could be difficult to create without some form of visual aid. So the E-R model helps with the initial design, but the two models are so similar that the E-R model can be easily converted to the Relational Model for physical database development using a DBMS, such as Oracle.

Advantages and Disadvantages

An advantage of using the E-R model over the relational model is the visualization aspect. In the E-R model, entities, attributes, and relationships can be easily visualized in a diagram using shapes that are linked together. In the relational model, relations are represented as tables, which aren't as visually appealing and are harder to work with when designing a database.

Another advantage the E-R Model has over the Relational Model is the fact that the E-R Model fully supports multi-valued and composite attributes while the Relational Model only supports single-valued and simple attributes. This can be problematic when real-world objects work better with multi-valued and composite attributes, such as when a car has more than one color. Conversion techniques can take care of this, but it is certainly an advantage over the Relational Model.

The main disadvantage of the E-R Model is that there does not exists a query language for that model. The Relational Model has the advantage of the SQL language, used for retrieving data, making the Relational Model a much more usable model.

2.2 Conversion from E-R Model to a Relational Database

2.2.1 Converting Entity Types to Relations

Strong Entity Types

When converting strong entity types to relations in the Relational Model, you must create a Relation that contains all of the simple attributes that are a part of the entity being converted. The relational model only supports simple attributes, and not composite or multi-valued attributes. One of the key attributes from the entity must also be selected as the primary key of the newly created relation.

Weak Entity Types

When converting weak entity types to relations in the Relational Model, a relation can be created using all of the simple attributes of the weak entity being converted. But because this is

a weak entity, which does not have a key of its own, a foreign key must be created from the primary key of the parent entity, so that the relations can be properly mapped for the relational model.

Simple and Composite Attributes

Only simple attributes are supported by the Relational Model. So for simple attributes in the E-R Model, simply include all simple attributes as simple attributes of their respective relation.

Composite attributes are not supported by the Relational Model. So for converting E-R Model composite attributes, you will break the composite attribute into a set of simple attributes that can easily be used in their respective relation. You can also create a separate relation for composite attributes instead of splitting them into simple attributes.

Single-valued and Multi-valued Attributes

Only single-valued attributes are supported by the Relational Model. So for single-valued attributes in the E-R Model, include them as simple, single-valued attributes in their respective relation.

Multi-valued attributes are not supported by the Relational Model. So for each multi-valued attribute, you must create a new relation that will hold the values for that multi-valued attribute. This new relation will have a foreign key that will relate this table to its parent relation

2.2.2 Converting Relationship Types to Relations

One to One Relationship

There are three options available for converting binary **One to One** relationship types. Given entities T and S who are in a One to One Relationship:

1. The first option is to include a foreign key in of the relations that references the other relation. Total participation of one entity in the other is very helpful in this case.

2. The second option is actually merge the two entities into one relation. The total participation from both entities is required for this to work.

3. The last option for binary One to One relationships is to create a cross-reference relation. Given entities T and S that are in a One to One relationship, create a relation from the relationship between T and S. The primary key of this relation R will be a combination of the primary keys of T and S. Any simple attributes from the relationship will be included as attributes of this new relation R.

One to Many Relationship

For converting binary **One to Many** Relationship types there are two options available. Given entities T and S who are in a One to Many Relationship:

1. If S is on the N-side of the relationship, include the primary key of T in the S relation as a foreign key. All simple attributes of the relationship are included as attributes of the S relation.

2. The last option for binary One to Many relationships is to create a cross-reference relation. Given entities T and S that are in a One to Many relationship, create a relation from the relationship between T and S. The primary key of this relation R will be a combination of the primary keys of T and S. Any simple attributes from the relationship will be included as attributes of this new relation R.

Many to Many Relationship

For converting binary **Many to Many** relationship types, there is only one option available. Given entities T and S who are in a Many to Many relationship:

1. The last option for binary Many to Many relationships is to create a cross-reference relation. Create a relation from the relationship between T and S. The primary key of this relation R will be a combination of the primary keys of T and S. Any simple attributes from the relationship will be included as attributes of this new relation R.

IS-A and HAS-A Relationships

IS A Relationships are specialization relationships. Specialization is the process of creating a set of subclasses from a certain superclass entity type. The resulting set of subclasses that forms a specialization is defined on the basis of some defining characteristic of the entities within the superclass.

This is the opposite of the abstraction process which we disregard the differences among several entity types and then identify their like features. Then, we can generalize them into a single superclass. This results in the original entity types being special subclasses. The term

generalization is used to refer to the process of defining a generalized entity type from the given entity types.

For converting an E-R specialization or generalization relationship to the Relational Model, there are four options:

1. Create relations for all superclasses and subclasses. Include a foreign key for each subclass relations to link them to their respective superclass. This works for any specialization.
2. Create relations only for the subclasses. This only works for specialization, and when the subclasses have total participation in the relationship.
3. Create a single relation for all super and subclasses, but the relation will contain a discrimination attribute. This attribute will indicate what subclass each tuple belongs to, if it indeed participates in the relationship.
4. Create a single relation with a set of type attributes that will indicate which subclass each tuple belongs to. This works for overlapping subclasses in specialization.

Recursive Relationship

A recursive relationship is when an entity is in a relationship with itself. To convert these relationships to the Relational Model, we can create a foreign key that will reference the primary key of its own relation. This way it can be in a relationship with itself.

For example, say in a company, there are Employees and Managers, but in the database there is only an Employee relation that contains both lower-level employees and upper-management employees. We can look at an employee tuple, and if the employee is a manager,

his primary key will show up as the foreign key in another employee tuple who is a lower-level employee. So it is in a recursive relationship.

N-ary Relationships

In an n-ary relation, you have more than 2 entities participating in a relationship. So for converting these relationships to the Relational Model, you must create a cross-reference table, which will contain an n number of foreign keys to relate each of the relations together. You would also include any and all simple and single-valued attributes.

Union Type Relationship

In the E-R Model, union type relationships, or categories, are when you create a subclass from the union of two or more entity types. These subclasses are otherwise known as a union type or a category.

For converting these relationships to the Relational Model, you must create a new relation which will only hold a surrogate key. A surrogate key is a new unique attribute created to uniquely identify members of the union relationship. This new key will then be added as a foreign key to each of the participating super-classes

2.2.3 Database Constraints

Databases must obey certain constraints that are either inherent in the relational schema itself, or are from business rules. Constraints are directly related to the database schema, or the method of organization. Constraints are related to how the data will be handled when deleting, inserting, or updating, and to ensure the integrity of the data in the database.

Domain Constraints

Domain constraints specify that all of the values within a tuple must be within the specified domain for each attribute, such as their datatype or enumerated datatype. For example, if the attribute's domain is an integer, than that attribute's value for all tuples must be an integer. Or if the domain is an enumerated datatype, the attribute's value for all tuples must exist within that enumerated set. When updating or inserting, the DBMS will not allow you to add a value to an attribute that does not exist in the domain of the attribute.

Key Constraints

Because the Relational Model is based on set theory, which by definition states that all elements in a set must be unique, there must be a way to distinguish tuples from each other for data retrieval. Through the use of primary and candidate keys, we can use these unique values to identify unique records in a database. However, a relation can have multiple unique keys, or candidate keys. One must be chosen as the primary key for this purpose. When updating or inserting, the DBMS will not allow you to create a key that is not unique.

NULL Constraints

Another constraint is when an attribute is not allowed to be null, such as for a first name or last name of a student. For updating or inserting, if the attribute is specified to be non-null, the DBMS will not allow you to insert null data.

Entity Integrity Constraints

The entity integrity constraint is the constraint where no primary key can be null. A primary key is used to identify unique records in a database, and each table is only allotted one primary key, so constraining the primary key to always being unique and not NULL is important. With joins, primary keys can also be used to relate tables together. For inserting or updating, the DBMS will not allow you to have a null primary key.

Referential Integrity Constraints

Referential Integrity Constraint is a constraint where a foreign key in a relation must reference an *existing* tuple from the relation it is referencing at any given time.

This constraint is important for database operations. For example, the delete operation can only be done if certain requirements are met. In our database, we could not delete an artist from the Artist relation without also deleting any and all albums and songs because this would violate the referential integrity constraint. For inserting or updating data in the database, you can't make foreign keys that reference tuples in other relations that do not currently exist. This would also violate the referential integrity constraint.

Check Constraint

Check constraints are constraints that are defined upon creating a database table using SQL. Check constraints are used to ensure the integrity of data that is being updated or inserted by making sure that a specific condition is met. If the value being inserted is not null, then the

check constraint will evaluate to TRUE or FALSE. If the value being inserted is null, then the check constraint will evaluate to UNKNOWN, but it will not violate the constraint.

An example of a check constraint would be to say that the price of an item is greater than $0.00. So that when data is entered or updated, no item in the database can have a price of less than or equal to $0.00. So, when inserting or updating data, if the data evaluates the check constraint to false, the DBMS will not allow you to insert that data.

Business Rules

Business rules are policies that businesses follow that must also apply to the data in their databases. For example, if a company's policy states that an accountant can't have a relative who works as a cashier, this rule also needs to be enforced in the database through either triggers or the application software.

These rules essentially maintain proper relationships between entities as well as ensure the integrity of the data. For example, for our database we have an entity called Artist. Every Artist has Albums, but no Album can exist without an Artist. Below are several more examples specific to our database.

1. No Member can exist in the database without being a part of at least one Artist.
2. No Artist can exist in the database without having a Contract in the database.
3. No Song can exist in the database without having an Album in the database.
4. No Album can exist in the database without having an Artist in the database.

5. No Transaction can exist without a Buyer, and no Transaction can exist without having at least one Album being sold in that Transaction.

2.3 Convert Record Company E-R Model to a Relational Database

2.3.1 Relation Schema

Using all previously mentioned conversion techniques from E-R Model to Relational Database, we are able to convert our entities to relations. Each of our Entities in our E-R Model we created were strong entities, so we created relations for each of them with all simple attributes.

All of the relationships we created between entities were either 1:N, N:1, or N:M. So for all of our 1:N or N:1 relationships, we used foreign keys for the conversion to the relational model. For the two N:M relationships we have, we used the method of creating a relation, where each tuple is an instance of a relationship between two relations.

The following tables are representations of our relation schemas for our relational database design.

Keys	PRIMARY	FOREIGN

Contract

For converting the strong entity Contract, we created a relation that contains all of the simple attributes from the Contract entity. The primary key is contract_ID.

Attribute	Domain	Description
Contract_ID	String	Unique contract ID
Album_terms	Int	Number of albums under contract
Start_date	Date	Date contract started
End_date	Date	Date contract ends

Artist

For converting the strong entity Artist, we created a relation that contains all of the simple attributes from the Artist entity. For the **Contract Hires Artist** relationship, which is **1:1**, we've added a foreign key, contract_ID, to the Contract relation. The primary key is artist_ID.

Attribute	Domain	Description
Artist_ID	String	Unique artist ID
Contract_ID	String	Contract number
Artist_name	String	Name of artist
Genre	String	Type of music

Composed_Of

Composed_Of was a relationship between the two strong entities Artist and Member. It is a **N:N** relationship, so for converting, we've created a relation where each tuple is an instance of that relationship. The primary key of this relation is the combination of the primary keys from Artist and Member.

Attribute	Domain	Description
artist_ID	String	Unique artist ID
Ssn	String	Social Security Number of artist member

Member

For converting the strong entity Member, we created a relation that contains all of the simple attributes from the Member entity. The primary key is ssn.

Attribute	Domain	Description
Ssn	String	Social security number
Fname	String	First name
Lname	String	Last name
Phone	String	Phone number
Address	String	Street address
City	String	Name of city
State	String	U.S. state
Zip	String	Zip code
instrument	String	Instrument played
Start_date	Date	Date contract started
End_date	Date	Date contract ends

Album

For converting the strong entity Album, we created a relation that contains all of the simple attributes from the Album entity. For the **Artist Records Album** relationship, which is **1:N,** we've added a foreign key, artist_ID, to the Album relation. For the **Album Recorded At Studio** relationship, which is **N:1**, we've added a foreign key, studio_ID, to the Album relation. The primary key is album_ID.

Attribute	Domain	Description
Album_ID	String	Unique album ID
Album_name	String	Album name
Date_released	Date	Date album released
Unit_price	Float	Selling price per album
artist_ID	String	Unique artist ID
Studio_ID	String	Unique studio ID
Start_date	Date	Date work on album began
End_date	Date	Date work on album ended
Hrs_worked	Int	Number of hours worked on album

Sold_Through

Sold_Through was a relationship between the two strong entities Album and Transaction. It is a **N:N** relationship, so for converting, we've created a relation where each tuple is an instance of that relationship. The primary key of this relation is the combination of the primary keys from Artist and Member.

Attribute	Domain	Description
Album_ID	String	Unique ID for album
Transaction_ID	String	Unique ID for transaction
Batch_units	Int	Amount of each album in transaction

Transaction

For converting the strong entity Transaction, we created a relation that contains all of the simple attributes from the Transaction entity. For the **Transaction Purchased By Buyer** relationship, which is **N:1**, we've added a foreign key, buyer_ID, to the Buyer relation. The primary key is transaction_ID.

Attribute	Domain	Description
Transaction_ID	String	Unique transaction ID
Buyer_ID	String	Unique buyer ID
Date	Date	Date of purchase

Buyer

For converting the strong entity Buyer, we created a relation that contains all of the simple attributes from the Buyer entity. The primary key is buyer_id.

Attribute	Domain	Description
Buyer_ID	String	Unique buyer ID
Buyer_name	String	Name of buyer
Phone	String	Phone number
Address	String	Street address
City	String	City
State	String	U.S. state
Zip	String	Zip code

Song

For converting the strong entity Song, we created a relation that contains all of the simple attributes from the Song entity. For the **Album Composed Of Song** relationship, which is **1:N**, we've added a foreign key, album_ID, to the Contract relation to link songs to their album. For the **Song Written By Writer** relationship, which is **N:1**, we've added a foreign key, writer_ID, to the Contract relation to link songs to their Writer. The primary key is artist_ID.

Attribute	Domain	Description
Song_ID	String	Unique Song ID
Song_name	String	Song name
Track_no	Int	Track number on album
Song_length	Time	Length of song
Album_ID	String	Unique album ID
Writer_ID	String	Unique writer ID

Studio

For converting the strong entity Studio, we created a relation that contains all of the simple attributes from the Studio entity. The primary key is studio_ID.

Attribute	Domain	Description
Studio_ID	String	Unique Studio ID
Studio_name	String	Name of studio
phone	String	Phone Number
Address	String	Street address
City	String	Name of city
State	String	U.S. state
Zip	String	Zip code
Open_date	Date	Date studio opened
Close_date	Date	Date studio closed
Hrly_cost	Float	Cost per hour to rent studio

Writer

For converting the strong entity Writer, we created a relation that contains all of the simple attributes from the Writer entity. The primary key is writer_ID.

Attribute	Domain	Description
Writer_ID	String	Unique ID for writer
fname	String	First name
lname	String	Last name
Ssn	String	Social Security Number
phone	String	Phone Number
Address	String	Street address
City	String	Name of city
State	String	U.S. state
Zip	String	Zip code
Start_date	Date	Date contract started
End_date	Date	Date contract ends

Relational Model

Keys	PRIMARY	FOREIGN

Contract

Contract_ID	album_terms	start_date	end_date

Artist

artist_ID	contract_ID	artist_name	genre

Compose_Of

Artist_ID	SSN

Member

SSN	fName	lName	Phone	Address	City	State	Zip	Instrument	Start_date	End_date

Album

Album_ID	Album_Name	Date_Released	Unit_Price	Artist_ID	Studio_ID	Start_date	End_date	Hours_work

Sold_Through

Transaction_ID	Album_ID	BatchUnits

Transaction

Transaction_ID	Buyer_ID	Date

Buyer

Buyer_ID	Buyer_Name	phone	address	city	state	zip

Song

Song_ID	Track_Number	Song_Name	Song_Length	Album_ID	Writer_ID

Studio

Studio_ID	Studio_Name	Phone	Address	City	State	Zip	Open_date	Close_date	HourlyCost

Writer

Writer_ID	fName	lName	SSN	Phone	Address	City	State	Zip	Start_date	End_date

2.3.2 Sample Data of Relation

Contract

ContractID	AlbumTerms	StartDate	EndDate
C17813	1	4/22/2010	4/1/2011
C44086	3	5/29/2010	5/1/2013
C11328	5	4/14/2011	NULL
C4752	3	12/06/14	NULL
C6789	2	9/20/2010	9/1/2012
C55390	5	10/22/13	NULL
C85862	6	10/17/12	NULL
C34728	2	4/8/2014	NULL
C93471	4	11/22/12	NULL
C50643	2	3/8/2010	3/1/2012
C19760	1	11/11/14	NULL
C53622	1	9/4/2011	9/1/2012
C42579	2	6/18/2012	6/1/2014
C96710	4	4/5/2014	NULL
C67044	5	8/23/2010	8/1/2015

Artist

Artist_ID	ContractID	Artist_Name	Genre
A17813	C17813	Solomon	Folk
A44086	C44086	Bell	Rock
A11328	C11328	Hilda	Rock
A4752	C4752	Baker	Metal
A6789	C6789	Jamalia	Classic
A55390	C55390	Linus	Folk
A85862	C85862	Steven	Rock
A34728	C34728	Colette	Alternative
A93471	C93471	Adena	Folk
A50643	C50643	Angela	Rock
A19760	C19760	Ralph	SoftRock
A53622	C53622	Eliana	SoftRock
A42579	C42579	Chanda	Alternative
A96710	C96710	Yardley	Folk
A67044	C67044	Solomon	Rock

Composed Of

Artist_ID	SSN
A17813	115-80-7312
A17813	898-63-6127
A17813	095-12-8618
A17813	718-40-2914
A17813	094-10-3269
A17813	390-79-6801
A17813	799-51-6609
A44086	466-69-1455
A44086	913-28-7463
A44086	316-63-2012
A44086	706-38-8832
A11328	230-40-5066
A11328	237-09-5808
A11328	030-52-9568
A4752	350-82-6131
A4752	966-72-8262
A4752	174-63-5988
A4752	555-69-8350
A4752	522-64-6239
A4752	033-39-0750
A4752	752-64-0200
A6789	060-96-8225
A6789	704-11-3574
A6789	380-38-4098
A6789	936-50-7878
A6789	080-69-9386
A55390	445-08-9490
A55390	694-77-5842
A55390	359-68-4742
A55390	022-62-8780
A55390	727-70-6860
A85862	297-66-5184

	A85862	727-43-1600
	A85862	835-47-1116
	A34728	681-99-1076
	A34728	326-33-7324
	A34728	794-07-1975
	A34728	065-29-1852
	A93471	238-44-2093
	A93471	647-75-9371
	A50643	493-54-0743
	A50643	315-18-5134
	A19760	934-66-9708
	A53622	103-18-0494
	A53622	689-61-9138
	A53622	519-05-5209
	A53622	130-87-1020
	A53622	648-37-7849
	A42579	018-38-7601
	A42579	647-90-6445
	A42579	737-34-7478
	A96710	043-98-9774
	A96710	276-39-2123
	A67044	800-96-0314
	A67044	055-92-3612
	A67044	372-27-3921
	A67044	703-99-8621
	A67044	815-09-6438
	A67044	249-64-8859
	A67044	539-63-8320
	A67044	985-14-8004
	A67044	355-42-2240
	A67044	196-45-7896

Member

SSN	fName	lName	Phone	Address	City	State	Zip	Instrument	Start_date	End_date
115-80-7312	Addison	Dane	1-771-325-3488	P.O. Box 140, 5851 Per Avenue	Indianapolis	IN	46201	Bass	11/08/2010	01/20/2014
898-63-6127	Harlan	Dexter	1-921-624-5844	P.O. Box 447, 3845 Erat, Avenue	West Hopewell Junction	NY	12533	Keyboard	05/10/2012	10/26/2015
095-12-8618	Hiroko	Martina	1-780-369-8946	980-5300 Dictum Av.	Glasgow	KY	42141	Keyboard	02/19/2012	04/20/2013
718-40-2914	September	Ila	1-535-182-5258	Ap #937-3791 Metus St.	Mount Prospect	IL	60056	Guitar	03/03/2009	10/14/2014
094-10-3269	Uta	Reed	1-239-176-1619	P.O. Box 527, 7610 Penatibus Rd.	Munster	IN	46321	Piano	07/08/2011	12/19/2013
390-79-6801	Joan	Ferris	1-510-766-2458	762-2790 Condimentum. St.	Forest Hills	NY	11375	Vocals	01/09/2013	05/26/2013
799-51-6609	Alden	Odysseus	1-392-940-5101	7839 Cras Ave	Palm Beach Gardens	FL	33410	Bass	03/01/2011	09/01/2013
114-81-6401	Timon	Calista	1-183-789-9028	Ap #732-3331 Cursus Avenue	West Orange	NJ	7052	Vocals	06/09/2010	08/05/2013
466-69-1455	Lara	Beau	1-299-943-6744	6780 Sapien, Street	Brockton	MA	2301	Vocals	01/11/2011	10/19/2014
913-28-7463	Francesca	Bree	1-105-417-8578	564-7305 Tellus St.	Orland Park	IL	60462	Bass	08/04/2009	04/11/2013
316-63-2012	Isaac	Leila	1-535-133-9149	P.O. Box 350, 3397 Nulla Rd.	East Winona	MN	55987	Drums	01/01/2011	07/05/2015
706-38-8832	Cameron	Tatum	1-172-969-4536	9623 Orci, Road	Chardon	OH	44024	Keyboard	03/27/2010	07/18/2014
230-40-5066	Thaddeus	Josiah	1-902-456-7204	Ap #312-5175 Magna, Rd.	Yorktown	VA	23693	Bass	10/15/2012	06/29/2013
237-09-5808	Thaddeus	Joan	1-711-341-2441	450-6815 Aliquet St.	Faribault	MN	55021	Drums	02/26/2013	11/29/2015
030-52-9568	Daryl	Jillian	1-262-876-8431	Ap #524-446 Velit. Rd.	Loxahatchee	FL	33470	Bass	04/12/2010	10/29/2014
350-82-6131	Heather	Timon	1-334-774-2003	P.O. Box 843, 4176 Non Ave	Indianapolis	IN	46201	Drums	11/08/2010	01/20/2014
966-72-8262	Berk	Jelani	1-790-197-8783	Ap #122-6362 Elementum Av.	Maryborough	IN	46201	Drums	11/08/2010	01/20/2014
174-63-5988	Shellie	Kessie	1-419-862-7899	Ap #463-4485 Amet St.	Saint-Honor	NY	12533	Keyboard	05/10/2012	10/26/2015
555-69-8350	Harding	Fay	1-676-512-8361	504-1999 Justo Ave	Bazel	KY	42141	Bass	02/19/2012	04/20/2013
522-64-6239	Zenia	Yvette	1-437-226-8794	796-9703 Arcu. St.	Brunn am Gebirge	IL	60056	Piano	03/03/2009	10/14/2014
033-39-0750	Karly	Ruby	1-125-105-9052	Ap #390-9509 Sed, Street	Watford	IN	46321	Keyboard	07/08/2011	12/19/2013
752-64-0200	Tucker	Kenyon	1-566-133-7495	925-6822 Elit Rd.	Harelbeke	NY	11375	Drums	01/09/2013	05/26/2013
060-96-8225	Adam	Tanisha	1-433-701-5336	P.O. Box 282, 1071 Et Ave	Cleveland	FL	33410	Drums	03/01/2011	09/01/2013
704-11-3574	Bradley	Reagan	1-225-367-0851	P.O. Box 587, 9953 In Avenue	Bomal	NJ	7052	Guitar	06/09/2010	08/05/2013
380-38-4098	Kendall	Patricia	1-273-838-1323	7443 Gravida Rd.	Taunusstein	MA	2301	Bass	01/11/2011	10/19/2014
936-50-7878	Rahim	Renee	1-240-948-5291	247 Gravida. Av.	Swadlincote	IL	60462	Guitar	08/04/2009	04/11/2013
080-69-9386	Shelley	Stella	1-152-175-2273	614 Tincidunt Street	Sint-Stevens-Woluwe	MN	55987	Vocals	01/01/2011	07/05/2015
445-08-9490	Walter	Ingrid	1-748-419-0250	4674 Enim. Rd.	Reading	OH	44024	Keyboard	03/27/2010	07/18/2014
694-77-5842	Sierra	Jin	1-624-476-8346	P.O. Box 393, 976 Erat, St.	Campbelltown	VA	23693	Bass	10/15/2012	06/29/2013
359-68-4742	Abraham	Quinlan	1-504-800-3789	106 Non, St.	Dhuy	MN	55021	Drums	02/26/2013	11/29/2015
022-62-8780	Lucas	Alec	1-139-337-8115	P.O. Box 105, 139 Imperdiet St.	Zonhoven	FL	33470	Piano	04/12/2010	10/29/2014
727-70-6860	Yoko	Jeanette	1-678-846-7744	P.O. Box 985, 1703 Commodo Avenue	Verrès	IN	46201	Guitar	11/08/2010	01/20/2014
297-66-5184	Heidi	Ainsley	1-104-862-3923	P.O. Box 514, 7993 Laoreet Road	Hattersheim am Main	IN	46201	Guitar	11/08/2010	01/20/2014
727-43-1600	Kadeem	Ali	1-962-154-1797	748-7299 Scelerisque Ave	Cottbus	NY	12533	Vocals	05/10/2012	10/26/2015
835-47-1116	Lacota	Ray	1-377-697-6552	1213 Tellus Avenue	Christchurch	KY	42141	Keyboard	02/19/2012	04/20/2013
681-99-1076	Unity	Fiona	1-310-642-8505	9362 Nullam Rd.	Senneville	IL	60056	Drums	03/03/2009	10/14/2014
326-33-7324	Martha	Brock	1-696-994-0134	5945 Dolor Ave	Barrie	IN	46321	Piano	07/08/2011	12/19/2013
794-07-1975	Willow	Honorato	1-751-866-8220	396-1218 Donec Rd.	Premeno	NY	11375	Piano	01/09/2013	05/26/2013
065-29-1852	Ramona	Halee	1-973-943-3237	Ap #577-904 Mauris Road	West Valley City	FL	33410	Drums	03/01/2011	09/01/2013
238-44-2093	Charles	Tara	1-908-870-1080	Ap #522-7038 Vivamus St.	Assebroek	NJ	7052	Piano	06/09/2010	08/05/2013
647-75-9371	Yoshi	Anne	1-952-388-5101	P.O. Box 722, 9775 Placerat Avenue	Asso	MA	2301	Piano	01/11/2011	10/19/2014
493-54-0743	Halla	Casey	1-852-575-0692	Ap #850-5478 Facilisis Road	Batiscan	IL	60462	Bass	08/04/2009	04/11/2013
315-18-5134	Fuller	Brock	1-929-124-9283	P.O. Box 281, 4648 Mauris Rd.	Kingston-on-Thames	MN	55987	Piano	01/01/2011	07/05/2015
934-66-9708	Damian	Violet	1-783-953-5683	957-4329 Tortor. Street	Rathenow	OH	44024	Drums	03/27/2010	07/18/2014
103-18-0494	Paki	Ezekiel	1-398-770-8648	P.O. Box 312, 3911 Fermentum St.	Berlin	VA	23693	Vocals	10/15/2012	06/29/2013
689-61-9138	Imogene	Darius	1-704-388-0188	683-8741 Nonummy St.	St. Albert	MN	55021	Piano	02/26/2013	11/29/2015
519-05-5209	Melvin	Joshua	1-187-981-3757	Ap #166-2136 Facilisis Rd.	Tramonti di Sopra	FL	33470	Drums	04/12/2010	10/29/2014
130-87-1020	Malachi	Nyssa	1-920-537-5019	855-7728 Risus Av.	Biggleswade	IN	46201	Guitar	11/08/2010	01/20/2014
648-37-7849	Noel	Holly	1-894-326-3460	681-5092 Nunc Avenue	Makurdi	IN	46201	Drums	11/08/2010	01/20/2014
018-38-7601	Martina	Blythe	1-338-546-0577	439-4198 Nisl. Road	Biez	NY	12533	Vocals	05/10/2012	10/26/2015
647-90-6445	Judith	Kirby	1-883-514-4306	4565 Ante. Av.	Dsseldorf	KY	42141	Vocals	02/19/2012	04/20/2013
737-34-7478	Rhona	Rashad	1-444-623-8960	P.O. Box 736, 8952 Suscipit Ave	Juneau	IL	60056	Keyboard	03/03/2009	10/14/2014
043-98-9774	Jamal	Jolene	1-975-148-6231	239-6604 Nunc Av.	Port Hope	IN	46321	Vocals	07/08/2011	12/19/2013
276-39-2123	Jaime	Laurel	1-733-265-1472	Ap #835-5410 Libero Road	Arrone	NY	11375	Keyboard	01/09/2013	05/26/2013
800-96-0314	Libby	Jeremy	1-537-410-7125	P.O. Box 350, 1722 Libero Street	Poitiers	FL	33410	Vocals	03/01/2011	09/01/2013
055-92-3612	Audra	Fiona	1-872-560-1899	5953 Vel, St.	Cimitile	NJ	7052	Drums	06/09/2010	08/05/2013
372-27-3921	Drake	Maggy	1-733-374-7345	754-5259 Orci, Rd.	Montebello	MA	2301	Bass	01/11/2011	10/19/2014
703-99-8621	Ulla	Isabelle	1-519-893-1634	960 Eget St.	Wolverhampton	IL	60462	Piano	08/04/2009	04/11/2013
815-09-6438	Sonya	Cheyenne	1-178-367-4775	8433 Adipiscing Avenue	Paglieta	MN	55987	Drums	01/01/2011	07/05/2015

Album

Album_ID	Album_Name	Date_released	Unit_price	Artist_ID	StudioID	Start_date	End_date	Hours_Worked
AL3626	microscope	8-Feb-11	9.99	A17813	S1262	1-Jan-11	3-Feb-11	200
AL4620	midget	7-Feb-13	12.99	A44086	S8241	17-Apr-12	2-Feb-13	200
AL2151	credit card	18-Jul-11	3.99	A11328	S5458	24-Apr-11	11-Jul-11	300
AL7741	system	15-Apr-15	9.99	A4752	S5894	15-Dec-14	15-Mar-15	1000
AL2381	observation	26-Aug-11	7.99	A6789	S6310	15-Feb-11	20-Aug-11	200
AL5377	fork	15-Jan-15	24.99	A55390	S4552	14-Sep-14	26-Dec-14	300
AL5780	t-shirt	24-Jan-13	2.99	A85862	S1384	18-Nov-12	14-Jan-13	500
AL1803	surgeon	15-Dec-14	1.99	A34728	S1099	17-Nov-14	1-Dec-14	700
AL5392	wisdom	25-May-14	7.99	A93471	S1017	13-Mar-13	18-May-14	800
AL7282	log	16-Aug-11	8.99	A50643	S2383	7-Jun-11	8-Aug-11	70
AL725	laser	24-Dec-15	9.99	A19760	S3370	10-Oct-15	12-Dec-15	59
AL8817	tap	17-Jan-12	9.99	A53622	S4099	25-Sep-11	1-Jan-12	300
AL9098	critic	24-Dec-13	12.99	A42579	S6369	8-Aug-13	12-Dec-13	250
AL3881	throne	19-Jun-15	14.99	A96710	S3386	2-Feb-15	6-Jun-15	50000
AL4688	hazard	19-Jan-13	16.99	A67044	S126	5-Oct-12	1-Jan-13	300

Sold Through

TransactionID	AlbumID	Batch Units
T17813	AL3626	200000
T44086	AL4620	250000
T11328	AL2151	150000
T4752	AL7741	10000
T6789	AL2381	500000
T55390	AL5377	25000
T85862	AL5780	1000
T34728	AL1803	100
T93471	AL5392	55000
T50643	AL7282	230000
T19760	AL725	100000
T53622	AL8817	1500
T42579	AL9098	2000
T96710	AL3881	250
T67044	AL4688	100000

Transaction

TransactionID	Buyer_ID	Date
T17813	B00001	10/05/15
T44086	B00005	06/26/14
T11328	B00003	07/25/14
T4752	B00004	09/09/14
T6789	B00005	09/23/15
T55390	B00006	03/21/15
T85862	B00004	11/21/15
T34728	B00005	09/18/14
T93471	B00006	04/10/15
T50643	B00005	08/05/14
T19760	B00005	04/01/15
T53622	B00001	01/21/14
T42579	B00004	11/30/13
T96710	B00006	09/19/15
T67044	B00001	10/05/15

Buyer

Buyer_ID	Buyer_name	phone	address	city	state	zip
B00001	FYE	1-410-941-0560	7910 Dolor. Road	Tacoma	WA	25345
B00002	Kmart	1-379-875-2434	8617 Adipiscing Rd.	West Valley City	UT	81502
B00003	Wal-Mart	1-932-271-7830	6554 Urna St.	Boise	ID	41529
B00004	Costco	1-249-978-7321	4592 Aliquet. Road	Aurora	CO	49142
B00005	Target	1-184-341-5591	1136 Id Street	Chicago	IL	34218
B00006	AmoebaRecords	1-481-394-3965	9627 Amet St.	Jonesboro	AR	72604
B00007	The Warehouse	1-463-400-3980	4382 Augue St.	Madison	WI	37518
B00008	Suncoast	1-421-736-7118	6594 Urna Avenue	Boise	ID	40083
B00009	Shop Here Please	1-121-935-1648	5537 Donec Street	Springfield	IL	91158
B00010	Dollar General	1-460-442-9430	4400 Aliquet Av.	Kailua	HI	32090
B00011	One Stop Shop	1-464-348-8587	5161 Aenean Rd.	Billings	MT	94754
B00012	S-Mart	1-656-784-3345	8367 Habitant St.	San Diego	CA	92003
B00013	Quick Stop Groceries	1-618-594-7681	3545 Quisque Street	Saint Paul	MN	69624
B00014	Sears	1-457-567-9254	2714 Nunc, Street	Davenport	IA	67485
B00015	Records R Us	1-613-336-1368	8167 Nibh. Road	Baltimore	MD	76135

Song

Song_ID	Track_Number	Song_name	Song_length	AlbumID	WriterID
SO8882	1	Dude Dude	8:84	AL3626	W1262
SO7345	2	Shut Clock	6:58	AL3626	W1262
SO3375	3	Unkempt Bone	9:13	AL3626	W1262
SO8701	4	Aquatic Downtown	2:27	AL3626	W1262
SO9804	5	Premium Sign	3:76	AL3626	W1262
SO0740	6	Old-Fashioned Bead	4:55	AL3626	W1262
SO8528	7	Neat Pigs	3:99	AL3626	W1262
SO4682	8	Few Crack	4:85	AL3626	W1262
SO8488	1	Flashy Ocean	4:77	AL3626	W8241
SO1533	2	Faithful Whip	0:76	AL3626	W8241
SO3099	3	Confused Screw	6:10	AL4620	W8241
SO1814	4	Holy Hell	1:98	AL4620	W8241
SO6409	1	Tent Treatment	8:53	AL4620	W8241
SO3550	2	Rainstorm Apples	6:02	AL4620	W8241
SO1839	3	Birth Soap	2:06	AL4620	W8241
SO2571	4	Minute Popcorn	5:35	AL4620	W8241
SO6969	5	Jewel Note	3:57	AL4620	W8241
SO7506	6	Change Question	6:68	AL4620	W5458
SO5478	6	Letter Circle	5:64	AL4620	W5458
SO5772	7	Base Fork	3:48	AL4620	W5458
SO1843	8	Good-Bye Meeting	1:32	AL4620	W5458
SO4135	9	Fight Expansion	6:61	AL4620	W5458
SO1915	1	Light syrup	5:21	AL4620	W5458
SO2408	1	Object Book	7:81	AL4620	W5458
SO0776	2	Marry Touch	6:10	AL4620	W5458
SO8410	3	Justify Chin	8:68	AL2151	W5894
SO8202	4	Bounce Stick	5:32	AL2151	W5894
SO2521	5	Cheat Air	5:52	AL2151	W5894
SO1231	6	Breed Uncle	1:68	AL2151	W5894

	SO8531	7	Mislead Beggar	9:81	AL2151	W5894
	SO3251	8	Analyze Secretary	6:07	AL2151	W5894
	SO3942	9	Exercise View	0:25	AL2151	W5894
	SO1421	10	Intend Approval	3:16	AL2151	W5894
	SO9412	11	That’s too much	6:62	AL7741	W5894
	SO4618	12	Talented Turkey	1:33	AL7741	W5894
	SO0985	13	Arrogant Apples	5:38	AL7741	W6310
	SO6835	14	Young Yard	4:93	AL7741	W6310
	SO8417	15	Simple Scent	9:76	AL7741	W6310
	SO9423	16	Ten Town	7:74	AL7741	W6310
	SO4161	17	Comfortable Cactus	6:49	AL7741	W6310
	SO3935	1	Cooing Crayon	6:69	AL7741	W6310
	SO8108	2	Pricey Party	8:73	AL7741	W6310
	SO0478	3	Poised Plastic	9:63	AL7741	W6310
	SO5138	4	Silly Scale	9:45	AL7741	W4552
	SO3275	5	Ketchup and A1	3:11	AL7741	W4552
	SO2116	6	Wax War	6:14	AL7741	W4552
	SO3114	7	Art Ants	8:46	AL7741	W4552
	SO6665	8	Mind Minister	7:05	AL7741	W4552
	SO6383	9	Mother Mice	0:63	AL7741	W4552
	SO4064	1	Texture Tramp	7:55	AL7741	W4552
	SO0519	2	Bread Bean	1:86	AL2381	W4552
	SO6505	1	Ship Space	5:97	AL2381	W4552
	SO8475	2	Prose Produce	6:98	AL2381	W4552
	SO8551	1	Plough Pump	2:11	AL2381	W4552
	SO1457	1	Skate Spot	1:04	AL5377	W1384
	SO6451	2	Don’t touch me	4:69	AL5377	W1384
	SO9401	3	Ascertain Army	2:36	AL5377	W1384
	SO0623	4	Jail Join	7:62	AL5377	W1384
	SO0892	5	Branch Bath	4:44	AL5377	W1384
	SO3588	6	Sense Slope	7:99	AL5377	W1384
	SO6478	7	Search Stop	2:39	AL1803	W1384

	SO4088	8	Smile Stew	0:28	AL1803	W1384
	SO4779	9	Bump Beam	6:63	AL1803	W1384
	SO0859	10	Receive Reward	1:77	AL1803	W1384
	SO3772	11	Sell Slope	1:37	AL1803	W1384
	SO7864	12	Misunderstand Meal	1:89	AL1803	W1384
	SO7597	13	Cough Range	5:51	AL1803	W1099
	SO0073	14	Zipper Camp	8:12	AL1803	W1099
	SO5223	1	Heat Vegetable	2:59	AL1803	W1099
	SO1528	2	Clouds Disgust	1:34	AL5392	W1099
	SO6312	3	Touch me	3:83	AL5392	W1099
	SO7996	4	Attack Uncle	9:81	AL5392	W1099
	SO3087	5	Determine Dress	8:50	AL5392	W1099
	SO6947	6	Recommend Direction	6:30	AL5392	W1099
	SO3981	7	Scream Birthday	7:59	AL5392	W1099
	SO9372	8	Fax Secretary	5:41	AL5392	W1099
	SO2585	9	Process Road	0:58	AL5392	W1099
	SO3545	1	Squash Pickle	2:07	AL5392	W1099
	SO7466	2	Diagnose Bomb	8:03	AL5392	W1017
	SO7763	3	Experiment Authority	4:58	AL7282	W1017
	SO8381	4	Finalize Plantation	0:24	AL7282	W1017
	SO3969	5	Help me	9:58	AL7282	W1017
	SO1657	6	Elite Edge	5:19	AL7282	W2383
	SO5914	7	Pushy Pollution	8:23	AL7282	W2383
	SO9667	8	Disillusioned Donkey	8:62	AL7282	W2383
	SO1319	9	Uppity Underwear	6:27	AL7282	W2383
	SO2713	1	Ad Aunt	8:39	AL725	W2383
	SO2932	1	Tacit Tooth	7:23	AL725	W2383
	SO2873	2	Married Morning	0:14	AL8817	W3370
	SO3144	3	Fallacious Fuel	0:86	AL8817	W3370
	SO7803	4	Massive Mass	1:25	AL9098	W3370
	SO1986	5	Careless Chair	3:54	AL9098	W3370
	SO7525	6	It was the dog	9:44	AL9098	W3370

SO6298	7	Smoke Shame	1:51	AL3881	W3370
SO1895	8	Crate Chance	3:48	AL3881	W3370
SO9648	9	Square Shop	2:47	AL3881	W4099
SO7926	10	Juice Jewel	9:38	AL4688	W6369
SO4249	11	Blow Baseball	5:85	AL4688	W3386
SO1891	12	Business Beam	6:68	AL4688	W126
SO4994	13	Stomach Shirt	5:92	AL4688	W1262

Writer

fName	lName	SSN	Phone	Address	City	State	Zip	Start_date	End_date
Rhea	Amaya	219-63-9062	1-786-379-4906	681 Route 32	Indianapolis	IN	46201	11/08/2010	01/20/2014
Neil	Christopher	217-80-9469	1-666-928-0290	747 2nd Street	West Hopewell Junction	NY	12533	05/10/2012	10/26/2015
Rafael	Henry	235-86-3894	1-553-866-5682	621 Main Street	Glasgow	KY	42141	02/19/2012	04/20/2013
Sarah	Jorden	476-35-0140	1-249-129-3089	458 Inverness Drive	Mount Prospect	IL	60056	03/03/2009	10/14/2014
Angela	Leilani	439-86-1590	1-664-565-6363	157 Williams Street	Munster	IN	46321	07/08/2011	12/19/2013
Idola	Aline	329-74-0439	1-223-723-4437	761 Ann Street	Forest Hills	NY	11375	01/09/2013	05/26/2013
Cody	Yetta	166-20-0007	1-145-811-5254	275 Beech Street	Palm Beach Gardens	FL	33410	03/01/2011	09/01/2013
Jermaine	Wayne	578-26-9691	1-318-413-1114	961 Route 44	West Orange	NJ	7052	06/09/2010	08/05/2013
Jocelyn	Patricia	478-32-1350	1-238-343-1057	657 Locust Street	Brockton	MA	2301	01/11/2011	10/19/2014
Fay	Cleo	680-22-5134	1-572-953-0248	957 Penn Street	Orland Park	IL	60462	08/04/2009	04/11/2013
Sharon	Hamilton	508-42-5295	1-749-852-3465	209 Main Street	East Winona	MN	55987	01/01/2011	07/05/2015
Fletcher	Kristen	358-64-3701	1-574-253-8981	613 Victoria Court	Chardon	OH	44024	03/27/2010	07/18/2014
Carolyn	Karen	530-53-0524	1-428-413-4720	570 Windsor Drive	Yorktown	VA	23693	10/15/2012	06/29/2013
Glenna	Fatima	036-07-7409	1-243-753-0311	978 Mill Street	Faribault	MN	55021	02/26/2013	11/29/2015
Rhea	Amaya	576-61-7175	1-786-379-4906	186 Glenwood Drive	Loxahatchee	FL	33470	04/12/2010	10/29/2014
Neil	Christopher	270-34-9838	1-666-928-0290	681 Route 32	Indianapolis	IN	46201	11/08/2010	01/20/2014

Studio

Studio_ID	Studio_Nam	Phone	Address	City	State	Zip	Start_date	End_date	HourlyCost
S1262	Ante Maecenas Mi Limited	1-786-379-4906	681 Route 32	Indianapolis	IN	46201	11/08/2010	01/20/2014	$116.74
S8241	Pede Malesuada Industries	1-666-928-0290	747 2nd Street	West Hopewell Junction	NY	12533	05/10/2012	10/26/2015	$86.88
S5458	Duis Institute	1-553-866-5682	621 Main Street	Glasgow	KY	42141	02/19/2012	04/20/2013	$110.44
S5894	Nec Euismod LLC	1-249-129-3089	458 Inverness Drive	Mount Prospect	IL	60056	03/03/2009	10/14/2014	$98.69
S6310	Imperdiet Ornare In Associates	1-664-565-6363	157 Williams Street	Munster	IN	46321	07/08/2011	12/19/2013	$114.62
S4552	Magna Institute	1-223-723-4437	761 Ann Street	Forest Hills	NY	11375	01/09/2013	05/26/2013	$136.50
S1384	Etiam Ligula Corp.	1-145-811-5254	275 Beech Street	Palm Beach Gardens	FL	33410	03/01/2011	09/01/2013	$135.69
S1099	Est Mauris Rhoncus Corp.	1-318-413-1114	961 Route 44	West Orange	NJ	7052	06/09/2010	08/05/2013	$154.24
S1017	Aenean Massa Associates	1-238-343-1057	657 Locust Street	Brockton	MA	2301	01/11/2011	10/19/2014	$197.25
S2383	Est Mollis Non LLP	1-572-953-0248	957 Penn Street	Orland Park	IL	60462	08/04/2009	04/11/2013	$135.08
S3370	Eu Accumsan Sed Inc.	1-749-852-3465	209 Main Street	East Winona	MN	55987	01/01/2011	07/05/2015	$171.32
S4099	Ultricies Adipiscing LLP	1-574-253-8981	613 Victoria Court	Chardon	OH	44024	03/27/2010	07/18/2014	$180.04
S6369	Arcu Inc.	1-428-413-4720	570 Windsor Drive	Yorktown	VA	23693	10/15/2012	06/29/2013	$97.29
S3386	Orci Lobortis Augue Ltd	1-243-753-0311	978 Mill Street	Faribault	MN	55021	02/26/2013	11/29/2015	$93.24
S126	Cursus Inc.	1-786-379-4906	186 Glenwood Drive	Loxahatchee	FL	33470	04/12/2010	10/29/2014	$91.78
S1262	Ante Maecenas Mi Limited	1-666-928-0290	681 Route 32	Indianapolis	IN	46201	11/08/2010	01/20/2014	$116.74

2.4 Sample Queries for Database

Once a database has been developed, data must be retrieved using formal queries. We will show a few simple SQL queries of our database before getting into more complex queries.

```
SELECT * FROM ARTIST;
```

This query would result in all the columns from the ARTIST table being extracted. The SELECT command allows for the extraction of information directly from a table and the * operator allows for the retrieval of all columns. FROM specifies where the data will be pulled from, with ARTIST being the relation we're querying in this example. The query, in English, reads as: Select all attributes from the table Artist.

We can also be much more specific in the data we wish to retrieve from our database. We can choose which fields we want to view.

```
SELECT album_name, date_released FROM ALBUM WHERE album_ID = '12345';
```

In Relational Algebra, the equivalent expression is the following:

$$\pi_{\text{album\_name, date\_released}} (\sigma_{\text{album\_ID='123456'}} (\text{Album}))$$

Using a Select operation, we are able to narrow down the results to only those Albums with the ID being equivalent to '123456', but using a Project operation we only retrieve two columns: album_name and date_released.

```
SELECT artist_id, artist_name FROM ARTIST WHERE genre = 'Rock' ORDER BY DESCENDING;
```

Using SELECT, we specify which attributes we want retrieved, which are artist_ID and artist_name. Then, using the FROM operator, we specify we want data FROM the ARTIST table, but only WHERE the genre of the artist is 'Rock'. This will output all Rock Artists. In SQL, the

result set can also be ordered to be more appealing or for a specific purpose, as shown in the example with ORDER BY DESCENDING.

In the case where a count is required, the following query is an example of this.

```
SELECT COUNT (*) FROM ALBUM;
```

This query would return the amount of albums within our ALBUM entity. This information could be useful for production purposes.

The entire purpose of this section was to display how powerful queries can be. Practically anything and everything can be retrieved from the database with the correct queries. But these were very simple examples. In the following sections, examples queries are shown that are far more complex and expressive.

2.4.1 Design of Queries

The following list are ten non-trivial sample queries that are designed specific to our database. These are great examples to show the expressiveness of data retrieval through queries and will be later written using three formal languages: Relational Algebra, Tuple Relational Calculus, and Domain Relational Calculus.

1. Find all active artists who have recorded at Amoeba Studios.

2. Find all artists who have recorded at least two albums.

3. Find all writers who have only written one song.

4. Find a list of albums that contain only one song (singles).

5. Find albums that have at most three songs

6. Find the longest song.

7. Find the least expensive studio.

8. Find all artists who have recorded at all studios.

9. Find the albums that have been purchased by every buyer.

10. Find the most worked on album between Jan 10th 2011 and October 3rd, 2013.

2.4.2 Relational Algebra Expressions

Relational Algebra is procedural language, comprised of a set of operations used for defining queries to retrieve data stored in a Relational Database. A sequence of these operations is a relational algebra expression, whose output is a relation that contains data queried from a database. Below are ten query examples from our database written using Relational Algebra.

1. Find all active artists who have recorded at Amoeba Studios.

$$\pi_{\text{Artist\_name}} (\sigma_{\text{Studio\_name = “Amoeba“}} (\text{Studio} * \text{Album} * (\sigma_{\text{End\_date = null}}(\text{Artist} * \text{Contract}))))$$

---

2. Find all artists who have recorded at least two albums.

$$A1 \leftarrow (\text{Artist} * \text{Album})$$
$$A2 \leftarrow (\text{Artist} * \text{Album})$$
$$\pi_{\text{A1.artist\_Name}} (\sigma_{\text{A1.artistID = A2.artistID} \wedge \text{A1.albumID} \neq \text{A2.albumID}} (A1 \times A2))$$

---

3. Find all writers who have only written one song.

$$s1 \leftarrow (\text{Song} * \text{Writer})$$
$$s2 \leftarrow (\text{Song} * \text{Writer})$$
$$\pi_{\text{s1.fname, s1.lname}} (s1 - \pi_{\text{s1.*}} (\sigma_{\text{s1.writerID = s2.writerID}} (s1 \times s2)))$$

---

4. Find a list of albums that contain only one song (singles).

$$S1 \leftarrow \text{Song}$$
$$S2 \leftarrow \text{Song}$$
$$\text{NotSingles} \leftarrow \pi_{\text{s1.*}} (\sigma_{\text{s1.song\_ID} \neq \text{s2.song\_ID} \wedge \text{s1.album\_ID = s2.album\_ID}} (s1 \times s2))$$

π_{Album_Name} (Album * (π $_{album\_ID}$ (Song - NotSingles)))

---

5. Find albums that have at most three songs

π $_{album\_ID}$ (σ $_{COUNT\_song\_ID <= 3}$ ($_{album\_ID}$ $\Im$ $_{COUNT\ song\_ID}$ (Song * Album))

---

6. Find the longest song.

S1←Song
S2←Song
π $_{s1.song\_name, s1.song\_length}$ (σ $_{s1.song\_length > s2.song\_length}$ (S1 × S2))

---

7. Find the least expensive studio.

S1←Studio
S2←Studio
Expensive ← (π $_{s1.*}$ (σ $_{s1.hourlyCost > s2.hourlyCost}$ (S1 x S2)))
π $_{studio\_name, hourly\_Cost}$ (Studio - Expensive)

---

8. Find all artists who have recorded at all studios.

π $_{artist\_name}$ (Artist * ((π $_{artist\_ID, studio\_ID}$ (Album)) ÷ (π $_{studio\_ID}$ (Studio)))

---

9. Find the albums that have been purchased by every buyer.

π $_{album\_name, album\_ID}$ (π $_{albumID, buyer\_ID}$ (Album * SoldThrough * Transaction) ÷ (π $_{buyer\_ID}$ (Buyer))

---

10. Find the most worked on album between Jan 10th 2011 and October 3rd, 2013.

A1← σ $_{start\_date < 10/3/2013 \wedge end\_date > 1/10/2011}$ (Album)
A2← σ $_{start\_date < 10/3/2013 \wedge end\_date > 1/10/2011}$ (Album)
π $_{a1.album\_name}$ (σ $_{a1.hrs\_worked > a2.hrs\_worked}$ (A1 × A2))

2.4.3 Tuple Relational Calculus Expressions

Tuple Relational Calculus is a non-procedural language used to define what information to retrieve from a database, but not *how* to retrieve it. Tuple Relational Calculus expressions utilize variables that range over tuples from relations in a Relational Database. These expressions are based on predicate logic, which make use of the Existential (∃) and Universal (∀) quantifiers. Below are ten example queries written using Tuple Relational Calculus.

1. Find all active artists who have recorded at Amoeba Studios.

{ t | Artist(t) ∧ (∃c) (contract(c) ∧ c.end_date = null ∧ t.contract_id = c.contract_id ∧ (∃a ∃s) (Album(a) ∧ Studio(s) ∧ a.artist_ID = t.artist_ID ∧ a.studio_ID = s.studio_ID ∧ s.studio_name = "Ameoba Studios"))}

---

2. Find all artists who have recorded at least two albums.

{ t | Artist(t) ∧ (∃A1,∃A2) (Album(A1) ∧ Album(A2) ∧ A1.albumID ≠ A2.albumID ∧ A1.artistID = t.artist_ID ∧ A2.artistID = t.artist_ID) }

---

3. Find all writers who have only written one song.

{ w | Writer(w) ∧ (∃s1) (Song(s1) ∧ s1.writerID = w.writer_ID ∧ ¬(∃s2) (song (s2) ∧ s2.writer_ID = w.writer_ID ∧ s2.song_ID ≠ s1.song_ID)) }

---

4. Find a list of albums that contain only one song (singles).

{ a | Album(a) ∧ (∃s1) (Song(s1) ∧ s1.album_ID = a.album_ID ∧ ¬(∃s2) (song(s2) ∧ s2.album_ID = a.album_ID ∧ s2.songID ≠ s1.song_ID))}

---

5. Find albums that have at most three songs

{ a | album(a) ∧ ¬(∃s)(Song(s) ∧ s.album_ID = a.album_ID ∧ s.track_number >= 4) }

---

6. Find the longest song.

{ s | Song(s) ∧ ¬ (∃s2) (Song(s2) ∧ s2.length > s.song_length) }
When you move in the negation:
{ s | Song(s) ∧ (∀s2) ¬(Song(s2) ^ s2.length > s.song_length) }
{ s | Song(s) ∧ (∀s2) (¬Song(s2) ∨ ¬(s2.length > s.song_length)) }
{ s | Song(s) ∧ (∀s2) (¬Song(s2) ∨ (s.2length <= s.song_length)) }
{ s | Song(s) ∧ (∀s2) (Song(s2) → s2.length <= s.song_length)) }

---

7. Find the least expensive studio.

{ s | Studio(s) ∧ ¬(∃s2) (Studio(s2) ∧ s2.hourly_cost < s.hourly_cost) }
When you move in the negation:
{ s | Studio(s) ∧ (∀s2) (¬Studio(s2) ∨ ¬(s2.hourly_cost < s.hourly_cost) }
{ s | Studio(s) ∧ (∀s2) (¬Studio(s2) ∨ (s2.hourly_cost >= s.hourly_cost) }
{ s | Studio(s) ∧ (∀s2) (Studio(s2) → (s2.hourly_cost >= s.hourly_cost) }

---

8. Find all artists who have recorded at all studios.

{ t | Artist(t) ∧ (∀s) (Studio(s) → (∃a) (Album(a) ∧ a.artist_ID = t.artist_ID ∧ a.studio_ID = s.studio_ID)) }

---

9. Find the albums that have been purchased by every buyer.

{ a | Album(a) ∧ (∀b) (Buyer (b) → (∃t ∃s) (Transaction(t) ∧ Sold_Through(s) ∧ s.transaction_ID = t.transaction_ID ∧ s.album_ID = a.album_ID ∧ t.buyerID = b.buyerID)) }

---

10. Find the most worked on album between Jan 10th 2011 and October 3rd, 2013.

{ a | Album(a) ∧ a.start_date < 10/3/2013 ∧ a.end_date > 1/10/2011 ∧ ¬(∃ a2) (a2.hrs_worked > a.hrs_worked ∧ a2.start_date < 10/3/2013 ∧ a2.end_date > 1/10/2011) }

2.3.3 Domain Relational Calculus Expressions

Domain Relational Calculus is very similar to Tuple Relational Calculus. It is also a non-procedural language used to define what information to retrieve from a database, but not *how*

to retrieve it. These expressions are also based on predicate logic, which make use of the Existential (∃) and Universal (∀) quantifiers. The main difference is that Domain Relational Calculus and Tuple Relational Calculus is that Domain Relational Calculus uses variables that range over domains of attributes, not tuples. Below are ten example queries written using Tuple Relational Calculus.

1. Find all active artists who have recorded at Amoeba Studios.

{ <aID, cID, n> | Artist(aID, cID, n, _) ∧ contract(cID, _, _, !=null) ∧ (∃s ∃a) (Studio(s, "Amoeba Studios", _, _, _, _, _, _, _, _) ∧ Album(_, _, _, _, aID, s, _, _, _)}

---

2. Find all artists who have recorded at least two albums.

{ < n, aID > | Artist(aID, _, n, _) ∧ (∃A1,∃A2) (Album(A1, _, _, _, aID, _, _, _, _) ∧ Album(A2, _, _, _, aID, _, _, _,) ∧ A1!=A2}

---

3. Find all writers who have only written one song.

{ w | Writer(w, _, _, _, _, _, _, _, _, _) ∧ (∃s1) (Song(s1, _, _, _, _, w) ∧ ¬(∃s2)(Song(s2, _, _, _, _, w) ∧ s1 != s2))}

---

4. Find a list of albums that contain only one song (singles).

{ a | Album(a, _, _, _, _, _, _, _, _) ∧ (∃s1) (Song(s1, _, _, _, _, w) ∧ ¬(∃s2)(Song(s2,_,_, _,_,w) ∧ s1 != s2))}

---

5. Find albums that have at most three songs

{ aID | Album(aID, _, _, _, _, _, _, _, _) ∧ ¬(Song(_, >= 4, _, _, aID, _) }

---

6. Find the longest song.

{ <N,L> | (Song(_, _, N, L, _, _) ∧ ¬(∃L2)(Song(_, _, N, L2, _, _) ∧ L2 > L))}

---

7. Find the least expensive studio.

{ < n, c > | Studio(_, n, _, _, _, _, _, _, _, c) ∧ ¬(∃c2) (Studio(_, n, _, _, _, _, _, _, _, c2) ∧ c2 > c)) }

---

8. Find all artists who have recorded at all studios.

{ < aID, n > | Artist(aID, _, n, _) ∧ (∀s) (Studio(s, _, _, _, _, _, _, _, _, _) → (∃a) Album(_, _, _, _, aID, s, _,_, _))}

---

9. Find the albums that have been purchased by every buyer.

{ < aID, n > | Album(aID, n, _, _, _, _, _, _, _) ∧ (∀b) (Buyer(b, _, _) → (∃ tID) Transaction(tID, b, _) ∧ SoldThrough(tID, aID, _)) }

---

10. Find the most worked on album between Jan 10th 2011 and October 3rd, 2013.

{ < a, aID, hw > | Album(aID, n, _, _, _, _, < 10/3/2013, > 1/10/2011, hw) ∧ ¬(∃ hw2) (Album(aID, n, _, _, _, _, < 10/3/2013, > 1/10/2011, hw2) ∧ hw2 > hw) }

Phase 3: Implementation of Relational Database

3.1 Relation Normalization

3.1.1 Anomalies

For relations that have yet to be normalized, it is possible to encounter a number of issues. The main issues are insertion, deletion, and modification anomalies.

Insertion Anomalies

For poorly defined relation schema, insertion anomalies are common. For example, if our Album table contained all attributes for the Studio where each album was recorded at, and we were going to insert a new Album in the Album table, we would need to make sure that the Studio attributes were consistent between Album tuples. This would also make it difficult to insert a new Studio into the database since there are no Albums that were recorded there yet. The only workaround for this issue is to insert NULL values in the attributes for the Album, which could lead to integrity constraint issues, or to normalize the relation, which will be discussed later.

Deletion Anomalies

The main issue with deletion anomalies is connected to the second insertion anomaly. If we were to delete an Album from the Album table that could possibly be the only Album that was recorded at a particular Studio, the information that once existed for that Studio would be completely lost.

Modification Anomalies

Update Anomalies are when, in a poorly designed schema, updating a value in one tuple can cause inconsistencies across many other tuples in a relation. For example, if somebody were to change the value of an attribute of a Studio, all Album tuples that contain data for this Studio would need to be updated to reflect this new information. If this is not done correctly, massive data inconsistencies can result from this.

3.1.2 Normalization

Normalization is the process of analyzing database relations in order to have them conform to certain degree of normal, aimed at reducing redundant data and data anomalies. The normalization procedure consists of a series of tests done against relations, looking for opportunities to decompose the relations when necessary to minimize potential data issues.

The goal of normalization is done to minimize the amount of data redundancy that occurs, as well as to minimize insertion, deletion, and update anomalies, as was previously described. But there are varying degrees of normalization. The main types of normalization are as follows: First Normal Form, Second Normal Form, Third Normal Form, and Boyce-Codd Normal Form.

First Normal Form

First Normal Form, or 1NF, states that the domain of a particular attribute must only be atomic values, and that the value of any attribute must be a single value from the domain of that attribute. This does not allow for having a set of values, tuple of values, or a combination of the two. This also does not allow for nested relations. 1NF is now considered to be a part of the

definition of a relation in the relational model, were multi-valued attributes, and composite values are not allowed.

Second Normal Form

Second Normal Form, or 2NF, utilizes the concept of full functional dependency. The functional dependency A → B is described as having full functional dependency if upon discarding any attribute C from A means that the dependency does not hold. So for relations that have primary keys containing more than one attribute, any non-key attribute should not be functionally dependent on a part of the primary key. 2NF must also satisfy 1NF.

For example, if we were to have a member tuple where {SSN, address} → fname is a functional dependency. If we removed address from the dependency and it still held as true, then it would not conform to 2NF.

Third Normal Form

Third Normal Form, or 3NF, utilizes the concept of transitive dependency. The functional dependency between A → B in a relational schema T is a transitive dependency if there is a set of attributes C in T that is neither a candidate key or a subset of any key of T, as well as both A → C and C→B. In other words, no relation should have any non-key attributes functionally dependent on another single non-key attribute, or set of non-key attributes. 3NF must also satisfy 1NF and 2NF.

Boyce-Codd Normal Form

Boyce-Codd Normal Form, or BCNF, is similar to 3NF, but is much more strict and stronger. For relational schema R to be in BCNF it must be the case such that, if whenever a nontrivial functional dependency X → A holds in R, then X is a super key of R. The definition of BCNF only differs slightly from 3NF since the condition of 3NF which would allow A to be prime is not within the definition of BCNF. The absence of this condition makes BCNF a stronger normal form than 3NF.

3.1.3 Relation Normalization

Upon designing this relational database, we followed the 1NF rule that multi-valued attributes and composite values are not allowed. Our initial E-R Model design did not contain any multi-valued attributes or composite values to begin with, so upon converting the E-R model to Relational Model, the adherence to 1NF came naturally from our design.

Following the Normalization principles allowed us eliminate redundant information, such as for the Buyer relation. In early design stages, Buyer information was stored in each Transaction tuple, which created a potential problem with inconsistent data. But by decomposing the Transaction relation, we were able to create a Buyer relation that would centralize all Buyer information as and eliminate any redundant and inconsistent data.

The only potential modification anomaly that is currently present in our database is how our songs are stored in the Song relation. Each song tuple holds is track number, which signifies its place in the track listing of its respective album. This information must be stored in this

manner, so to minimize any problems, either a procedure will be created to correct any potential problems, or this will be handled on the back-end of the GUI.

3.2 SQL *Plus

3.2.1 Main Purpose

SQL *Plus is a powerful, yet simple, command-line program that allows the user to execute commands that allow for the creation of Oracle database components, as well as the ability to maintain and update an Oracle database. SQL *Plus also allows for the execution of SQL, and PL/SQL statements for querying, updating, or deleting data, as well as to create procedures or schema objects.

3.2.2 Oracle Schema Objects

Schema objects are logical structures of data in a database. Oracle allows many different types of schema objects for their relational databases. Some examples of schema objects are dimensions, sequences, synonyms, and clusters. The main objects that we are using so far and the objects we plan to use are listed as follows:

Tables

Tables are the basic structures used by databases to store data. They are composed of tuples and attributes.

Syntax:

```
CREATE TABLE tableName
{
        columnName1 dataType constraint1 constraint2 …,
        columnName2 dataType constraint1 constraint2…,
```

```
    …
    CONSTRAINT constraintName1 PRIMARY KEY (columnName, columnName,…),
    CONSTRAINT constraintName2 FOREIGN KEY (columnName) REFERENCES tableName
(columnName),
…
    CONSTRAINT constraintName3 CHECK(condition)
}
```

Views

A view is a logical, or virtual, table that is based on a stored query that gets data from already existing base tables. Views are useful in the case of building GUI's, as the application can pull its information from views, rather than base tables.

Syntax:

```
CREATE VIEW viewName AS
SELECT columnNames
FROM tableName
WHERE condition
```

Stored procedures

Stored procedures are a set of SQL statements that, together, perform a specific task on the database. They can take parameters, similar to functions, and can be used to encapsulate a set of operations.

Syntax:

```
CREATE OR REPLACE PRODEDURE procedureName
IS
BEGIN
    --CODE HERE
END;
```

Triggers

Triggers are procedures that are implicitly executed whenever an insert, delete, or update command is executed on the database.

Syntax:

```
CREATE TRIGGER triggerName
   triggerTime triggerEvent
   ON tableName FOR EACH ROW
   triggerBody
```

Indexes

An index is an object that contains an entry for each value that every indexed column contains. Indexes allow for direct and fast access to rows in the database, which makes queries perform much more efficiently.

Syntax:

```
CREATE INDEX indexName
ON tableName (columnName)
```

3.3 Relation Schema and Data

3.3.1 Contract

CONTRACT_ID	NOT NULL	VARCHAR2(10)
ALBUM_TERMS		NUMBER(5)
START_DATE		DATE
END_DATE		DATE

CONTRACT_ID	ALBUM_TERMS	START_DATE	END_DATE
C00001	10	23-AUG-10	
C17813	1	22-APR-10	01-APR-11
C44086	3	29-MAY-10	01-MAY-13
C11328	5	14-APR-11	
C4752	3	06-DEC-14	
C6789	2	20-SEP-10	01-SEP-12
C55390	5	22-OCT-13	
C85862	6	17-OCT-12	
C34728	2	08-APR-14	
C93471	4	22-NOV-12	
C50643	2	08-MAR-10	01-MAR-12
C19760	1	11-NOV-14	
C53622	1	04-SEP-11	01-SEP-12
C42579	2	18-JUN-12	01-JUN-14
C96710	4	05-APR-14	
C67044	5	23-AUG-10	

3.3.2 Artist

ARTIST_ID	NOT NULL	VARCHAR2(10)
CONTRACT_ID	NOT NULL	VARCHAR2(10)
ARTIST_NAME	NOT NULL	VARCHAR2(60)
GENRE		VARCHAR2(20)

ARTIST_ID	CONTRACT_ID	ARTIST_NAME	GENRE
A17813	C17813	Solomon	Folk
A44086	C44086	Bell	Rock
A11328	C11328	Hilda	Rock
A4752	C4752	Baker	Metal
A6789	C6789	Jamalia	Classic
A55390	C55390	Linus	Folk
A85862	C85862	Steven	Rock
A34728	C34728	Colette	Alternative
A93471	C93471	Adena	Folk
A50643	C50643	Angela	Rock
A19760	C19760	Ralph	SoftRock
A53622	C53622	Eliana	SoftRock
A42579	C42579	Chanda	Alternative
A96710	C96710	Yardley	Folk
A67044	C67044	Shaman	Rock
A00001	C00001	Queen	Rock

3.3.3 Composed_Of

ARTIST_ID	NOT NULL	VARCHAR2(10)
SSN	NOT NULL	VARCHAR2(9)

ARTIST_ID	SSN
A11328	030529568
A11328	230405066
A11328	237095808
A17813	094103269
A17813	095128618
A17813	115807312
A17813	390796801
A17813	718402914
A17813	799516609
A17813	898636127
A19760	934669708
A34728	065291852
A34728	326337324
A34728	681991076
A34728	794071975
A42579	018387601
A42579	647906445
A42579	737347478
A44086	316632012
A44086	466691455
A44086	706388832
A44086	913287463
A4752	033390750
A4752	174635988
A4752	350826131
A4752	522646239
A4752	555698350
A4752	752640200
A4752	966728262
A50643	315185134
A50643	493540743
A53622	103180494
A53622	130871020
A53622	519055209
A53622	648377849
A53622	689619138
A55390	022628780
A55390	359684742
A55390	445089490
A55390	694775842
A55390	727706860
A67044	055923612
A67044	196457896
A67044	249648859
A67044	355422240
A67044	372273921
A67044	539638320
A67044	703998621
A67044	800960314
A67044	815096438
A67044	985148004
A6789	060968225
A6789	080699386
A6789	380384098
A6789	704113574
A6789	936507878
A85862	297665184
A85862	727431600
A85862	835471116
A93471	238442093

3.3.4 Member

SSN	NOT NULL	VARCHAR2(10)
FNAME	NOT NULL	VARCHAR2(45)
LNAME		VARCHAR2(45)
PHONE		VARCHAR2(45)
ADDRESS		VARCHAR2(50)
CITY		VARCHAR2(45)
STATE		VARCHAR2(20)
ZIP		VARCHAR2(10)
INSTRUMENT		VARCHAR2(60)
START_DATE		DATE
END_DATE		DATE

SSN	FNAME	LNAME	PHONE	ADDRESS	CITY	STATE	ZIP	INSTRUMENT	START_DATE	END_DATE
115807312	Addison	Dane	17713253488	P.O. Box 140, 5851 Per Avenue	Indianapolis	IN	46201	Bass	08-NOV-10	20-JAN-14
898636127	Harlan	Dexter	19216245844	P.O. Box 447, 3845 Erat, Avenue	West Hopewell	NY	12533	Keyboard	10-MAY-12	26-OCT-15
095128618	Hiroko	Martina	17803698946	980-5300 Dictum Av.	Glasgow	KY	42141	Keyboard	19-FEB-12	20-APR-13
718402914	September	Ila	15351825258	Ap #937-3791 Metus St.	Mount Prospect	IL	60056	Guitar	03-MAR-09	14-OCT-14
094103269	Uta	Reed	12391761619	P.O. Box 527, 7610 Penatibus Rd.	Munster	IN	46321	Piano	08-JUL-11	19-DEC-13
390796801	Joan	Ferris	15107662458	762-2790 Condimentum. St.	Forest Hills	NY	11375	Vocals	09-JAN-13	26-MAY-13
799516609	Alden	Odysseus	13929405101	7839 Cras Ave	Palm Beach Gardens	FL	33410	Bass	01-MAR-11	01-SEP-13
114816401	Timon	Calista	11837899028	Ap #732-3331 Cursus Avenue	West Orange	NJ	7052	Vocals	09-JUN-10	05-AUG-13
466691455	Lara	Beau	12999436744	6780 Sapien, Street Brockton	MA	2301	Vocals	11-JAN-11	19-OCT-14	
913287463	Francesca	Bree	11054178578	564-7305 Tellus St.	Orland Park	IL	60462	Bass	04-AUG-09	11-APR-13
316632012	Isaac	Leila	15351339149	P.O. Box 350, 3397 Nulla Rd.	East Winona	MN	55987	Drums	01-JAN-11	05-JUL-15
706388832	Cameron	Tatum	11729694536	9623 Orci, Road	Chardon	OH	44024	Keyboard	27-MAR-10	18-JUL-14
230405066	Thaddeus	Josiah	19024567204	Ap #312-5175 Magna, Rd.	Yorktown	VA	23693	Bass	15-OCT-12	29-JUN-13
237095808	Thaddeus	Joan	17113412441	450-6815 Aliquet St.	Faribault	MN	55021	Drums	26-FEB-13	29-NOV-15
030529568	Daryl	Jillian	12628768431	Ap #524-446 Velit. Rd.	Loxahatchee	FL	33470	Bass	12-APR-10	29-OCT-14
350826131	Heather	Timon	13347742003	P.O. Box 843, 4176 Non Ave	Indianapolis	IN	46201	Drums	08-NOV-10	20-JAN-14
966728262	Berk	Jelani	17901978783	Ap #122-6362 Elementum Av.	Maryborough	IN	46201	Drums	08-NOV-10	20-JAN-14
174635988	Shellie	Kessie	14198627899	Ap #463-4485 Amet St.	Saint-Honor	NY	12533	Keyboard	10-MAY-12	26-OCT-15
555698350	Harding	Fay	16765128361	504-1999 Justo Ave	Bazel	KY	42141	Bass	19-FEB-12	20-APR-13
522646239	Zenia	Yvette	14372268794	796-9703 Arcu. St.	Brunn am Gebirge	IL	60056	Piano	03-MAR-09	14-OCT-14
033390750	Karly	Ruby	11251059052	Ap #390-9509 Sed, Street	Watford	IN	46321	Keyboard	08-JUL-11	19-DEC-13
752640200	Tucker	Kenyon	15661337495	925-6822 Elit Rd.	Harelbeke	NY	11375	Drums	09-JAN-13	26-MAY-13
060968225	Adam	Tanisha	14337015336	P.O. Box 282, 1071 Et Ave	Cleveland	FL	33410	Drums	01-MAR-11	01-SEP-13
704113574	Bradley	Reagan	12253670851	P.O. Box 587, 9953 In Avenue	Bomal	NJ	7052	Guitar	09-JUN-10	05-AUG-13
380384098	Kendall	Patricia	12738381323	7443 Gravida Rd.	Taunusstein	MA	2301	Bass	11-JAN-11	19-OCT-14
936507878	Rahim	Renee	12409485291	247 Gravida. Av.	Swadlincote	IL	60462	Guitar	04-AUG-09	11-APR-13
080699386	Shelley	Stella	11521752273	614 Tincidunt Street	Sint-Stevens-Woluwe	MN	55987	Vocals	01-JAN-11	05-JUL-15
445089490	Walter	Ingrid	17484190250	4674 Enim. Rd.	Reading	OH	44024	Keyboard	27-MAR-10	18-JUL-14
694775842	Sierra	Jin	16244768346	P.O. Box 393, 976 Erat, St.	Campbelltown	VA	23693	Bass	15-OCT-12	29-JUN-13
359684742	Abraham	Quinlan	15048003789	106 Non, St.	Dhuy	MN	55021	Drums	26-FEB-13	29-NOV-15
022628780	Lucas	Alec	11393378115	P.O. Box 105, 139 Imperdiet St.	Zonhoven	FL	33470	Piano	12-APR-10	29-OCT-14
727706860	Yoko	Jeanette	16788467744	P.O. Box 985, 1703 Commodo Avenue	VerÃ´s	IN	46201	Guitar	08-NOV-10	20-JAN-14
297665184	Heidi	Ainsley	11048623923	P.O. Box 514, 7993 Laoreet Road	Hattersheim am Main	IN	46201	Guitar	08-NOV-10	20-JAN-14
727431600	Kadeem	Ali	19621541797	748-7299 Scelerisque Ave	Cottbus	NY	12533	Vocals	10-MAY-12	26-OCT-15
835471116	Lacota	Ray	13776976552	1213 Tellus Avenue	Christchurch	KY	42141	Keyboard	19-FEB-12	20-APR-13
681991076	Unity	Fiona	13106428505	9362 Nullam Rd.	Senneville	IL	60056	Drums	03-MAR-09	14-OCT-14
326337324	Martha	Brock	16969940134	5945 Dolor Ave	Barrie	IN	46321	Piano	08-JUL-11	19-DEC-13
794071975	Willow	Honorato	17518668220	396-1218 Donec Rd.	Premeno	NY	11375	Piano	09-JAN-13	26-MAY-13
065291852	Ramona	Halee	19739433237	Ap #577-904 Mauris Road	West Valley City	FL	33410	Drums	01-MAR-11	01-SEP-13
238442093	Charles	Tara	19088701080	Ap #522-7038 Vivamus St.	Assebroek	NJ	7052	Piano	09-JUN-10	05-AUG-13
647759371	Yoshi	Anne	19523885101	P.O. Box 722, 9775 Placerat Avenue	Asso	MA	2301	Piano	11-JAN-11	19-OCT-14
493540743	Halla	Casey	18525750692	Ap #850-5478 Facilisis Road	Batiscan	IL	60462	Bass	04-AUG-09	11-APR-13
315185134	Fuller	Brock	19291249283	P.O. Box 281, 4648 Mauris Rd.	Kingston-on-Thames	MN	55987	Piano	01-JAN-11	05-JUL-15
934669708	Damian	Violet	17839535683	957-4329 Tortor. Street	Rathenow	OH	44024	Drums	27-MAR-10	18-JUL-14
103180494	Paki	Ezekiel	13987708648	P.O. Box 312, 3911 Fermentum St.	Berlin	VA	23693	Vocals	15-OCT-12	29-JUN-13
689619138	Imogene	Darius	17043880188	683-8741 Nonummy St.	St. Albert	MN	55021	Piano	26-FEB-13	29-NOV-15
519055209	Melvin	Joshua	11879813757	Ap #166-2136 Facilisis Rd.	Tramonti di Sopra	FL	33470	Drums	12-APR-10	29-OCT-14
130871020	Malachi	Nyssa	19205375019	855-7728 Risus Av.	Biggleswade	IN	46201	Guitar	08-NOV-10	20-JAN-14
648377849	Noel	Holly	18943263460	681-5092 Nunc Avenue	Makurdi	IN	46201	Drums	08-NOV-10	20-JAN-14
018387601	Martina	Blythe	13385460577	439-4198 Nisl. Road	Biez	NY	12533	Vocals	10-MAY-12	26-OCT-15
647906445	Judith	Kirby	18835144306	4565 Ante. Av.	Dsseldorf	KY	42141	Vocals	19-FEB-12	20-APR-13
737347478	Rhona	Rashad	14446238960	P.O. Box 736, 8952 Suscipit Ave	Juneau	IL	60056	Keyboard	03-MAR-09	14-OCT-14
043989774	Jamal	Jolene	19751486231	239-6604 Nunc Av.	Port Hope	IN	46321	Vocals	08-JUL-11	19-DEC-13
276392123	Jaime	Laurel	17332651472	Ap #835-5410 Libero Road	Arrone	NY	11375	Keyboard	09-JAN-13	26-MAY-13
800960314	Libby	Jeremy	15374107125	P.O. Box 350, 1722 Libero Street	Poitiers	FL	33410	Vocals	01-MAR-11	01-SEP-13
055923612	Audra	Fiona	18725601899	5953 Vel, St.	Cimitile	NJ	7052	Drums	09-JUN-10	05-AUG-13
372273921	Drake	Maggy	17333747345	754-5259 Orci, Rd.	Montebello	MA	2301	Bass	11-JAN-11	19-OCT-14
703998621	Ulla	Isabelle	15198931634	960 Eget St.	Wolverhampton	IL	60462	Piano	04-AUG-09	11-APR-13
815096438	Sonya	Cheyenne	11783674775	8433 Adipiscing Avenue	Paglieta	MN	55987	Drums	01-JAN-11	05-JUL-15
249648859	Axel	Nadine	18062981831	Ap #259-5148 Velit. Street	Fort Saskatchewan	OH	44024	Keyboard	27-MAR-10	18-JUL-14
539638320	Risa	Rosalyn	13012349405	Ap #499-7649 Neque Av.	Grand-Halleux	VA	23693	Drums	15-OCT-12	29-JUN-13
985148004	Margaret	Hedda	11248964998	2154 Penatibus Road	Cerreto di Spoleto	MN	55021	Piano	26-FEB-13	29-NOV-15
355422240	Ignacia	Ivory	13561342664	393-1514 Vel Rd.	Blind River	FL	33470	Vocals	12-APR-10	29-OCT-14
196457896	Cathleen	Linus	14143301934	Ap #858-3948 Tincidunt, Road	Caplan	IN	46201	Guitar	08-NOV-10	20-JAN-14

3.3.5 Album

ALBUM_ID	NOT NULL	VARCHAR2(10)
ALBUM_NAME	NOT NULL	VARCHAR2(60)
DATE_RELEASED		DATE
UNIT_PRICE		NUMBER(6,2)
ARTIST_ID	NOT NULL	VARCHAR2(10)
STUDIO_ID	NOT NULL	VARCHAR2(10)
START_DATE		DATE
END_DATE		DATE
HRS_WORKED		NUMBER(5)

ALBUM_ID	ALBUM_NAME	DATE_REL	UNIT_PRICE	ARTIST_ID	STUDIO_ID	START_DAT	END_DATE	HRS_WORKED
AL3626	Microscope	08-FEB-11	9.99	A17813	S1262	01-JAN-11	03-FEB-11	200
AL4620	Midget	07-FEB-13	12.99	A44086	S8241	17-APR-12	02-FEB-13	200
AL2151	Credit Card	18-JUL-11	3.99	A11328	S5458	24-APR-11	11-JUL-11	300
AL7741	System	15-APR-15	9.99	A4752	S5894	15-DEC-14	15-MAR-15	1000
AL2381	Observation	26-AUG-11	7.99	A6789	S6310	15-FEB-11	20-AUG-11	200
AL5377	Fork	15-JAN-15	24.99	A55390	S4552	14-SEP-14	26-DEC-14	300
AL5780	T-shirt	24-JAN-13	2.99	A85862	S1384	18-NOV-12	14-JAN-13	500
AL1803	Surgeon	15-DEC-14	1.99	A34728	S1099	17-NOV-14	01-DEC-14	700
AL5392	Wisdom	25-MAY-14	7.99	A93471	S1017	13-MAR-13	18-MAY-14	800
AL7282	Log	16-AUG-11	8.99	A50643	S2383	07-JUN-11	08-AUG-11	70
AL725	Laser	24-DEC-15	9.99	A19760	S3370	10-OCT-15	12-DEC-15	59
AL8817	Tap	17-JAN-12	9.99	A53622	S4099	25-SEP-11	01-JAN-12	300
AL9098	Critic	24-DEC-13	12.99	A42579	S6369	08-AUG-13	12-DEC-13	250
AL3881	Throne	19-JUN-15	14.99	A96710	S3386	02-FEB-15	06-JUN-15	50000
AL4688	Hazard	19-JAN-13	16.99	A67044	S126	05-OCT-12	01-JAN-13	300
AL0001	Queen	08-FEB-11	9.99	A00001	S1262	01-JAN-11	03-FEB-11	200
AL0002	Queen II	07-FEB-13	12.99	A00001	S8241	17-APR-12	02-FEB-13	200
AL0003	Sheer Heart Attack	18-JUL-11	3.99	A00001	S5458	24-APR-11	11-JUL-11	300
AL0004	A Night at the Opera	15-APR-15	9.99	A00001	S5894	15-DEC-14	15-MAR-15	1000
AL0005	Day at the Races	26-AUG-11	7.99	A00001	S6310	15-FEB-11	20-AUG-11	200
AL0006	News of the World	15-JAN-15	24.99	A00001	S4552	14-SEP-14	26-DEC-14	300
AL0007	Jazz	24-JAN-13	2.99	A00001	S1384	18-NOV-12	14-JAN-13	500
AL0008	The Game	15-DEC-14	1.99	A00001	S1099	17-NOV-14	01-DEC-14	700
AL0009	Hot Space	25-MAY-14	7.99	A00001	S1017	13-MAR-13	18-MAY-14	800
AL0010	The Works	16-AUG-11	8.99	A00001	S2383	07-JUN-11	08-AUG-11	70
AL0011	A Kind of Magic	24-DEC-15	9.99	A00001	S3370	10-OCT-15	12-DEC-15	59
AL0012	The Miracle	17-JAN-12	9.99	A00001	S4099	25-SEP-11	01-JAN-12	300
AL0013	Innuendo	24-DEC-13	12.99	A00001	S6369	08-AUG-13	12-DEC-13	250
AL0014	Made in Heaven	19-JUN-15	14.99	A00001	S3386	02-FEB-15	06-JUN-15	50000
AL0015	Live Killers	19-JAN-13	16.99	A00001	S126	05-OCT-12	01-JAN-13	300
AL4689	Blah	19-JAN-14	16.99	A67044	S126	05-OCT-13	01-JAN-14	40

3.3.6 Sold_Through

TRANSACTION_ID	NOT NULL	VARCHAR2(10)
ALBUM_ID	NOT NULL	VARCHAR2(10)
BATCH_UNITS		NUMBER(15)

TRANSACTION_ID	ALBUM_ID	BATCH_UNITS
T17813	AL3626	200000
T44086	AL4620	250000
T11328	AL2151	150000
T4752	AL7741	10000
T6789	AL2381	500000
T55390	AL5377	25000
T85862	AL5780	1000
T34728	AL1803	100
T93471	AL5392	55000
T50643	AL7282	230000
T19760	AL725	100000
T53622	AL8817	1500
T42579	AL9098	2000
T96710	AL3881	250
T67044	AL4688	100000
T00001	AL3626	100
T00001	AL4620	100
T00001	AL2151	100
T00001	AL7741	100
T00001	AL2381	100
T00001	AL5377	100
T00001	AL5780	100
T00001	AL1803	100
T00001	AL5392	100
T00001	AL7282	100
T00001	AL725	100
T00001	AL8817	100
T00001	AL9098	100
T00001	AL3881	100
T00001	AL4688	100
T00001	AL0001	100
T00001	AL0002	100
T00001	AL0003	100
T00001	AL0004	100
T00001	AL0005	100
T00001	AL0006	100
T00001	AL0007	100
T00001	AL0008	100
T00001	AL0009	100
T00001	AL0010	100
T00001	AL0011	100
T00001	AL0012	100
T00001	AL0013	100
T00001	AL0014	100
T00001	AL0015	100
T00001	AL4689	100

3.3.7 Transaction

TRANSACTION_ID	NOT NULL	VARCHAR2(10)
BUYER_ID	NOT NULL	VARCHAR2(10)
TDATE		DATE

TRANSACTION_ID	BUYER_ID	TDATE
T17813	B00001	05-OCT-15
T44086	B00005	26-JUN-14
T11328	B00003	25-JUL-14
T4752	B00004	09-SEP-14
T6789	B00005	23-SEP-15
T55390	B00006	21-MAR-15
T85862	B00004	21-NOV-15
T34728	B00005	18-SEP-14
T93471	B00006	10-APR-15
T50643	B00005	05-AUG-14
T19760	B00005	01-APR-15
T53622	B00001	21-JAN-14
T42579	B00004	30-NOV-13
T96710	B00006	19-SEP-15
T67044	B00001	05-OCT-15
T00001	B00002	01-NOV-15

3.3.8 Buyer

BUYER_ID	NOT NULL	VARCHAR2(10)
BUYER_NAME	NOT NULL	VARCHAR2(45)
PHONE		VARCHAR2(11)
ADDRESS		VARCHAR2(45)
CITY		VARCHAR2(25)
STATE		VARCHAR2(20)
ZIP		VARCHAR2(6)

BUYER_ID	BUYER_NAME	PHONE	ADDRESS	CITY	STATE	ZIP
B00001	FYE	14109410560	7910 Dolor. Road	Tacoma	WA	25345
B00002	Kmart	13798752434	8617 Adipiscing Rd.	West Valley City	UT	81502
B00003	Wal-Mart	19322717830	6554 Urna St.	Boise	ID	41529
B00004	Costco	12499787321	4592 Aliquet. Road	Aurora	CO	49142
B00005	Target	11843415591	1136 Id Street	Chicago	IL	34218
B00006	Amoeba Records	14813943965	9627 Amet St.	Jonesboro	AR	72604
B00007	The Warehouse	14634003980	4382 Augue St.	Madison	WI	37518
B00008	Suncoast	14217367118	6594 Urna Avenue	Boise	ID	40083
B00009	Shop Here Please	11219351648	5537 Donec Street	Springfield	IL	91158
B00010	Dollar General	14604429430	4400 Aliquet Av.	Kailua	HI	32090
B00011	One Stop Shop	14643488587	5161 Aenean Rd.	Billings	MT	94754
B00012	S-Mart	16567843345	8367 Habitant St.	San Diego	CA	92003
B00013	Quick Stop Groc.	16185947681	3545 Quisque St	Saint Paul	MN	69624
B00014	Sears	14575679254	2714 Nunc, Street	Davenport	IA	67485
B00015	Records R Us	16133361368	8167 Nibh. Road	Baltimore	MD	76135

3.3.9 Song

SONG_ID	NOT NULL	VARCHAR2(10)
TRACK_NUMBER		NUMBER(5)
SONG_NAME		VARCHAR2(60)
SONG_LENGTH		TIMESTAMP(6)
ALBUM_ID	NOT NULL	VARCHAR2(10)
WRITER_ID	NOT NULL	VARCHAR2(10)

SONG_ID	TRACK_NUMBER	SONG_NAME	SONG_LENGTH	ALBUM_ID	WRITER_ID
SO8882	1	Dude Dude	01-JAN-01 12.04.06.000000000 AM	AL3626	W1262
SO7345	2	Shut Clock	01-JAN-01 12.05.50.000000000 AM	AL3626	W8241
SO3375	3	Unkempt Bone	01-JAN-01 12.24.45.000000000 AM	AL3626	W8241
SO8701	4	Aquatic Downtown	01-JAN-01 12.47.34.000000000 AM	AL3626	W8241
SO9804	5	Premium Sign	01-JAN-01 12.31.03.000000000 AM	AL3626	W8241
SO0740	6	Old-Fashion Bead	01-JAN-01 12.39.50.000000000 AM	AL3626	W8241
SO8528	7	Neat Pigs	01-JAN-01 12.24.02.000000000 AM	AL3626	W8241
SO4682	8	Few Crack	01-JAN-01 12.02.27.000000000 AM	AL3626	W8241
SO8488	1	Flashy Ocean	01-JAN-01 12.09.09.000000000 AM	AL3626	W8241
SO1533	2	Faithful Whip	01-JAN-01 12.18.45.000000000 AM	AL3626	W8241
SO3099	3	Confused Screw	01-JAN-01 12.16.08.000000000 AM	AL4620	W8241
SO1814	4	Holy Hell	01-JAN-01 12.56.11.000000000 AM	AL4620	W8241
SO6409	1	Tent Treatment	01-JAN-01 12.45.16.000000000 AM	AL4620	W8241
SO3550	2	Rainstorm Apples	01-JAN-01 12.24.00.000000000 AM	AL4620	W8241
SO1839	3	Birth Soap	01-JAN-01 12.59.47.000000000 AM	AL4620	W8241
SO2571	4	Minute Popcorn	01-JAN-01 12.32.41.000000000 AM	AL4620	W8241
SO6969	5	Jewel Note	01-JAN-01 12.44.01.000000000 AM	AL4620	W8241
SO7506	6	Change Question	01-JAN-01 12.56.11.000000000 AM	AL4620	W5458
SO5478	6	Letter Circle	01-JAN-01 12.48.26.000000000 AM	AL4620	W5458
SO5772	7	Base Fork	01-JAN-01 12.43.20.000000000 AM	AL4620	W5458
SO1843	8	Good-Bye Meeting	01-JAN-01 12.43.02.000000000 AM	AL4620	W5458
SO4135	9	Fight Expansion	01-JAN-01 12.46.23.000000000 AM	AL4620	W5458
SO1915	1	Light syrup	01-JAN-01 12.21.20.000000000 AM	AL4620	W5458
SO2408	1	Object Book	01-JAN-01 12.36.25.000000000 AM	AL4620	W5458
SO0776	2	Marry Touch	01-JAN-01 12.19.30.000000000 AM	AL4620	W5458
SO8410	3	Justify Chin	01-JAN-01 12.48.05.000000000 AM	AL2151	W5894
SO8202	4	Bounce Stick	01-JAN-01 12.27.08.000000000 AM	AL2151	W5894
SO2521	5	Cheat Air	01-JAN-01 12.23.58.000000000 AM	AL2151	W5894
SO1231	6	Breed Uncle	01-JAN-01 12.09.32.000000000 AM	AL2151	W5894
SO8531	7	Mislead Beggar	01-JAN-01 12.29.44.000000000 AM	AL2151	W5894
SO3251	8	Analyze Secretary	01-JAN-01 12.24.57.000000000 AM	AL2151	W5894

SO3942	9	Exercise View	01-JAN-01 12.40.25.000000000 AM	AL2151	W5894
SO1421	10	Intend Approval	01-JAN-01 12.49.22.000000000 AM	AL2151	W5894
SO9412	11	That's too much	01-JAN-01 12.24.15.000000000 AM	AL7741	W5894
SO4618	12	Talented Turkey	01-JAN-01 12.48.52.000000000 AM	AL7741	W5894
SO0985	13	Arrogant Apples	01-JAN-01 12.13.50.000000000 AM	AL7741	W6310
SO6835	14	Young Yard	01-JAN-01 12.14.20.000000000 AM	AL7741	W6310
SO8417	15	Simple Scent	01-JAN-01 12.39.28.000000000 AM	AL7741	W6310
SO9423	16	Ten Town	01-JAN-01 12.46.51.000000000 AM	AL7741	W6310
SO4161	17	Comfortable Cactus	01-JAN-01 12.54.39.000000000 AM	AL7741	W6310
SO3935	1	Cooing Crayon	01-JAN-01 12.31.24.000000000 AM	AL7741	W6310
SO8108	2	Pricey Party	01-JAN-01 12.28.35.000000000 AM	AL7741	W6310
SO0478	3	Poised Plastic	01-JAN-01 12.16.15.000000000 AM	AL7741	W6310
SO5138	4	Silly Scale	01-JAN-01 12.21.13.000000000 AM	AL7741	W4552
SO3275	5	Ketchup and A1	01-JAN-01 12.53.08.000000000 AM	AL7741	W4552
SO2116	6	Wax War	01-JAN-01 12.41.17.000000000 AM	AL7741	W4552
SO3114	7	Art Ants	01-JAN-01 12.47.59.000000000 AM	AL7741	W4552
SO6665	8	Mind Minister	01-JAN-01 12.15.46.000000000 AM	AL7741	W4552
SO6383	9	Mother Mice	01-JAN-01 12.01.30.000000000 AM	AL7741	W4552
SO4064	1	Texture Tramp	01-JAN-01 12.51.59.000000000 AM	AL7741	W4552
SO0519	2	Bread Bean	01-JAN-01 12.10.20.000000000 AM	AL2381	W4552
SO6505	1	Ship Space	01-JAN-01 12.52.16.000000000 AM	AL2381	W4552
SO8475	2	Prose Produce	01-JAN-01 12.57.00.000000000 AM	AL2381	W4552
SO8551	1	Plough Pump	01-JAN-01 12.55.40.000000000 AM	AL2381	W4552
SO1457	1	Skate Spot	01-JAN-01 12.06.51.000000000 AM	AL5377	W1384
SO6451	2	Don't touch me	01-JAN-01 12.59.34.000000000 AM	AL5377	W1384
SO9401	3	Ascertain Army	01-JAN-01 12.19.33.000000000 AM	AL5377	W1384
SO0623	4	Jail Join	01-JAN-01 12.29.47.000000000 AM	AL5377	W1384
SO0892	5	Branch Bath	01-JAN-01 12.29.49.000000000 AM	AL5377	W1384
SO3588	6	Sense Slope	01-JAN-01 12.52.46.000000000 AM	AL5377	W1384
SO6478	7	Search Stop	01-JAN-01 12.02.39.000000000 AM	AL1803	W1384
SO4088	8	Smile Stew	01-JAN-01 12.09.21.000000000 AM	AL1803	W1384
SO4779	9	Bump Beam	01-JAN-01 12.52.40.000000000 AM	AL1803	W1384
SO0859	10	Receive Reward	01-JAN-01 12.53.48.000000000 AM	AL1803	W1384
SO3772	11	Sell Slope	01-JAN-01 12.02.44.000000000 AM	AL1803	W1384
SO7864	12	Misunderstan Meal	01-JAN-01 12.32.24.000000000 AM	AL1803	W1384
SO7597	13	Cough Range	01-JAN-01 12.13.18.000000000 AM	AL1803	W1099
SO0073	14	Zipper Camp	01-JAN-01 12.27.36.000000000 AM	AL1803	W1099
SO5223	1	Heat Vegetable	01-JAN-01 12.19.09.000000000 AM	AL1803	W1099
SO1528	2	Clouds Disgust	01-JAN-01 12.10.11.000000000 AM	AL5392	W1099
SO6312	3	Touch me	01-JAN-01 12.51.45.000000000 AM	AL5392	W1099
SO7996	4	Attack Uncle	01-JAN-01 12.17.16.000000000 AM	AL5392	W1099
SO3087	5	Determine Dress	01-JAN-01 12.50.10.000000000 AM	AL5392	W1099
SO6947	6	Recommend Direct	01-JAN-01 12.41.12.000000000 AM	AL5392	W1099
SO3981	7	Scream Birthday	01-JAN-01 12.05.18.000000000 AM	AL5392	W1099
SO9372	8	Fax Secretary	01-JAN-01 12.03.05.000000000 AM	AL5392	W1099
SO2585	9	Process Road	01-JAN-01 12.59.53.000000000 AM	AL5392	W1099
SO3545	1	Squash Pickle	01-JAN-01 12.03.48.000000000 AM	AL5392	W1099
SO7466	2	Diagnose Bomb	01-JAN-01 12.25.07.000000000 AM	AL5392	W1017
SO7763	3	Experiment Author	01-JAN-01 12.40.34.000000000 AM	AL7282	W1017
SO8381	4	Finalize Plantation	01-JAN-01 12.20.38.000000000 AM	AL7282	W1017
SO3969	5	Help me	01-JAN-01 12.59.55.000000000 AM	AL7282	W1017
SO1657	6	Elite Edge	01-JAN-01 12.44.25.000000000 AM	AL7282	W2383
SO5914	7	Pushy Pollution	01-JAN-01 12.39.28.000000000 AM	AL7282	W2383
SO9667	8	Disillusioned Donk	01-JAN-01 12.02.07.000000000 AM	AL7282	W2383
SO1319	9	Uppity Underwear	01-JAN-01 12.10.37.000000000 AM	AL7282	W2383
SO2713	10	Ad Aunt	01-JAN-01 12.15.56.000000000 AM	AL7282	W2383
SO2932	1	Tacit Tooth	01-JAN-01 12.30.55.000000000 AM	AL725	W2383
SO2873	2	Married Morning	01-JAN-01 12.51.39.000000000 AM	AL8817	W3370
SO3144	3	Fallacious Fuel	01-JAN-01 12.22.18.000000000 AM	AL8817	W3370
SO7803	4	Massive Mass	01-JAN-01 12.09.18.000000000 AM	AL9098	W3370
SO1986	5	Careless Chair	01-JAN-01 12.44.41.000000000 AM	AL9098	W3370
SO7525	6	It was the dog	01-JAN-01 12.57.32.000000000 AM	AL9098	W3370
SO6298	7	Smoke Shame	01-JAN-01 12.16.13.000000000 AM	AL3881	W3370
SO1895	8	Crate Chance	01-JAN-01 12.05.17.000000000 AM	AL3881	W3370
SO9648	9	Square Shop	01-JAN-01 12.52.50.000000000 AM	AL3881	W4099
SO7926	10	Juice Jewel	01-JAN-01 12.33.24.000000000 AM	AL4688	W6369
SO4249	11	Blow Baseball	01-JAN-01 12.52.11.000000000 AM	AL4688	W3386
SO1891	12	Business Beam	01-JAN-01 12.59.14.000000000 AM	AL4688	W126
SO4994	13	Stomach Shirt	01-JAN-01 12.17.37.000000000 AM	AL4688	W126

3.3.10 Studio

STUDIO_ID	NOT NULL	VARCHAR2(10)
STUDIO_NAME	NOT NULL	VARCHAR2(45)
PHONE		VARCHAR2(11)
ADDRESS		VARCHAR2(45)
CITY		VARCHAR2(25)
STATE		VARCHAR2(20)
ZIP		VARCHAR2(10)
OPEN_DATE		DATE
CLOSE_DATE		DATE
HOURLY_COST		NUMBER(6,2)

STUDIO_ID	STUDIO_NAME	PHONE	ADDRESS	CITY	STATE	ZIP	OPEN_DATE	CLOSE_DATE	HOURLY_COST
S1262	Ante Maecenas Mi Limited	17863794906	681 Route 32	Indianapolis	IN	46201	08-NOV-10		116.74
S8241	Pede Malesuada Industries	16669280290	747 2nd Street	West Hopewell J.	NY	12533	10-MAY-12	26-OCT-15	86.88
S5458	Duis Institute	15538665682	621 Main Street	Glasgow	KY	42141	19-FEB-12		110.44
S5894	Nec Euismod LLC	12491293089	458 Inverness Dr	Mount Prospect	IL	60056	03-MAR-09	14-OCT-14	98.69
S6310	Imperdiet Ornare In Associates	16645656363	157 Williams Street	Munster	IN	46321	08-JUL-11		114.62
S4552	Magna Institute	12237234437	761 Ann Street	Forest Hills	NY	11375	09-JAN-13	26-MAY-13	136.5
S1384	Etiam Ligula Corp.	11458115254	275 Beech Street	Palm Beach Gard.	FL	33410	01-MAR-11		135.69
S1099	Est Mauris Rhoncus Corp.	13184131114	961 Route 44	West Orange	NJ	7052	09-JUN-10	05-AUG-13	154.24
S1017	Aenean Massa Associates	12383431057	657 Locust Street	Brockton	MA	2301	11-JAN-11		197.25
S2383	Est Mollis Non LLP	15729530248	957 Penn Street	Orland Park	IL	60462	04-AUG-09	11-APR-13	135.08
S3370	Eu Accumsan Sed Inc.	17498523465	209 Main Street	East Winona	MN	55987	01-JAN-11	05-JUL-15	171.32
S4099	Ultricies Adipiscing LLP	15742538981	613 Victoria Court	Chardon	OH	44024	27-MAR-10		180.04
S6369	Arcu Inc.	14284134720	570 Windsor Drive	Yorktown	VA	23693	15-OCT-12	29-JUN-13	97.29
S3386	Orci Lobortis Augue Ltd	12437530311	978 Mill Street	Faribault	MN	55021	26-FEB-13		93.24
S126	Cursus Inc.	17863794906	186 Glenwood Dr	Loxahatchee	FL	33470	12-APR-10	29-OCT-14	91.78

3.3.11 Writer

WRITER_ID	NOT NULL	VARCHAR2(10)
FNAME		VARCHAR2(45)
LNAME		VARCHAR2(45)
SSN	NOT NULL	VARCHAR2(9)
PHONE		VARCHAR2(11)
ADDRESS		VARCHAR2(45)
CITY		VARCHAR2(25)
STATE		VARCHAR2(20)
ZIP		VARCHAR2(6)
START_DATE		DATE
END_DATE		DATE

WRITER_ID	FNAME	LNAME	SSN	PHONE	ADDRESS	CITY	STATE	ZIP	START_DATE	END_DATE
W1262	Rhea	Amaya	219639062	17863794906	681 Route 32	Indianapolis	IN	46201	08-NOV-10	20-JAN-14
W8241	Neil	Christopher	217809469	16669280290	747 2nd Street	West Hopewell J.	NY	12533	10-MAY-12	
W5458	Rafael	Henry	235863894	15538665682	621 Main Street	Glasgow	KY	42141	19-FEB-12	20-APR-13
W5894	Sarah	Jorden	476350140	12491293089	458 Inverness Dr	Mount Prospect	IL	60056	03-MAR-09	14-OCT-14
W6310	Angela	Leilani	439861590	16645656363	157 Williams Street	Munster	IN	46321	08-JUL-11	
W4552	Idola	Aline	329740439	12237234437	761 Ann Street	Forest Hills	NY	11375	09-JAN-13	26-MAY-13
W1384	Cody	Yetta	166200007	11458115254	275 Beech Street	Palm Beach Gard	FL	33410	01-MAR-11	
W1099	Jermaine	Wayne	578269691	13184131114	961 Route 44	West Orange	NJ	7052	09-JUN-10	05-AUG-13
W1017	Jocelyn	Patricia	478321350	12383431057	657 Locust Street	Brockton	MA	2301	11-JAN-11	19-OCT-14
W2383	Fay	Cleo	680225134	15729530248	957 Penn Street	Orland Park	IL	60462	04-AUG-09	
W3370	Sharon	Hamilton	508425295	17498523465	209 Main Street	East Winona	MN	55987	01-JAN-11	05-JUL-15
W4099	Fletcher	Kristen	358643701	15742538981	613 Victoria Court	Chardon	OH	44024	27-MAR-10	18-JUL-14
W6369	Carolyn	Karen	530530524	14284134720	570 Windsor Drive	Yorktown	VA	23693	15-OCT-12	
W3386	Glenna	Fatima	36077409	12437530311	978 Mill Street	Faribault	MN	55021	26-FEB-13	29-NOV-15
W126	Rhea	Amaya	576617175	17863794906	186 Glenwood Dr	Loxahatchee	FL	33470	12-APR-10	29-OCT-14

3.4 SQL Queries

The following queries are translated from the Phase 2 relational algebra and relational calculus queries.

1. Find all active artists who have recorded at Amoeba Studios.

```
SELECT DISTINCT art.artist_name
FROM artist art, studio s, album al, contract con
WHERE art.contract_id = con.contract_id AND
        con.end_date is null AND
        al.artist_id = art.artist_id AND
        al.studio_id = s.studio_id AND
        s.studio_name = 'Cursus Inc.'
```

Output:

Artist_name
Shaman
Queen

---

2. Find all artists who have recorded at least two albums.

```
SELECT DISTINCT a.artist_id, a.artist_name
FROM artist a
INNER JOIN album al
ON a.artist_id = al.artist_id
WHERE a.artist_id IN (
        SELECT artist_id
        FROM album
        GROUP BY artist_id
        HAVING COUNT (*) >= 2
        )
```

Output:

Artist_ID	Artist_Name
A1010	Led Zeppelin
A67044	Shaman
A00001	Queen

---

3. Find all writers who have only written one song.

```
SELECT DISTINCT w.writer_id, w.fname, w.lname
FROM writer w
INNER JOIN song s
ON w.writer_id = s.writer_id
WHERE w.writer_id IN (
        SELECT writer_id
        FROM song
        GROUP BY writer_id
        HAVING COUNT (*) = 1
        )
```

Output:

Writer_ID	Fname	Lname
W127	Rhea	Amaya
W1264	Rhea	Amaya

---

4. Find a list of albums that contain only one song (singles).

```
SELECT distinct al.album_id, al.album_name
FROM album al
INNER JOIN song s
ON al.album_id = s.album_id
WHERE al.album_id IN (
        SELECT album_id
        FROM song
        GROUP BY album_id
        HAVING COUNT (*) = 1
        )
```

Output:

Album_ID	Album_Name
AL1313	Coda
AL2020	I Am the Greatest Ever
AL725	Laser
AL0004	A Night at the Opera

---

5. Find albums that have at most three songs

```
SELECT DISTINCT al.album_id, al.album_name
FROM album al
INNER JOIN song s
ON al.album_id = s.album_id
WHERE al.album_id IN (
        SELECT album_id
        FROM song
        GROUP BY album_id
        HAVING COUNT (*) >= 3
        )
```

Output:

Album_ID	Album_Name
AL7741	System
AL5377	Fork
AL7282	Log
AL1803	Surgeon
AL3881	Throne
AL3627	Microscope
AL4688	Hazard
AL4621	Midget
AL2381	Observation
AL2151	Credit Card
AL5392	Wisdom

---

6. Find the longest song.

```
SELECT s.song_id, s.song_name
FROM song s
WHERE NOT EXISTS (
        SELECT s2.song_id, s2.song_length
        FROM song s2
        WHERE s2.song_length > s.song_length
        )
```

Output:

Song_ID	Album_Name
S0010	Death on Two Legs

---

7. Find the least expensive studio.

```
SELECT s.studio_id, s.studio_name
FROM studio s
WHERE NOT EXISTS (
        SELECT s2.HOURLY_COST
        FROM _studio s2
        WHERE s2.hourly_cost < s.hourly_cost )
```

Output:

Studio_ID	Studio_Name
S8241	Pede Malesuada Industries

---

8. Find all artists who have recorded at all studios.

```
SELECT a.artist_id, a.artist_name
FROM artist a
INNER JOIN album al
ON al.artist_id = a.artist_id
GROUP BY a.artist_id, a.artist_name
HAVING COUNT (distinct al.studio_id) = (SELECT COUNT (*) FROM studio)
```

Output:

Artist_id	Artist_name
A00001	Queen

---

9. Find the albums that have been purchased by every buyer.

```
SELECT b.buyer_id, b.buyer_name
FROM buyer b
INNER JOIN transaction t
ON b.buyer_id = t.buyer_id
INNER JOIN sold_through st
ON st.transaction_id = t.transaction_id
GROUP BY b.buyer_id, b.buyer_name
HAVING COUNT (DISTINCT st.album_id) = (SELECT COUNT (*) FROM album)
```

Output:

Buyer_id	Buyer_name
B00002	Kmart

---

10. Find the most worked on album between Jan 10th 2011 and October 3rd, 2013.

```
CREATE OR REPLACE VIEW transaction_info AS
SELECT a.album_id, a.album_name, a.hrs_worked
FROM album a
WHERE a.start_date < DATE '2013-10-03'
AND a.end_date > DATE '2011-01-10'
AND NOT EXISTS (
        SELECT a2.hrs_worked
        FROM album a2
        WHERE a2.start_date < DATE '2013-10-03'
        AND a2.end_date > DATE '2011-01-10'
        AND a2.hrs_worked > a.hrs_worked
        )
```

Output:

Album ID	Album Name	Hrs_Worked
AL56	Physical Graffiti	8000

---

11. For all transactions, find the total number of album units purchased, as well as the total cost for each transaction.

```
SELECT t.transaction_id, b.buyer_name, (
        SELECT sum(st.batch_units)
        FROM maal_sold_through st
        WHERE st.transaction_id = t.transaction_id
        ) AS Total_units, (
        SELECT sum(al.unit_price * st.batch_units)
        FROM album al
        INNER JOIN sold_through st
        ON al.album_id = st.album_id
        WHERE st.transaction_id = t.transaction_id
        ) AS Total_Price, t.tdate
FROM transaction t
INNER JOIN buyer b
ON b.buyer_id = t.buyer_id
```

Output:

Transaction_ID	Buyer_name	Total_units	Total_price	Tdate
T53622	FYE	1500	14985	21-JAN-14
T67044	FYE	100000	1699000	05-OCT-15
T00001	Kmart	3000	31326	01-NOV-15
T11328	Wal-Mart	150000	598500	25-JUL-14
T4752	Costco	10000	99900	09-SEP-14

T85862	Costco	1000	2990	21-NOV-15
T44086	Target	250000	3247500	26-JUN-14
T6789	Target	500000	3995000	23-SEP-15
T34728	Target	100	199	18-SEP-14
T50643	Target	230000	2067700	05-AUG-14
T19760	Target	100000	999000	01-APR-15
T55390	Amoeba Records	25000	624750	21-MAR-15
T93471	Amoeba Records	55000	439450	10-APR-15
T96710	Amoeba Records	250	3747.5	19-SEP-15

Phase 4: Stored Procedures, Packages, and Triggers

4.1 Oracle PL/SQL

4.1.1 What is PL/SQL?

PL/SQL, which stands for Procedural Language extensions to SQL, is a language used by Oracle in order to add more programming ability to the SQL language for creating more complex objects and operations. PL/SQL has procedural language attributes such as conditions and loops and allows declaration of constants and variables, as well as functions and error handling.

Benefits of PL/SQL

PL/SQL allows for the creation of stored procedures, functions, and triggers. These objects are very useful because:

1. They allow for database automation. Instead of having to worry about complicated updates and deletes, procedures can be created that will handle all of this for you automatically, instead of having to execute the individual statements in the client, which can be less secure.
2. If a database is used by many applications written in different languages, it can then be stored in the database and then called by any of those different applications.
3. There is less latency, or quicker response time, when stored procedures are used.
4. A procedure can assist in creating a view by allowing for a more complex type of derived data to become available to the user. They can also be used to check for more advanced constraints.
5. Error handling can be stored as part of the procedure.

Control Statements

PL/SQL has a number of control statements that are very useful for creating procedures. There are three categories of PL/SQL control statements. They are:

1. Conditional Selection Statement: Runs different statements for different data value. These statements are IF and CASE.

2. Loop Statements: Run the same statements with a series of different data values. The loop statements are the LOOP, FOR LOOP, and WHILE LOOP.

3. Sequential Control Statements: Are not necessary to PL/SQL, but can be used. The sequential control statements are GOTO and NULL.

4.1.2 PL/SQL Syntax

Stored Procedure

Stored procedures are similar to functions, in that they are a set of stored PL/SQL statements that can be executed any number of times. These procedures can also take in parameters. Stored procedures are helpful in that instead of explicitly running the same series of PL/SQL statements over and over, you can just store it once and call it as many times as necessary, which is more efficient.

Syntax:

```
CREATE [OR REPLACE] PROCEDURE <procedure name> [list of parameters]
IS
<Declaration Section>
BEGIN
<procedure body>
END;
```

Stored Functions

A stored function is a set of PL/SQL statements that can be called by a function name. They are almost the same as a stored procedures, except a function returns a value where it is called. A procedure may or may not return a value.

Syntax:

```
CREATE [OR REPLACE] FUNCTION <function name> [parameters]
RETURN <return datatype>;
IS
<Declaration section>
BEGIN
        <Function Body>
        Return <return variable>;
      EXCEPTION
        <Exception section>
        Return <return variable>;
END;
```

Packages

Packages are schema objects that are used to logically group related procedures and functions, as well as variables. Upon grouping these items into a package, an application that is using these subprograms only needs to know the name of the subprogram and the parameters needed for these subprograms, and does not need to know any of the specific implementation details. This idea is similar to prototypes and function bodies in C++.

Syntax:

```
CREATE PACKAGE <package name>  AS
<variables and their specifications>
PROCEDURE A1
PROCEDURE A2
PROCEDURE A3
END <package name>
```

Triggers

Triggers are programs or procedures that are stored in the database and implicitly executed before, after, or instead of an update, insert, or deletion. This is a method of specifying

certain rules, and will help to enforce these rules whenever data has been modified within a table. The below code is a sample of how trigger syntax looks.

Syntax:

```
CREATE TRIGGER <trigger name>
<BEFORE | AFTER | INSTEAD OF>
<INSERT | DELETE | UPDATE>
<OF column name>
ON <table name>
FOR EACH ROW
WHEN <conditions>
BEGIN
<Desired statement go here>
END;
```

Cursor

A Cursor is a temporary work space created in the system memory when SQL statements are executed. Cursors allows you to give a select statement a name, so that you can then access the information retrieved in that select statement in some kind of procedure. There are two types of cursors, which are explicit and implicit. Both have the same functionality but are different in the way they are accessed.

Syntax:

```
CURSOR <cursor name> IS
        <Desired FUNCTIONALITY>
BEGIN
        OPEN <cursor name>
        <BODY>
        CLOSE <cursor name>
END;
```

4.2 MS SQL Server and MySQL Stored Procedures

4.2.1 Microsoft SQL Server and T-SQL

Microsoft SQL Server uses an extension of SQL called **T-SQL**, short for **Transaction-SQL**, which, similar to PL/SQL, has a number of features that are not available with SQL, including procedural programming and variables, which allow for the creation of stored procedures.

A major feature that T-SQL has, that differs from Oracle, is that the DELETE and UPDATE commands both allow for the inclusion of the FROM clause which allows that usage of JOINS, making filtering records much easier, and the deletion of records far easier than in PL/SQL.

T-SQL Procedure

The major difference between T-SQL and PL/SQL is parameter passing. PL/SQL uses IN, OUT, and INOUT to differentiate between different types of parameters. T-SQL uses OUT, OUTPUT, and READONLY for this purpose, which is the same idea as what Oracle implements, but just with different syntax. Another key difference is the use of the @ to signify the use of a variable. T-SQL also allows the use of a SELECT statement without having to use the keyword INTO, which PL/SQL requires when assigning SELECT results into a variable. Despite these differences, the overall structure is very similar to that of PL/SQL.

Syntax:

```
CREATE { PROCEDURE | PROC } [schema_name.]procedure_name
  [ @parameter [type_schema_name.] datatype
    [ VARYING ] [ = default ] [ OUT | OUTPUT | READONLY ]
  , @parameter [type_schema_name.] datatype
    [ VARYING ] [ = default ] [ OUT | OUTPUT | READONLY ] ]
[ WITH { ENCRYPTION | RECOMPILE | EXECUTE AS Clause } ]
```

```
[ FOR REPLICATION ]
AS
BEGIN
  [declaration_section]
  executable_section
END;
```

T-SQL Function

T-SQL Functions are very similar to T-SQL procedures, in terms of syntax and semantics. Functions in T-SQL also make use of OUT, OUTPUT, and READONLY variable types, and their overall structure is very similar to that of PL/SQL. The symbol @ is also used throughout for variables and parameters.

Syntax:

```
CREATE FUNCTION [schema_name.]function_name
( [ @parameter [ AS ] [type_schema_name.] datatype
   [ = default ] [ READONLY ]
  , @parameter [ AS ] [type_schema_name.] datatype
   [ = default ] [ READONLY ] ]
)
RETURNS return_datatype
[ WITH { ENCRYPTION
      | SCHEMABINDING
      | RETURNS NULL ON NULL INPUT
      | CALLED ON NULL INPUT
      | EXECUTE AS Clause ]
[ AS ]
BEGIN
  [declaration_section]
  executable_section
  RETURN return_value
END;
```

4.2.2 MySQL Server Routines

MySQL Stored Procedure

MySQL uses what are called Routines, which are equivalent to Procedures in PL/SQL, and are very similar to Oracle Procedures in terms of syntax and semantics, as they have the ability to use cursors, as well as all control statements Oracle has, such as IF, ELSE, case statements, and loops. Similar to Oracle, MySQL also allows the use of IN, OUT, and INOUT parameter passing. And MySQL also requires that you use the keyword INTO when using SELECT in a procedure, which Oracle also requires.

Syntax:

```
CREATE
   [DEFINER = { user | CURRENT_USER }]
   PROCEDURE sp_name ([proc_parameter[,...]])
   [characteristic ...]
BEGIN
routine_body
END
```

MySQL Function

MySQL Functions are very similar to Oracle functions, but in MySQL you cannot use OUT or INOUT parameters. By default, all parameters are IN and this cannot be changed. This is a major difference between Oracle, which allows the use of all three types of parameters in functions and procedures. MySQL functions are also only allowed to return a single value.

Syntax:

```
CREATE
   [DEFINER = { user | CURRENT_USER }]
   FUNCTION sp_name ([func_parameter[,...]])
   RETURNS type
```

[characteristic ...]
BEGIN
routine_body
End

4.3 PL/SQL Subprogram Implementations

4.3.1 Procedures

The following procedures were created for our database. They serve various functions and are specific to our database.

Delete Album Procedure

The following procedure is used to delete an Album and all of its Songs from their respective tables in the correct order to avoid violating referential constraints.

```
CREATE OR REPLACE PROCEDURE maal_delete_album(aid IN maal_album.album_id%type)
IS
BEGIN
    DELETE FROM maal_song
    WHERE maal_song.album_id= maal_delete_album.aid;
    DELETE FROM maal_album
    WHERE maal_album.album_id = maal_delete_album.aid;
COMMIT;
END;
```

Delete Artist Procedure

The following procedure is used to delete an Artist from the database. In order to fully delete an Artist from the database, all albums, songs, contract information, and transaction information must be deleted to avoid any referential constraint problems.

```
CREATE OR REPLACE PROCEDURE maal_delete_artist_proc(aid IN maal_artist.artist_id%type)
IS
BEGIN
    DELETE FROM maal_song
    WHERE album_id IN
    (
        SELECT album_id FROM maal_song
        NATURAL JOIN maal_album
        WHERE artist_id = maal_delete_artist_proc.aid
    );
    DELETE FROM maal_composed_of
    WHERE maal_composed_of.artist_id = maal_delete_artist_proc.aid;
    DELETE FROM maal_member
    WHERE ssn IN
    (
        SELECT ssn FROM maal_member
        NATURAL JOIN maal_composed_of
        WHERE maal_composed_of.artist_id = maal_delete_artist_proc.aid
    );
    DELETE FROM maal_sold_through
    WHERE album_id IN
    (
        SELECT album_id FROM maal_sold_through
        NATURAL JOIN maal_album
        WHERE maal_album.artist_id = maal_delete_artist_proc.aid
    );
    DELETE FROM maal_album
    WHERE maal_album.artist_id = maal_delete_artist_proc.aid;
    DELETE FROM maal_contract
    WHERE CONTRACT_ID IN
    (
        SELECT contract_id FROM maal_contract
        NATURAL JOIN maal_artist
        Where maal_artist.artist_id = maal_delete_artist_proc.aid
    );
    DELETE FROM maal_artist
    WHERE artist_id = maal_delete_artist_proc.aid;
COMMIT;
END;
```

Album Sales and Average Revenue Procedure

The following procedure is used to get the total amount of revenue generated from an album transaction purchased by a major retailer distributor. This procedure also retrieves the average transaction revenue.

```
CREATE OR REPLACE PROCEDURE MAAL_Get_Sales_Average
IS

--DECLARE
revenue number(20);
total   number(20)    :=0 ;
counts   number(10);
average number(20,4);

Cursor cur1 is
   select transaction_id, album_id, batch_units
   from maal_sold_through;

   v_ST_transid             maal_SOLD_THROUGH.transaction_id%type;
   v_ST_albumid              maal_SOLD_THROUGH.album_id%type;
   v_ST_batch               maal_SOLD_THROUGH.batch_units%type;
   v_a_unit                maal_ALBUM.unit_price%type;

Cursor cur2 is
   select album_id, unit_price
   from maal_album;

   v_T_album           maal_album.album_id%type;
   v_T_unit_price      maal_album.unit_price%type;

BEGIN
   select count(transaction_id) into counts from MAAL_sold_through;
   dbms_output.put_line( RPAD('TransID',15,' ') || RPAD('albumID',15,' ') || RPAD('Batch',15,' ')
|| RPAD('Revenue',15,' '));
   dbms_output.put_line('-----------------------------------------------------------------');
   OPEN cur1;
   LOOP
      fetch cur1 into v_ST_transid, v_ST_albumid, v_ST_batch;
      exit when cur1%NOTFOUND;
         OPEN cur2;
         LOOP
```

```
            fetch cur2 into v_T_album, v_T_unit_price;
            exit when cur2%NOTFOUND;
            if v_ST_albumid = v_T_album then
               revenue := (v_T_unit_price * v_ST_batch);
            end if;
         End LOOP;
         CLOSE cur2;
      dbms_output.put_line( RPAD(v_ST_transid,15, ' ') || RPAD(v_ST_albumid,15, ' ') ||
RPAD(v_ST_batch,15,' ') || RPAD(revenue,15,' '));
      total := (total + revenue);
      --dbms_output.put_line( RPAD('Total Revenue', 15, ' ') || RPAD(total,15,' '));
   END LOOP;
   CLOSE cur1;
   dbms_output.put_line('---------------------------------------------------------------------');
   average := (total / counts);
   dbms_output.put_line( RPAD('Average Revenue',20, ' ') || RPAD(average,15,' '));
end;
```

Insert Contract Procedure

The following procedure is used to insert a new Contract into the Contract relation.

```
CREATE OR REPLACE PROCEDURE maal_insert_contract
(
        CON_ID IN VARCHAR2,
        ALTERMS IN NUMBER,
        SDATE IN DATE,
        EDATE IN DATE
)
AS
BEGIN
   INSERT INTO maal_contract
   VALUES
   (
        CON_ID,
        ALTERMS,
        SDATE,
        EDATE
   );
END maal_insert_contract;
```

4.3.2 Triggers

Album ID Update Trigger

The following trigger is execute when you attempt to update the primary key Album_ID. The new value will be updated everywhere else it is referenced as a foreign key.

```
CREATE OR REPLACE TRIGGER maal_alupdate
AFTER UPDATE OR INSERT ON maal_album
FOR EACH ROW
BEGIN
        UPDATE maal_song set maal_song.ALBUM_ID = :NEW.album_id
        WHERE maal_song.album_id = :OLD.album_id;
        UPDATE maal_sold_through set ALBUM_ID = :NEW.album_id
        WHERE maal_sold_through.album_id = :OLD.album_id;
END;
```

Album Info View Update Trigger

The following trigger is used when you attempt to update data on the Album_Info view in the database. Instead of updating the view, this trigger instead uses the new values to insert into the base tables that the view uses to draw data from.

```
CREATE OR REPLACE TRIGGER maal_update_album_info
INSTEAD OF UPDATE ON maal_album_info
FOR EACH ROW
BEGIN
        UPDATE maal_album
        SET album_id=:NEW.album_id, album_name=:NEW.album_name, date_released=
      :NEW.date_released
        WHERE album_id = :OLD.album_id;
        UPDATE maal_song
        SET track_number=:NEW.track_number, song_name=:NEW.song_name
        WHERE album_id = :OLD.album_id;
COMMIT;
END;
```

Writer ID Update Trigger

This trigger is used when you attempt to update a writer_id, which is the primary key of the Writer relation. This trigger will update this value everywhere it is referenced as a foreign key in the database.

```
CREATE OR REPLACE TRIGGER maal_writer_update
AFTER UPDATE ON maal_writer
FOR EACH ROW
BEGIN
        UPDATE maal_song set maal_song.writer_id = :NEW.writer_id
        WHERE maal_song.writer_id = :OLD.writer_id;
END;
```

Delete Transaction Triggers

The following triggers are used together in order to do three things. First, upon attempting to delete a Transaction record, all Sold_Through records associated with that specific Transaction will also be deleted. After this has occurred, the second two triggers will be executed, which will back up this deleted data into two log tables created specifically for this operation.

```
CREATE OR REPLACE TRIGGER maal_trans_delete
BEFORE DELETE ON maal_transaction
FOR EACH ROW
BEGIN
  DELETE FROM maal_sold_through
  WHERE maal_sold_through.transaction_id = :OLD.transaction_id;
END;

CREATE OR REPLACE TRIGGER maal_trans_delete_log
AFTER DELETE ON maal_transaction
FOR EACH ROW
BEGIN
  INSERT INTO maal_transaction_log
  VALUES (:OLD.transaction_id, :OLD.buyer_id, :OLD.tdate,sysdate );
```

```
END;

CREATE OR REPLACE TRIGGER maal_sold_Through_delete_log
AFTER DELETE ON maal_sold_through
FOR EACH ROW
BEGIN
   INSERT INTO maal_sold_through_log
  VALUES (:OLD.TRANSACTION_ID, :OLD.ALBUM_ID, :OLD.batch_units, sysdate);
END;
```

Phase 5: Graphical User Interface

5.1 User Groups

5.1.1 Executive Assistants

Executive assistants group will use this database at any location in order to gather information on new artists that have been contracted out, as well as the amount of terms they have been assigned to. They gather this information for higher-up executives as well as for financial purposes.

Daily Activities:

The daily activities of executive assistants are as follows:

- Retrieve contract information for artists currently working for company.
- Retrieve album information, especially cost to make album
- Assign or change studios for albums being recorded.
- Search for information needed for executives, such as financial information, and overall information on the hiring of artists by contract.

View

The views executive assistants would need to access are as follows:

- ARTIST_CONTRACT INFO
- CURRENT_CONTRACT REPORT
- MEMBER_GEN_INFO

5.1.2 Artists

The artist group will use this database in order to keep informed with their current albums and track listings, as well as their current studio that they will be working at. The group will have the ability to view and add/update members.

Daily Activities:

The daily activities for this

- View current albums that they have released or are currently working on
- The number of albums left under their current contract with the company.

Views

The views that Artist would need to access are as follows:

- ARTIST_CONTRACT_INFO
- ALBUM_SONG

5.2 GUI Design and Development in Java

Both of us have had a little experience with Java prior to us starting the database project. This allowed us to ease into the developer tool called NetBeans, which is an IDE that can be used for application development. We began by looking into internet resources as well as online tutorials for help on how to develop a simple user interface. Once we were able to get a few examples down, we began to work out a layout that we would use for the database GUI. The steps and we took are as follows:

1. We brainstormed ideas on what the user interface should allow the user to do. We then drew several diagrams on paper and did quick implementations in NetBeans to see the practicality of those designs and ideas. We thought of a few ideas on how to display the information and how the user could access the information.

2. Once we found a layout that we both liked, we moved forward and began to implement the final design in NetBeans.

3. We designed two reports for the GUI. One report is designed to print Album information, and the other is designed for printing Contract information.

We learned that even though we had an IDE to help us navigate our GUI, it would not always turn out the way we had hoped. There were many times where a GUI design would look fine during development, but when we would test the application, text fields and labels would start to move in different places.

Pulling data from a table and display it through a GUI was a main goal, and then display information in different areas based off of their relations. We learned that if our prepared

statements did not currently match our data model, our queries would fail and leave us with no information.

There are many components that can be added to the database, but for the purpose of this project, we wanted to limit the amount information for now. This can later be fixed by adding more.

5.3 Major Features

The below code is used to connect to our database, and add values to the current selected table. We use the prepared statements to insert or update values as they are passed to text fields in the GUI

5.3.1 Connecting to the Oracle Database

This code allows us to establish a connection to the CSUB Delphi database. We first create a Connection class and specify the credentials needed to connect. An instance of this class is used in the main Menu class and the function getDBConnection() is called to connect to the database before every SQL statement execution.

```
public class OracleConnect {

    Connection con = null;

    public OracleConnect() {
    }

    public Connection getDBConnection() {
        try {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            Connection con = DriverManager.getConnection("jdbc:oracle:thin:@delphi.cs.csubak.edu:1521:dbs01", "cs342", "c3m4p2s");
            return con;
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.toString());
            return null;
        }
    }

}
```

5.3.2 Inserting into Database

The following code is an example of an insertion into the Contract and Artist tables. This code is executed when the Save button is clicked on the Contract Information panel. This code pulls text from all the appropriate text fields and uses a Prepared Statement to set up a SQL Insert statement to insert a new Contract and Artist into the database.

```
con = OracleCon.getDBConnection();
    String contract_id = contractIDField.getText();
    String artist_id = artistIDField.getText();
    String artist_name = artistNameField.getText();
    String genre = genreField.getText();
    int album_terms = Integer.parseInt(albumTermsField.getText());
    String start_date = startDateField.getText();
    start_date = convertDateFormat(start_date);
    String end_date = endDateField.getText();
    if(end_date.charAt(0)==' ')
    end_date = "";
    else
    end_date = convertDateFormat(end_date);
    String insert = "Insert into maal_contract values(?,?,?,?)";
    pst = con.prepareStatement(insert);
    pst.setString(1, contract_id);
    pst.setInt(2, album_terms);
    pst.setString(3, start_date);
    pst.setString(4, end_date);
    pst.executeUpdate();

    insert = "Insert into maal_artist values (?,?,?,?)";
    pst = con.prepareStatement(insert);
    pst.setString(1, artist_id);
    pst.setString(2, contract_id);
    pst.setString(3, artist_name);
    pst.setString(4, genre);
    pst.executeUpdate();
```

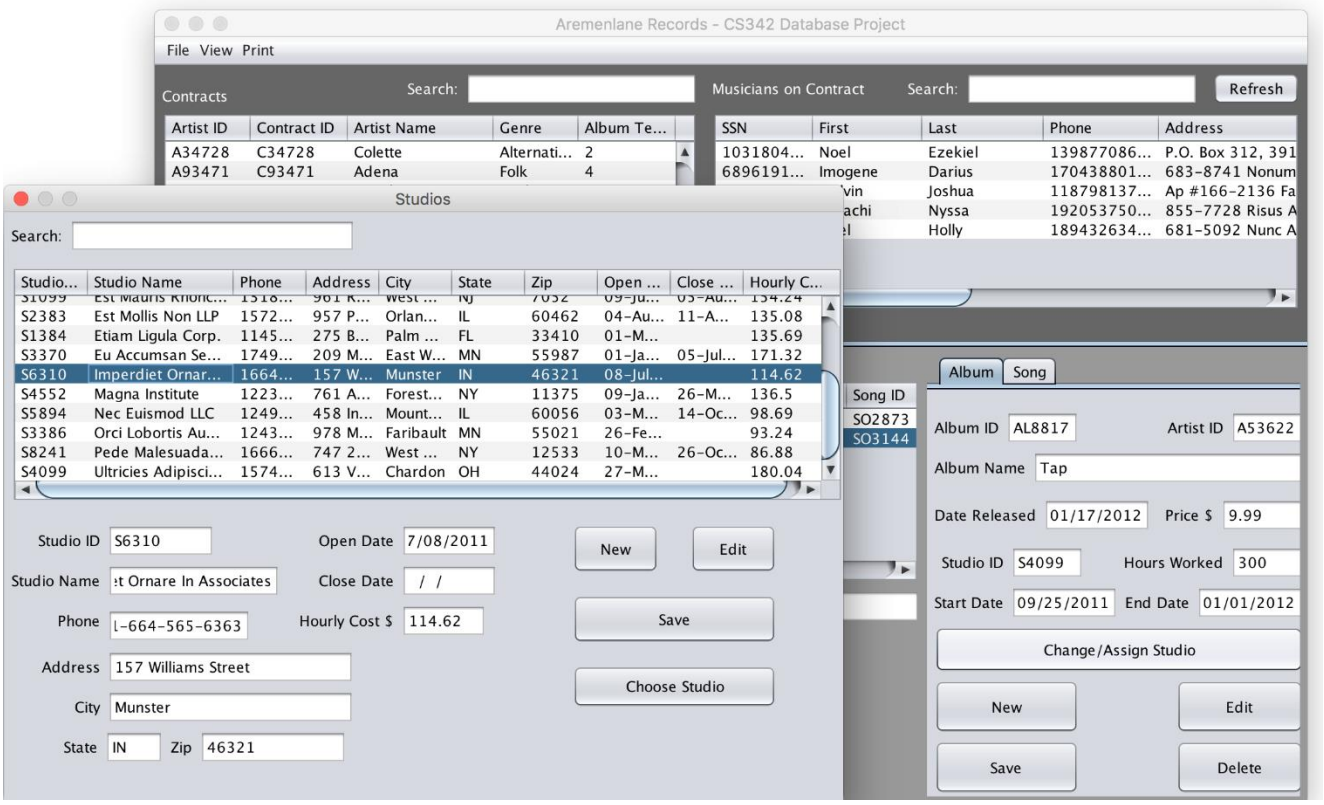
5.3.3 Row Filtering For Searching

The below code is used to filter table results in the table. When a user wants to search for a specific attribute, they can use the search text field to narrow down information. The data is filtered by each letter entered. For example, the letter "A" will filter fields that contain the letter "A". By typing "Ab", the search will filter fields that contain the combination "Ab" and so on.

```
private void contractSearchFieldKeyPressed(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    String target = contractSearchField.getText();
    TableRowSorter<TableModel> sorter = new TableRowSorter<TableModel>(artistTable.getModel());
    artistTable.setRowSorter(sorter);
    sorter.setRowFilter(RowFilter.regexFilter(".*"+target+".*"));
}
```

5.3.4 Assigning Studios to Albums

A major feature of our GUI is the ability to assign a Studio to an Album. Using a JDialog form, when a user is entering in Album information, you have the ability to click on "Change/Assign Studio" which will open the JDialog form. In this JDialog form, you can add a new Studio if the Album was recorded at a Studio that does not currently exist in the database, or search for the already existing Studio and click "Choose Studio" to assign that Studio to the Album that is currently be entered or edited.



Through the use of static variables, whenever a JDialog is created, the JDialog is able to access and change data in the Menu frame, which allows for simple communication between the JDialog and the Menu frame, which allows for assigning Studios to Albums.

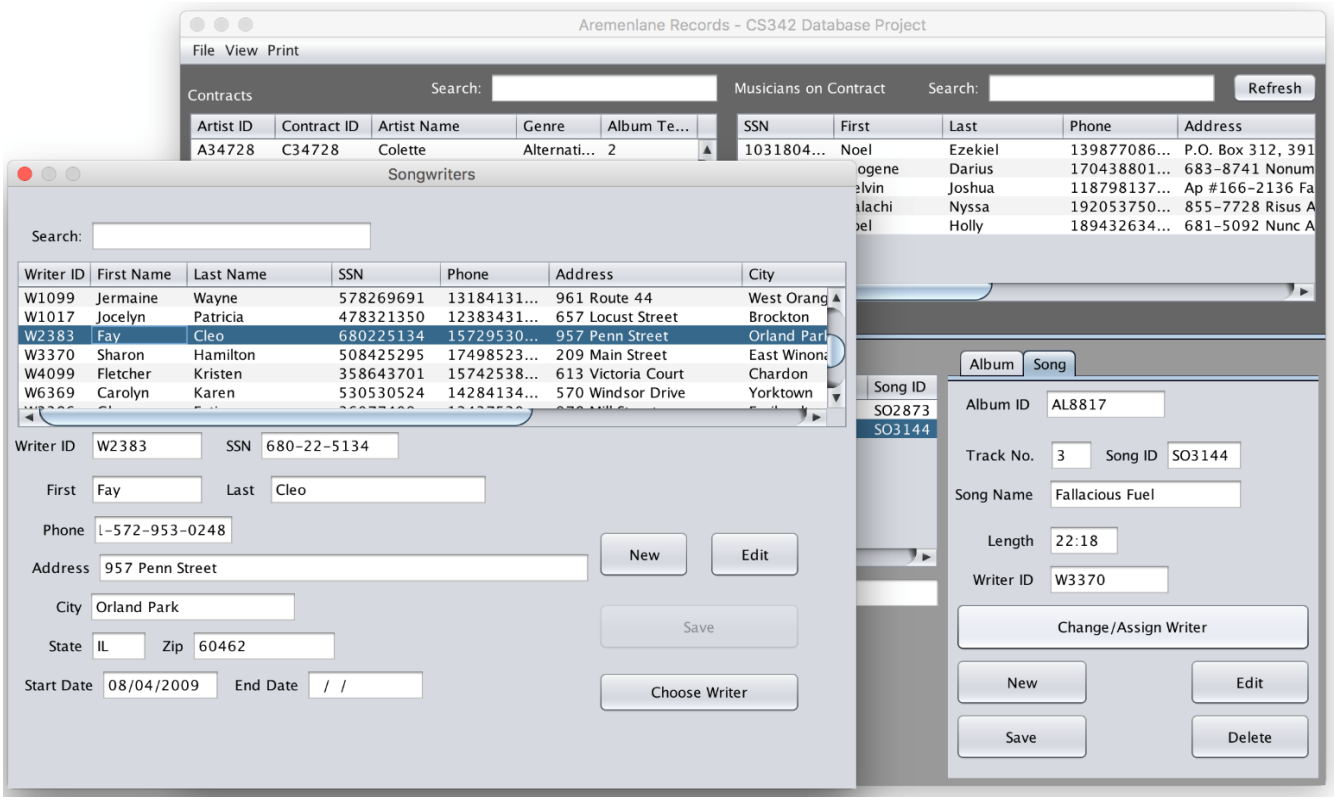
```
public static String studioChosen;
public static boolean studioWindow = false;
public static boolean writerWindow = false;
public static String writerChosen;

private void formWindowGainedFocus(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    if(studioWindow)
    {
        System.out.println(studioChosen);
        if(!(studioChosen.isEmpty() && !studioIDField.getText().isEmpty()))
        studioIDField.setText(studioChosen);
        studioWindow = false;
    }
    if(writerWindow)
    {
        System.out.println(writerChosen);
        if(!(writerChosen.isEmpty() && !writerIDField.getText().isEmpty()))
        writerIDField.setText(writerChosen);
        writerWindow = false;
    }
}
```

When focus is gained on the Menu frame, that means that the JDialog has completed its task, and a Studio has been chosen. So it will automatically pull the string data from the static variable that has been assigned a value in the JDialog, and it will use it to fill the proper JTextField.

5.3.5 Assigning Writers to Songs

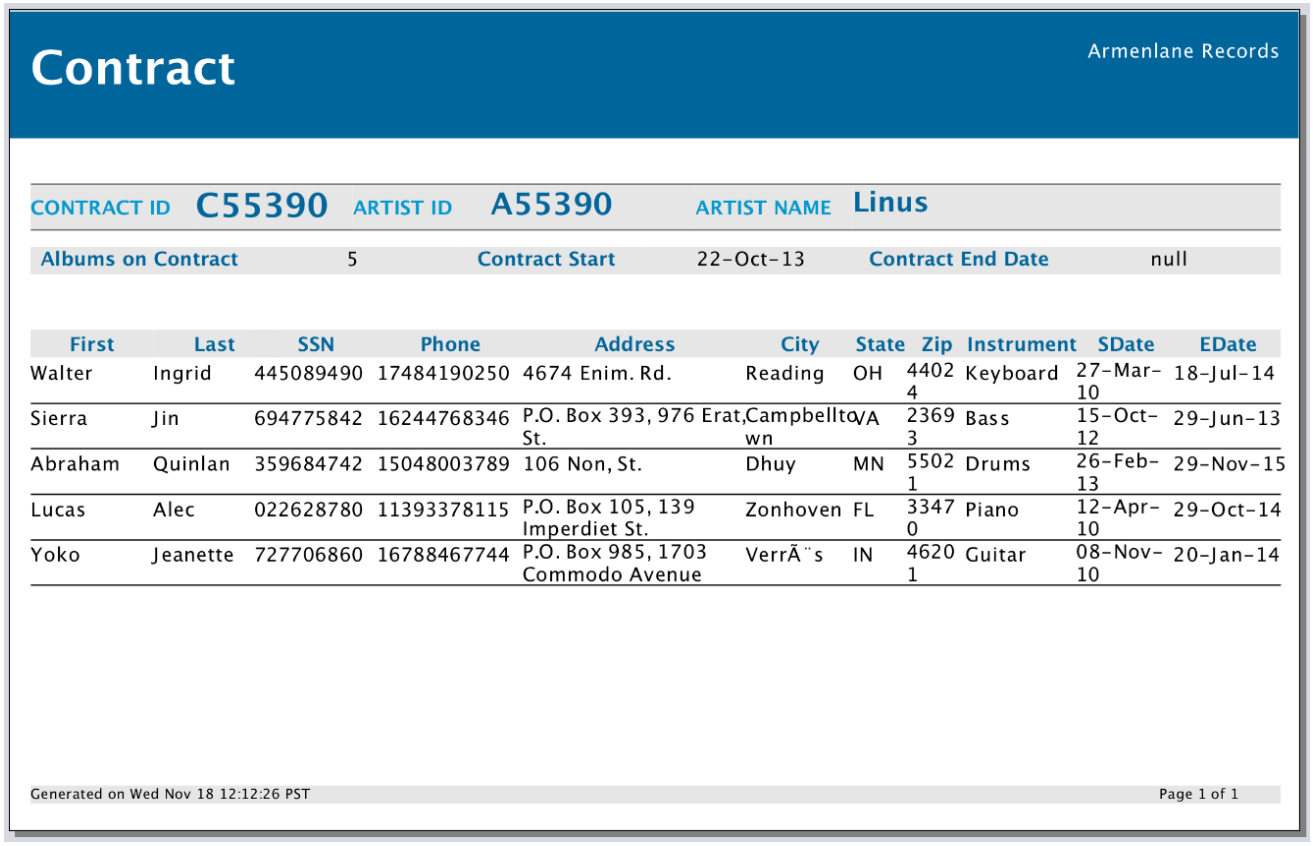
A major feature of our GUI is the ability to assign a Songwriter to a Song. Using a JDialog form, when a user is entering in Song information, you have the ability to click on "Change/Assign Writer" which will open the JDialog form. In this JDialog form, you can add a new Writer if the Song was written by a Songwriter that does not currently exist in the database, or search for the already existing Songwriter and click "Choose Writer" to assign that writer to the song that is currently be entered or edited.



This is implemented in the exact same way as when an Album is assigned a Studio. By using static varibles, the JDialog and the Menu frame are able to communicate and pass data between the two objects.

5.3.6 Generated Reports

A major feature of our GUI is the ability to generate reports. For a generated report, the user has the option of either selecting a single artist and displaying the contract information related to that artist, or they can print all available contracts under the company. This information is displayed in a separate window for readability. Our reports are generated using a plugin for NetBeans called Jasper reports. The image below is a contract report generated for a single contract and artist.



Contract

Armenlane Records

CONTRACT ID **C55390** ARTIST ID **A55390** ARTIST NAME **Linus**

Albums on Contract 5 **Contract Start** 22-Oct-13 **Contract End Date** null

First	Last	SSN	Phone	Address	City	State	Zip	Instrument	SDate	EDate
Walter	Ingrid	445089490	17484190250	4674 Enim. Rd.	Reading	OH	44024	Keyboard	27-Mar-10	18-Jul-14
Sierra	Jin	694775842	16244768346	P.O. Box 393, 976 Erat, St.	Campbellton	VA	23693	Bass	15-Oct-12	29-Jun-13
Abraham	Quinlan	359684742	15048003789	106 Non, St.	Dhuy	MN	55021	Drums	26-Feb-13	29-Nov-15
Lucas	Alec	022628780	11393378115	P.O. Box 105, 139 Imperdiet St.	Zonhoven	FL	33470	Piano	12-Apr-10	29-Oct-14
Yoko	Jeanette	727706860	16788467744	P.O. Box 985, 1703 Commodo Avenue	VerrÃ¨s	IN	46201	Guitar	08-Nov-10	20-Jan-14

Generated on Wed Nov 18 12:12:26 PST Page 1 of 1

Jasper Report utilizes HashMaps in order to take in parameters to fill in the XML reports based on a specified query. So for the Contract report, the Contract ID of the selected artist is passed in to the report when it is generated. This allows the report to be dynamic and change based on the users selection.

```
/*PRINT SELECTED CONTRACT INFORMATION*/
private void printCurrentContractButtonActionPerformed(java.awt.event.ActionEvent evt) {

    int temp = artistTable.getSelectedRow();
    if (temp >= 0) {
        try {
            if(memberCount>0){
                con = OracleCon.getDBConnection();

                String user = contractIDField.getText();
                HashMap map = new HashMap();
                map.put("CID", user);

                JasperPrint jp = JasperFillManager.fillReport(getClass().getResourceAsStream("currentContractReport.jasper"), map, con);
                JasperViewer.viewReport(jp, false);
                con.close();
            }
            if(memberCount==0)
            {
                /*If band has no members, then print different report*/
                con = OracleCon.getDBConnection();

                String user = contractIDField.getText();
                HashMap map = new HashMap();
                map.put("CID", user);

                JasperPrint jp = JasperFillManager.fillReport(getClass().getResourceAsStream("currentContractReportNoMember.jasper"), map, con);
                JasperViewer.viewReport(jp, false);
                con.close();
            }
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```

5.4 Outcome

In conclusion, we learned that if we had an issue that could not immediately be solved, outside sources or communication with each other is needed. We also learned that collaborating with a partner allows for a better design. Communication with each other is key when designing and operating a database.

From a technical aspect, we learned how databases are designed and implemented in great detail. We were able to explore databases from initial design, to implementation and GUI development. This covered several large topics, and we were able to get experience in all of them.

Outcome	Members' Outcome Answers btw 1 & 10
(3b) An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution.	Member 1: 10 Member 2: 10
(3e) An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs. An ability to understand the analysis, design, and implementation of a computerized solution to a real-life problem.	Member 1: 10 Member 2: 10
(3f) An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software specification white paper or a user manual.	Member 1: 10 Member 2: 10
(3j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.	Member 1: 10 Member 2: 10