Art Gallery Management System

Sophia Hall, Grace Leverett, Emily Tibbens

December 5, 2021

Background

Art galleries can host up to thousands of pieces of visual art. Each gallery has information about each piece of artwork, including artist details such as their name, address, and age. As well as information about the artwork itself like artist, year made, title, type, and price. Art galleries can also host art shows that potential customers can attend to view the various pieces of artwork to purchase or rent. Once the piece is sold or rented out, the art gallery also keeps information about the buyer or renter such as name, address, total amount spent, etc. Keeping track of all of this information can get very complicated. We are proposing a database to model an art gallery. The database will allow galleries to track and organize their collected data so that they can better understand what pieces will sell best in their gallery, and who they may be able to market the piece to. It will keep track of art shows that have taken place at the gallery, as well as various artworks that have been sold or rented through the gallery. This will help art galleries keep an updated record of all sales or rent agreements that have taken place. The database will also store information about potential customers, such as styles, mediums, and artists they prefer. This will allow the gallery to market specific pieces to customers who are likely to buy them. The database is essentially aimed at providing a cohesive model of all information an art gallery needs to be successful.

Data Description

We created our data because we could not find consistent data for each of the entities in our database. Creating our own assured us that our queries would be successful, and while it was more work upfront, we do not have to clean and inspect a database we would have found online. Below is an example of some of the tuples created for the Artwork table.

title	year	style	medium	asking_price	artist_id	collector_id
Disturbance	2018	Abstract	Ink on Paper		98452	100423
Hollowed Dr	2020	Modern	Graphite on Paper		98320	100424
Lonely Subm	2015	Impressionis	Oil Paint		82304	100425
Approval of F	2019	Modern	Ink on Paper		76182	100426
Duty of Crim	2021	Contemporai	Acrylic		56329	100427
Repulsive En	2021	Pop Art	Chalk		17840	100428
Divine Freed	2020	Abstract	Graphite on Paper		42940	100429
Dramatic Dir	2013	Impressionis	Oil Paint		90732	100430
Mirrors of De	2017	Contemporai	Ink on Paper		80374	100431
Discovery of	2020	Modern	Pastels		98234	100432

Functional Dependencies

These dependencies are for the relational schemas in the following section.

Artwork

Artwork's attributes are a list of information about the piece of artwork. These attributes are all functional independent of one another, so the only functional dependency is the primary key `{title, artist_id}` implying all of the other attributes.

- `title, artist_id → {title, year, style, type, medium, asking_price, artist_id, collector_id}`
- This is in BCNF because `{title`, `artist_id}` is a superkey

Artist

Artist's attributes are a list of personal information about the artist and the type of artwork they create. These attributes are all functional independent of one another, so the only functional dependency is the primary key `artist_id` implying all of the other attributes.

- `artist_id → {artist_id, first_name, … art_type} (all attributes of artist)`
- This is in BCNF because `artist_id` is a superkey

Customer

Customer's attributes are a list of personal information about the customer and the type of artwork they like. These attributes are all functional independent of one another, so the only functional dependency is the primary key `customer_id` implying all of the other attributes.

- `customer_id → {customer_id, first_name, … artist_id} (all attributes of customer)`
- This is in BCNF because `customer_id` is a superkey

Collector

Collector's attributes are a list of personal information about the collector and the type of artwork they collect. These attributes are all functional independent of one another, so the only functional dependency is the primary key `collector_id` implying all of the other attributes.

- `collector_id → {collector_id, first_name, … artist_id} (all attributes of collector)`
- This is in BCNF because `collector_id` is a superkey

Artshow

Artshow has the added constraint that no two art shows can have the same name, as well as the constraint that more than one art show cannot occur at the same address on the same day. This we have the functional dependencies

- `show_name → {show_name, start_date, … time} (all attributes of artshow)`
- `start_date, end_date, address → {show_name, start_date, … time} (all attributes of artshow)`
- This is in BCNF because `{show_name}` and `{start_date, end_date, address}` are superkeys

Rent

Rent's attributes are a list of important information about a rent agreement. These attributes are all functional independent of one another, so the only functional dependency is the primary key `invoice_num` implying all of the other attributes.

- `invoice_num → {invoice_num, start_date, return_date, duration, rent_fee, artist_percentage,renter_id}`
- This is in BCNF because `invoice_num` is a superkey

Sale

Sale's attributes are a list of important information about a sale contract. These attributes are all functional independent of one another, so the only functional dependency is the primary key `invoice_num` implying all of the other attributes.

- `invoice_num → {sale_date, invoice_num, sale_price, artist_percentage, buyer_id}`
- This is in BCNF because `invoice_num` is a superkey

Renter

Renter's attributes are a list of personal information about the renter. These attributes are all functional independent of one another, so the only functional dependency is the primary key `renter_id` implying all of the other attributes.

- `renter_id → {renter_id, first_name, last_name, street_num, street_name, city, state, zip_code, phone_num, num_rents}`
- This is in BCNF because `renter_id` is a superkey

Buyer

Buyer's attributes are a list of personal information about the buyer. These attributes are all functional independent of one another, so the only functional dependency is the primary key `buyer_id` implying all of the other attributes.

- `buyer_id → {buyer_id, first_name, last_name, street_num, street_name, city, state, zip_code, phone_num, num_purchases}`
- This is in BCNF because `buyer_id` is a superkey

Displayed_in, Sold_in, Rented_in

These relations all have three attributes in either a many-to-many or a one-to-one relationship. For displayed_in, many artworks can be displayed in the same art show, and many art shows can display the same artworks. For sold_in, only one artwork can be associated with a single sale contract. The same goes for rented_in; only one artwork can participate in a single rent agreement. Therefore, since no single attribute is a super key, none of these relations have non-trivial functional dependencies and are all in BCNF.

Relational Schemas

```
ArtWork      (title            CHAR(100),
              year             YEAR,
              style            CHAR(50),
              type             CHAR(50),
              medium           CHAR(50),
              asking_price     DECIMAL(8,2),
              artist_id        INT,
```

```
                collector_id,     INT,
                PRIMARY KEY(title, artist_id),
                FOREIGN KEY(artist_id REFERENCES Artist),
                FOREIGN KEY(collector_id REFERENCES Collector))
```

This table holds information about artwork at the gallery. The Artwork entity in the ER diagram is a weak entity that is many-to-one with the Artist entity through the relation Creates. So, it has been merged into this relation by adding the foriegn key "artist_id" as a primary key. The Artwork entity also has a many-to-one relationship with the owns relation in the ER diagram, so it has been merged into this relation by adding the foreign key "collector_id".

```
Artist       (artist_id         INT,
              first_name        CHAR(25),
              last_name         CHAR(25),
              street_num        INT,
              street_name        CHAR(50),
              city               CHAR(50),
              state              CHAR(50),
              zip_code           INT,
              age                INT,
              art_medium         CHAR(50),
              art_style         CHAR(50),
              art_type           CHAR(50),
              PRIMARY KEY(artist_id))
```

This entity represents an artist that can create artwork. The attributes art_medium, art_type, and art_style are the medium used to create the artwork (i.e. watercolor paints), the type of artwork (i.e. a painting), and the style of artwork (i.e. modern art).

```
Customer     (customer_id       INT,
              first_name        CHAR(25),
              last_name         CHAR(25),
              street_num        INT,
              street_name        CHAR(50),
              city               CHAR(50),
              state              CHAR(50),
              zip_code           INT,
              preferred_style   CHAR(50),
              preferred_medium  CHAR(50),
              phone_num         VARCHAR(25),
              artist_id          INT,
              PRIMARY KEY(customer_id),
              FOREIGN KEY(artist_id REFERENCES Artist))
```

This table holds information about potential customers that visit an art gallery. The Likes relation in the ER diagram is many to one (many customers can like one artist, but one customer cannot like more than one artist) so it has been merged into this relation by adding the artist_id field.

```
Collector    (first_name        CHAR(25),
              last_name         CHAR(25),
              collector_id       INT,
              Collection_type    CHAR(50),
              collection_style  CHAR(50),
              collection_medium CHAR(50),
              street_num        INT,
              street_name       CHAR(50),
              city              CHAR(50),
              state             CHAR(50),
              zip_code          INT,
              phone_num         VARCHAR(25),
              artist_id         INT,
              PRIMARY KEY(collector_id),
              FOREIGN KEY(artist_id REFERENCES Artist))
```

This table holds information about collectors that collect an artist's works. The collected_by relation in the ER diagram is many to one (many collectors can collect from one artist, but one collector cannot collect from more than one artist) so it has been merged into this relation by adding the artist_id field.

```
ArtShow      (show_name         CHAR(50),
              start_date        DATE,
              end_date           DATE,
              time              TIME,
              street_num        INT,
              street_name        CHAR(50),
              city               CHAR(50),
              state              CHAR(50),
              zip_code           INT,
              PRIMARY KEY(show_name))
```

This entity represents an art show that can take place at an art gallery and host artwork. The attribute show_name is the name of the art show. We are assuming that art show names are all unique, so this can be the primary key.

```
Rent         (invoice_num       INT,
              start_date        DATE,
              return_date       DATE,
              duration           INT,
              rent_fee           DECIMAL(8,2),
              artist_percentage  INT,
              renter_id          INT,
              PRIMARY KEY(invoice_num),
              FOREIGN KEY(renter_id REFERENCES Renter))
```

This table holds information about Rent contracts when an artwork is rented to a renter. The rented_to relation in the ER diagram is a many-to-one (one renter can have many rent contracts but one rent contract cannot belong to more than one renter) so it has been merged into this relation by adding the renter_id field. There is total participation on the Rent side because a renter cannot have a rent without a rent agreement, and a rent agreement cannot exist without a renter.

```
Sale        (sale_date        DATE,
             invoice_num      INT,
             sale_price       DECIMAL(8,2),
             artist_percentage INT,
             buyer_id         INT,
             PRIMARY KEY(invoice_num),
             FOREIGN KEY(buyer_id REFERENCES Buyer))
```

This table holds information about Sale contracts when an artwork is sold to a buyer. The sold_to relation in the ER diagram is a many-to-one (one buyer can have many sale contracts but one sale contract cannot belong to more than one buyer) so it has been merged into this relation by adding the buyer_id field. There is total participation on the Sale and the Buyer side because a buyer cannot be sold to without a sale agreement, and a sale agreement cannot exist without a buyer.

```
Renter(renter_id              INT,
             first_name       CHAR(25),
             last_name        CHAR(25),
             street_num       INT,
             street_name       CHAR(50),
             city              CHAR(50),
             state             CHAR(50),
             zip_code          INT,
             phone_num        VARCHAR(25),
             num_rents        INT,
             PRIMARY KEY(renter_id))
```

This entity represents a person who is renting artwork. The attribute “num_rents” keeps track of the number of times each renter has rented artwork.

```
Buyer  (buyer_id              INT,
             first_name       CHAR(25),
             last_name        CHAR(25),
             street_num       INT,
             street_name       CHAR(50),
             city              CHAR(50),
             state             CHAR(50),
             zip_code          INT,
             phone_num        VARCHAR(25),
             num_purchases    INT,
             PRIMARY KEY(buyer_id))
```

This entity represents a person who is buying artwork. The attribute “num_purchases” keeps track of the number of times each buyer has bought artwork.

```
Displayed_in(show_name          CHAR(50),
             title              CHAR(100),
             artist_id          CHAR(50),
             PRIMARY KEY(show_name, title, artist_id),
             FOREIGN KEY(show_name REFERENCES ArtShow),
             FOREIGN KEY(title, artist_id REFERENCES ArtWork))
```

This relation keeps track of which artworks are displayed in an art show. Each artwork can be displayed in multiple shows, and each show can have many artworks displayed at it. Therefore, this relationship is many-to-many. There is total participation on the ArtShow side because artwork cannot be displayed without an art show.

```
Rented_in(invoice_num           INT,
             title              CHAR(100),
             artist_id           INT,
             PRIMARY KEY(invoice_num, title, artist_id),
             FOREIGN KEY(invoice_num REFERENCES Rent),
             FOREIGN KEY(title, artist_id REFERENCES ArtWork))
```

This relation stores invoice numbers that are given to a piece of artwork when it is rented out. Each artwork can be a part of one rent interaction, and each rent interaction can only be for one piece of artwork. Therefore the relationship is one-to-one. There is total participation on the Rent side because an artwork can not be rented without a rent contract.

```
Sold_in      (invoice_num       INT,
             title              CHAR(100),
             artist_id           INT,
             PRIMARY KEY(title, artist_id, invoice_num),
             FOREIGN KEY(invoice_num REFERENCES Sale),
             FOREIGN KEY(title, artist_id) REFERENCES ArtWork))
```

This relation stores invoice numbers that are given to a piece of artwork when it is sold. Each artwork can be a part of one sale interaction, and each sale interaction can only be for one piece of artwork. We are assuming that once an artwork is sold to a buyer, it will not be sold again. Therefore the relationship is one-to-one. There is total participation on the Sale side because an artwork can not be sold without a sale contract.

The Likes relation in the ER diagram has been merged into Customer since each customer can like at most one artist (the Customer to Artist relationship is many-to-one). Note that Customer does not have total participation in Likes, so the artist_id field in Customer can take null values.

The Collected_By relation in the ER diagram has been merged into Collector since each collector can collect at most one artist (the Collector to Artist relationship is many-to-one). Note that Collector does not have total participation in Collected_By, so the artist_id field in Collector can take null values.

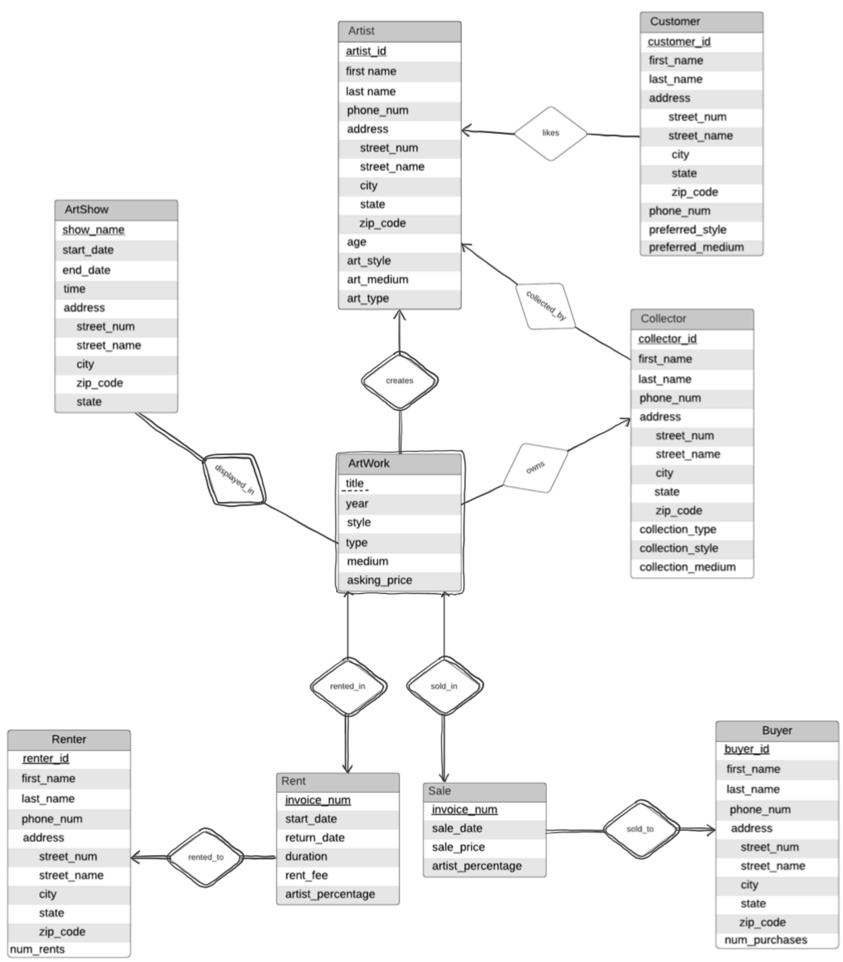
The Creates relation in the ER diagram has been merged into Artwork since Artwork is a weak entity, and each artwork can be created by at most one artist (the Artwork to Artist relationship is many-to-one). Note that Artwork has total participation in Creates, so the artist_id field in Artwork cannot be null.

The Owns relation in the ER diagram has been merged into Artwork since each Artwork can be owned by at most one collector (the Artwork to Collector relationship is many-to-one). Note that Artwork does not have total participation in Owns, so the collector_id in Artwork can take null values.

The Rented_To relation in the ER diagram has been merged into Rent since each rent agreement can be for at most one renter (the Rent to Renter relationship is many-to-one). Note that Rent does have total participation in rented_to, so the renter_id field in Rent cannot take null values.

The Sold_To relation in the ER diagram has been merged into Sale since each sale agreement can be for at most one buyer (the Sale to Buyer relationship is many-to-one). Note that Sale does have total participation in sold_to, so the buyer_id field in Sale cannot take null values.

ER Diagram



Example Queries

1. It is important for art galleries to know which shows are successful. The more successful an individual show is, the more money they make, and the more galleries can improve future events. There can be many measures of success. For this query, we will be finding the list of cities that have sold more than 10 pieces.
   - **SQL:**

```
SELECT a.show_name, a.city, COUNT(aw.title)
FROM Artwork aw, ArtShow a, Displayed_in d
WHERE aw.artist_id = d.artist_id
     AND a.show_name = d.show_name
     AND d.title = aw.title
GROUP BY a.show_name, a.city
```

   - **Relational Algebra:**

$\Pi_{show.name,\, city}(ArtShow)(\sigma_{titleCount>10}(_{title}\varsigma_{Count(title)\ as\ titleCount}(ArtWork)))$

   - **Query Results:**

```
SELECT a.show_name, a.city, COUNT(aw.title)
FROM Artwork aw, Artshow a, Displayed_in d
WHERE aw.artist_id = d.artist_id
    AND a.show_name = d.show_name
    AND d.title = aw.title
GROUP BY a.show_name, a.city
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

show_name	city	COUNT(aw.title)
Relational Imagination	Chicago	1
Remixing Sustainability	Cleveland	1
After the Chemistry	Los Angeles	1
Alchemical Dreams	New York	1
An Overwhelming Dreams	New York	1
Fantastic Chemistry	New York	1
The Politics of Relevance	San Francisco	1
After the Sustainability	Chicago	1
Romancing Media	Los Angeles	1
After the Dissent	Seattle	1
Parsing Charm	Los Angeles	1

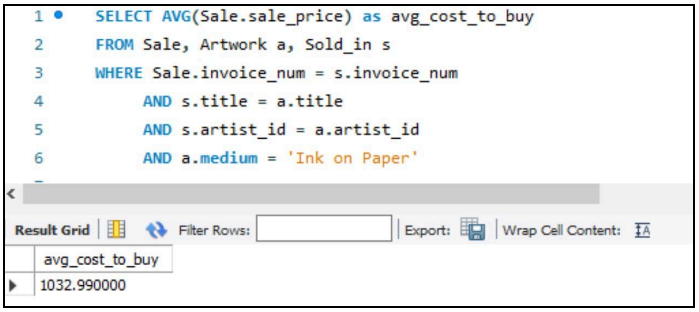
2. Valuing pieces of art can be difficult. The artist may overvalue a piece they put a lot of work into, and a gallery may undervalue a piece so they can sell it quickly. An artist asks the art gallery to help price one of their art pieces. The piece is of ink on paper medium. The art gallery will query the data to find the average cost of pieces with ink on paper medium.
   - SQL:

```
SELECT AVG(Sale.sale_price) as avg_cost_to_buy
FROM Sale, Artwork a, Sold_in s
WHERE Sale.invoice_num = s.invoice_num
      AND s.title = a.title
      AND s.artist_id = a.artist_id
      AND a.medium = "ink on paper"
```

   - **Relational Algebra:**

$\Pi_{AVG(sale.price)}(Sale \bowtie Sold.\,in) \bowtie (\sigma_{ArtWork.medium\,=\,"ink\ on\ paper"}(ArtWork)$

- **Query Results:**



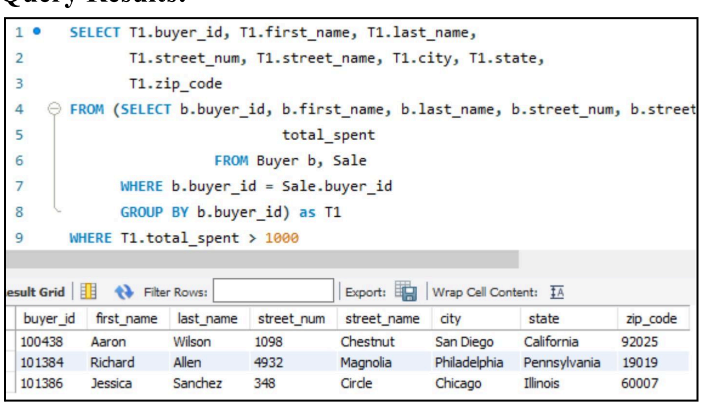
3. Some art galleries will host a preview or open house before the initial opening. They invite prospective buyers to presale some of the work. In order for this to be successful, due to the limited space available, they only want to send invites to the buyers that are most likely to purchase a piece of art at the event. In this query we are finding the buyers that have spent more than $1000 at the gallery in the past 5 years, and their address to send the invitation to.
   - **SQL:**

```
SELECT T1.buyer_id, T1.first_name, T1.last_name,
       T1.street_num, T1.street_name, T1.city, T1.state,
       T1.zip_code
FROM (SELECT b.buyer_id, b.first_name, b.last_name,
b.street_num, b.street_name, b.city, b.state, b.zip_code,
SUM(Sale.sale_price) as total_spent
      FROM Buyer b, Sale
      WHERE b.buyer_id = Sale.buyer_id
      GROUP BY b.buyer_id) as T1
WHERE T1.total_spent > 1000
```

   - **Relational Algebra:**

$$\Pi_{id,\, first.name,\, last.name,\, street.num,\, street.name,\, city,\, state,\, zip.code}(Buyer)(\sigma_{SUM(Sale.sale.price) > 1000}(Sale) \wedge (Buyer \bowtie_{Buyer.buyer.id=Sale.buyer.id} Sale))$$

   - **Query Results:**



buyer_id	first_name	last_name	street_num	street_name	city	state	zip_code
100438	Aaron	Wilson	1098	Chestnut	San Diego	California	92025
101384	Richard	Allen	4932	Magnolia	Philadelphia	Pennsylvania	19019
101386	Jessica	Sanchez	348	Circle	Chicago	Illinois	60007

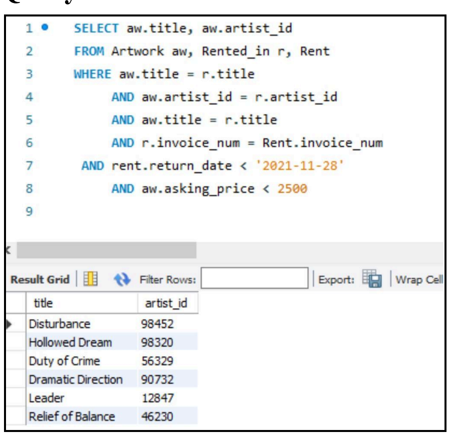
4. There are many components to what makes an art gallery successful. One of these components is renting. Designers will often call galleries with a certain price range for a client, and the style they are looking for in hopes curators can help select a piece. To help the curators find pieces for designers to select to rent, we will query the data to find artworks under $2500 that are currently available to rent.
   - **SQL:**

```
SELECT aw.title, aw.artist_id
FROM Artwork aw, Rented_in r, Rent
WHERE aw.title = r.title
      AND aw.artist_id = r.artist_id
      AND aw.title = r.title
      AND r.invoice_num = Rent.invoice_num
      AND rent.return_date < GETDATE()
      AND aw.asking_price < 2500
```

   - **Relational Algebra:**

$$\Pi_{title,\, artist.id}(\sigma_{ArtWork.asking.price\ <\ 2500}(ArtWork \bowtie Rented.in) \bowtie (\sigma_{Rent.return.date\ <\ 12/5/21}(Rent))$$

   - **Query Results:**



```
SELECT aw.title, aw.artist_id
FROM Artwork aw, Rented_in r, Rent
WHERE aw.title = r.title
     AND aw.artist_id = r.artist_id
     AND aw.title = r.title
     AND r.invoice_num = Rent.invoice_num
 AND rent.return_date < '2021-11-28'
     AND aw.asking_price < 2500
```

Result Grid | Filter Rows: | Export: | Wrap Cell

title	artist_id
Disturbance	98452
Hollowed Dream	98320
Duty of Crime	56329
Dramatic Direction	90732
Leader	12847
Relief of Balance	46230

5. Customer service is extremely important to this industry. With the end of the calendar year quickly approaching, curators want to make sure they call and thank their newest customers in hopes that they will continue to buy pieces in the future. This query will find the buyer name and phone number of buyers who have **only** bought pieces in 2021.
   - **SQL:**

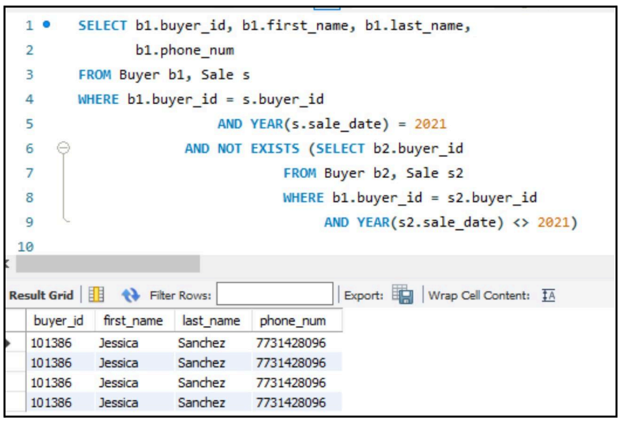
```
SELECT b1.buyer_id, b1.first_name, b1.last_name,
       b1.phone_num
FROM Buyer b1, Sale s
WHERE b1.buyer_id = s.buyer_id
      AND YEAR(s.sale_date) == 2021
```

```
AND NOT EXISTS (SELECT b2.buyer_id
                FROM Buyer b2, Sale s2
                WHERE b1.buyer_id = s2.buyer_id
                      AND YEAR(s2.sale_date) <> 2021)
```

- **Relational Algebra:**

$$\Pi_{buyer.id,\, first.name,\, last.name,\, phone.num}(\sigma_{\ sale.date\ =2021} Buyer \bowtie\ Sale)$$
$$/\Pi_{buyer.id,\, first.name,\, last.name,\, phone.num}(\sigma_{\ sale.date\ <>2021} Buyer \bowtie\ Sale)$$

- **Query Results:**



```
SELECT b1.buyer_id, b1.first_name, b1.last_name,
       b1.phone_num
FROM Buyer b1, Sale s
WHERE b1.buyer_id = s.buyer_id
                  AND YEAR(s.sale_date) = 2021
              AND NOT EXISTS (SELECT b2.buyer_id
                              FROM Buyer b2, Sale s2
                              WHERE b1.buyer_id = s2.buyer_id
                                    AND YEAR(s2.sale_date) <> 2021)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

buyer_id	first_name	last_name	phone_num
101386	Jessica	Sanchez	7731428096
101386	Jessica	Sanchez	7731428096
101386	Jessica	Sanchez	7731428096
101386	Jessica	Sanchez	7731428096

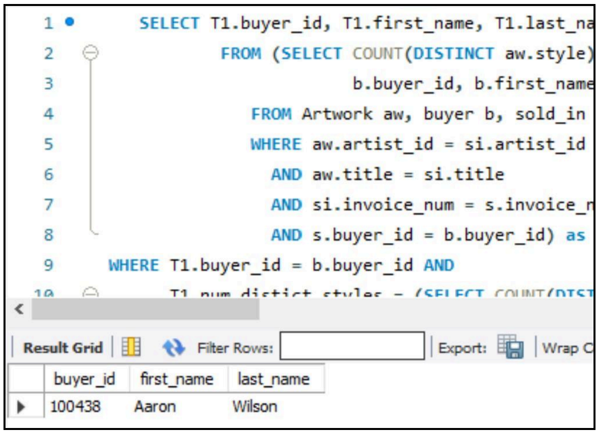
6. Curators are trying to find a better way to potentially identify designers as they typically purchase more than the average buyer. We will query the data to find the names and ids of buyers who have purchased at least one of every style.
   - **SQL:**

```
SELECT T1.buyer_id, T1.first_name, T1.last_name
FROM (SELECT COUNT(DISTINCT aw.style) as num_distict_styles,
                    b.buyer_id, b.first_name, b.last_name
      FROM Artwork aw, buyer b, sold_in si, sale s
      WHERE aw.artist_id = si.artist_id
           AND aw.title = si.title
           AND si.invoice_num = s.invoice_num
           AND s.buyer_id = b.buyer_id) as T1, buyer b
WHERE T1.buyer_id = b.buyer_id AND
    T1.num_distict_styles = (SELECT COUNT(DISTINCT artwork.style)
                             FROM Artwork)
```

- **Relational Algebra:**

$$temp1 \rightarrow (\ \Pi_{p\ distinct.styles(count(style))}(Artwork \bowtie\ Sold.in)) \bowtie\ \ (\Pi_{buyer.id,\, first.name,\, last.name}(Buyer \bowtie Sale))$$
$$temp2 \rightarrow (\ \Pi_{p\ num.styles(count(style))}(Artwork\ ))$$
$$\Pi_{buyer.id,\, first.name,\, last.name}(\sigma_{\ temp1.buyer.id\, =\, buyer.buyer.id\ \ and\ \ temp1.distinct.styles\, =\, temp2.num.styles}(temp1 \bowtie\ Buyer)\ )$$

- **Query Results:**



Note: With the way our database is set up, all of the queries that we came up with to gather information that would be relevant to an art gallery use aggregate functions or special SQL functions. Aggregate functions and SQL functions are not supported in tuple relational calculus, so we were not able to write tuple relational calculus for any of our queries. Therefore, we came up with one additional query for the sake of demonstrating that we can write in tuple relational calculus.

7. Find the pairs of ids of buyers and renters who share the same name.
   - **SQL:**

```
SELECT b.buyer_id, r.renter_id
FROM Buyer b, Renter r
WHERE b.first_name=r.first_name AND
b.last_name=r.last_name
```

   - **Relational Algebra:**

$$\Pi_{buyer.id,\ renter.id}(Buyer \bowtie_{Buyer.first.name\, =\, Renter.first.name\, \wedge\, Buyer.last.name = Renter.last.name} Renter)$$

   - **Tuple Relational Calculus:**

$$\{t^{(2)} | (\exists b)(\exists r)\,(Buyer(b)\ \wedge\ Renter(r)\ \wedge\ t[1] = b[buyer.id]\ \wedge\ t[2] = r[renter.id]\ \wedge\ b[first.name]\ =\ r[first.name]\ \wedge\ b[last.name]\ =\ r[last.name]\,\}$$

Implementation

We used MySQL as our DBMS for the backend of the application. The MySQL workbench allowed us to create and test our queries easily. Our front end web app was created using Django and python web framework. The front end was written in html and css, and integrated with Django to handle website routing and backend functionality. Our web app and MySQL server are run locally.

Team Contributions

Sophia

- Created ER model and ER diagram
- Went through relational schemas and combined redundant tables using foreign keys
- Wrote queries in SQL
- Wrote the code for the front end of the web app
- Worked on web app functionality/back end with Grace
- Worked on the Functional Dependency section with Emily
- Worked on the relational algebra with Emily

Grace:

- Created the DBMS in MySQL and Django
- Created the queries in MySQL and Django
- Wrote the basic code needed for the web app
- Created the rough draft of the relational schemas

Emily:

- Responsible for the background section
- Creating the data and writing the corresponding section
- Wrote the initial example queries in plain english
- Assisted in Functional Dependency section

Demo

Team Github repository: https://github.com/gleverett/ArtDatabaseSite
Demo given to TA Minh Pham on 12/3/21.

Takeaways

Getting more in depth practice of writing queries in the multiple ways we learned throughout the course was very beneficial. The added layer of writing these queries in SQL and allowing them to run was the largest takeaway from this project. This allowed us to not only double check that the queries we wrote were correct, but let us visualize the outcomes that we are predicting when doing the theoretical practice problems. Using Django and python to create a working web app was above and beyond what we learned in this course. We were able to put together the components we learned in this course and other CS courses to implement our database and create a functional front end user interface.