Red Fox Art Gallery

Relational Database Design

Sydney George

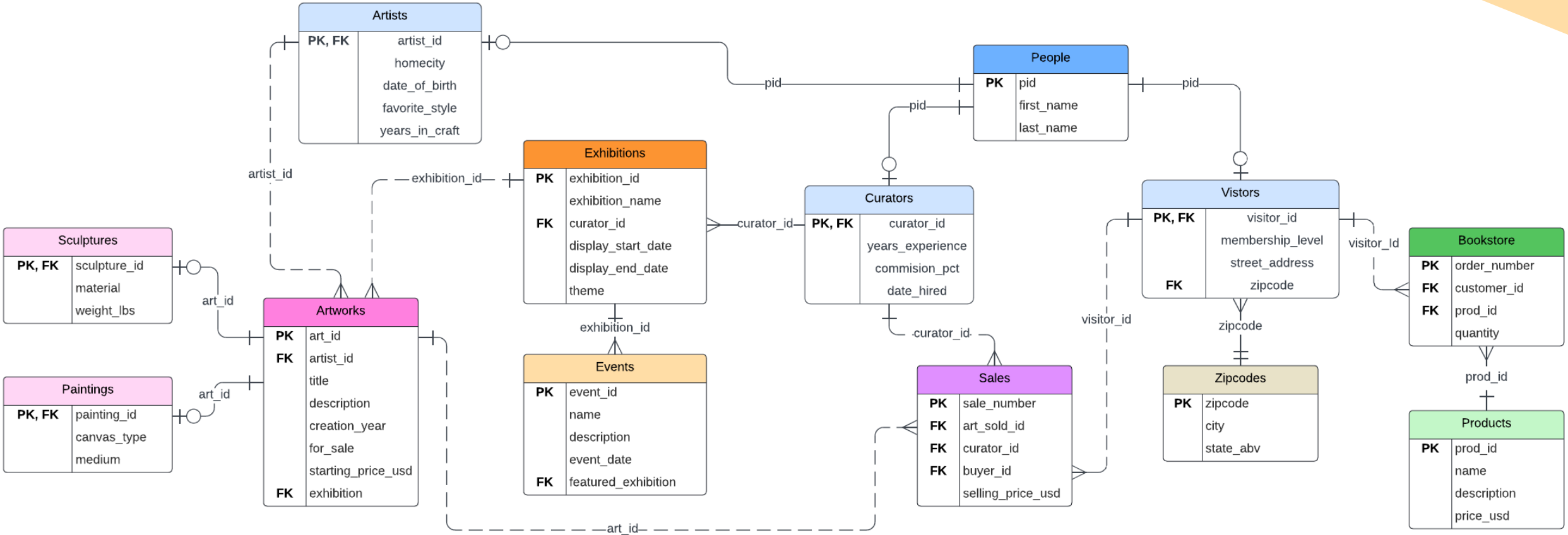
Table of Contents

** Implementation Notes throughout

Executive Summary

- The Red Fox Art Gallery is steadily growing its collection of artwork and presence in the community through frequent events. Keeping track of the artwork, sales, as well as the plethora of exhibitions and events is important in order to sustain the success of this beloved gallery!
- Hence, the purpose of the Red Fox Art Gallery database is to provide a way to keep all of this information organized.
- This presentation will begin with an ER diagram to outline a database design that will fulfill all the needs of the art gallery. Following this are the create statements for those tables and sample outputs for what the database would look like. Additionally, views, reports, stored procedures, and triggers that the art gallery may find useful, and security for how each user role may use the database.

E/R Diagram


Artists
PK, FK
artist_id
homecity
date_of_birth
favorite_style
years_in_craft
People
PK
pid
first_name
last_name
pid
pid
pid
artist_id
Exhibitions
PK
exhibition_id
exhibition_name
FK
curator_id
display_start_date
display_end_date
theme
exhibition_id
exhibition_id
Curators
PK, FK
curator_id
years_experience
commision_pct
date_hired
curator_id
Vistors
PK, FK
visitor_id
membership_level
street_address
FK
zipcode
visitor_Id
Bookstore
PK
order_number
FK
customer_id
FK
prod_id
quantity
Sculptures
PK, FK
sculpture_id
material
weight_lbs
art_id
Artworks
PK
art_id
FK
artist_id
title
description
creation_year
for_sale
starting_price_usd
FK
exhibition
Paintings
PK, FK
painting_id
canvas_type
medium
art_id
Events
PK
event_id
name
description
event_date
FK
featured_exhibition
curator_id
Sales
PK
sale_number
FK
art_sold_id
FK
curator_id
FK
buyer_id
selling_price_usd
visitor_id
zipcode
Zipcodes
PK
zipcode
city
state_abv
prod_id
Products
PK
prod_id
name
description
price_usd
art_id

People Table

The People table contains an id for each person as well as their first and last name. This table is a place for common information between Artists, Curators, and Visitors.

```
CREATE TABLE People (
  pid                 char(4) not null unique,
  first_name          text not null,
  last_name           text not null,
 primary key(pid)
);
```

<u>Functional Dependencies</u>

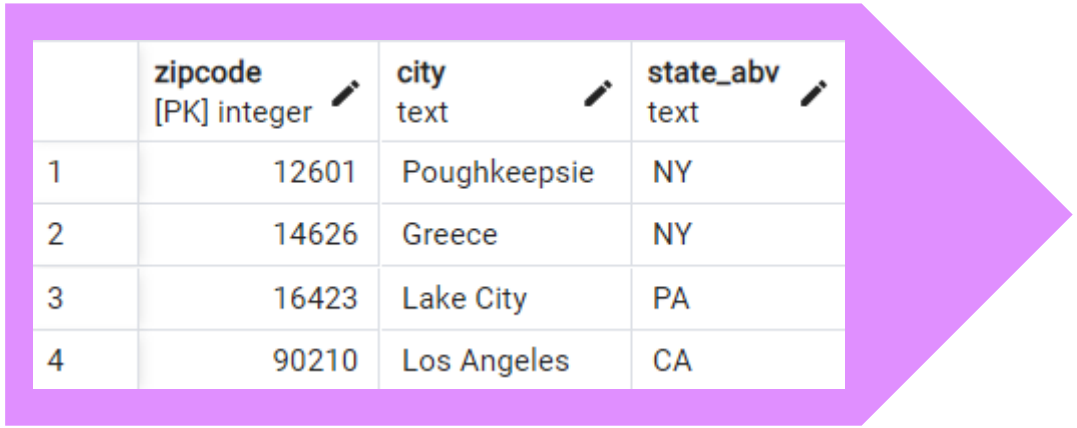
pid → first_name, last_name

	pid [PK] character	first_name text	last_name text
1	p001	Alan	Labouseur
2	p002	Tommy	Newman
3	p003	Sydney	George
4	p004	Ryann	Lambert
5	p005	Julia	Reid
6	p006	Oliver	Banks
7	p007	Jeff	Koons
8	p008	Cecily	Brown
9	p009	Damien	Hirst

Zipcodes Table

The Zipcodes table contains the city and state corresponding to each unique zipcode.

```
CREATE TABLE Zipcodes (
  zipcode                    int not null unique,
  city                       text not null,
  state_abv                  text not null,
 primary key(zipcode)
);
```



	zipcode [PK] integer	city text	state_abv text
1	12601	Poughkeepsie	NY
2	14626	Greece	NY
3	16423	Lake City	PA
4	90210	Los Angeles	CA

Functional Dependencies

zipcode → city, state_abv

Visitors Table

The Visitors table contains information that only pertains to visitors such as membership level, address, and zipcode. This information is needed to send visitors mail, appropriate flyers or event invitations, ship purchases to, etc.

	visitor_id [PK] character	membership_level text	street_address text	zipcode integer
1	p001	Gold	123 Main Street	12601
2	p002	Gold	3399 North Road	12601
3	p003	Platinum	157 August Drive	14626
4	p004	Bronze	301 New Court	16423
5	p005	Platinum	80 East Hills Lane	90210

```
CREATE TABLE Visitors (
  visitor_id            char(4) not null references People(pid),
  membership_level      text,
  street_address        text,
  zipcode               int references Zipcodes(zipcode),
 primary key(visitor_id)
);
```

Functional Dependencies

Visitor_id → membership_level, street_address, zipcode

Products Table

The Products table contains information about the products offered in the art gallery bookstore, such as the name, description, and price. Who doesn't want to take a souvenir home from their art gallery trip?

```
CREATE TABLE Products (
  prod_id                 char(4) not null unique,
  name                    text not null,
  description             text,
  price_usd               decimal(4,2) not null,
 primary key(prod_id)
);
```

Functional Dependencies

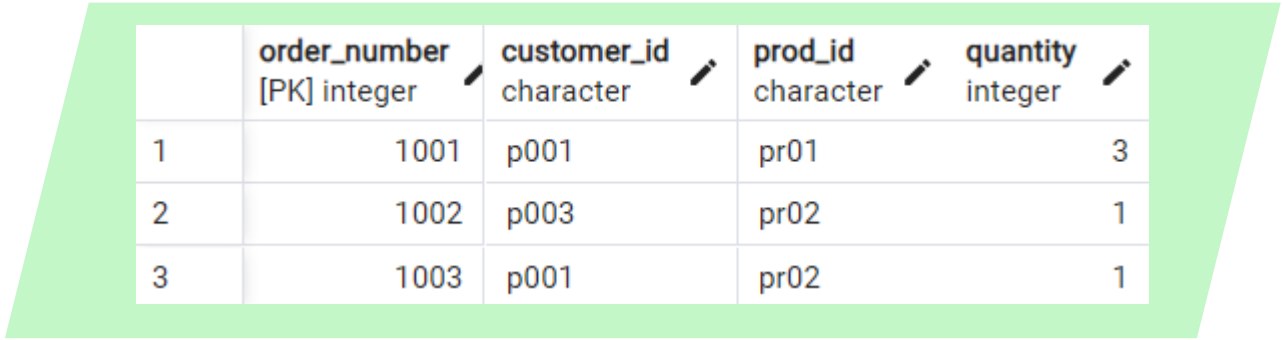
prod_id → name, description, price_usd

	prod_id [PK] character	name text	description text	price_usd numeric (4,2)
1	pr01	Magnet	Sqaure magent with logo	4.99
2	pr02	Magazine	Pictures and descriptions of all artworks	15.99
3	pr03	T-shirt	Shirt with art gallery logo	21.99

Bookstore Table

The Bookstore table contains records of all the sales made in the art gallery bookstore. Each purchase has an order number, a customer id, and the quantity and product that they bought.

```
CREATE TABLE Bookstore (
  order_number          int not null unique,
  customer_id           char(4) not null references Visitors(visitor_id),
  prod_id               char(4) not null references Products(prod_id),
  quantity              int not null,
 primary key(order_number)
);
```



	order_number [PK] integer	customer_id character	prod_id character	quantity integer
1	1001	p001	pr01	3
2	1002	p003	pr02	1
3	1003	p001	pr02	1

Functional Dependencies

Order_number → customer_id, prod_id, quantity

Artists Table

The Artists table contains specific information relevant to artists, such as home city, date of birth, the artist's favorite style to work in, and their years as an artist.

	artist_id [PK] character	homecity text	date_of_birth date	favorite_style text	years_in_craft integer
1	p008	New York City	1980-12-17	modernism	20
2	p009	Atlanta	1970-06-18	abtract	42

```
CREATE TABLE Artists (
  artist_id          char(4) not null references People(pid),
  homecity           text not null,
  date_of_birth      date not null,
  favorite_style     text,
  years_in_craft     int,
 primary key(artist_id)
);
```

Functional Dependencies

Artist_id→ homecity, date_of_birth,
favorite_style,
years_in_craft

Curators Table

The Curators table contains specific information relevant to curators. This is years of experience, the commission percent they earn, and the date they were hired. These are the most important fields to track because they will be useful in following reports.

	curator_id [PK] character	years_experience text	commision_pct numeric (3,2)	date_hired date
1	p006	10	0.30	2018-02-15
2	p007	4	0.12	2020-04-25

```
CREATE TABLE Curators (
  curator_id          char(4) not null references People(pid),
  years_experience     text,
  commision_pct       decimal(3,2) not null,
  date_hired           date not null,
 primary key(curator_id)
);
```

Functional Dependencies

curator_id → years_experience,
commission_pct,
date_hired

Exhibitions Table

Exhibitions are the different groups of paintings put together by curators throughout the gallery. This table contains an exhibition id, name, the curator who put the artworks together, an optional theme description, and a start/end dates for the exhibition duration.

```
CREATE TABLE Exhibitions (
  exhibition_id          char(4) not null unique,
  exhibition_name        text not null,
  curator_id              char(4) not null references Curators(curator_id),
  display_start_date     date not null,
  display_end_date       date not null,
  theme                  text,
 primary key(exhibition_id)
);
```

Functional Dependencies

exhibition_id→ exhibition_name,
curator_id,
display_start_date,
display_end_date,
theme

	exhibition_id [PK] character	exhibition_name text	curator_id character	display_start_date date	display_end_date date	theme text
1	ex01	Colors in Motion	p006	2020-04-10	2024-04-20	Colorful abstract perspectives
2	ex02	Emerging Echoes	p006	2024-01-01	2024-02-01	Showcasing new artists and additions to the galle...
3	ex03	Light and Shadow	p007	2022-05-12	2022-05-12	One day event exploring contrast

Events Table

The Events table contains information for the events the art gallery hosts. Events only last one day, often as a way of showcasing off an exhibition. Events have a unqiue id, name, description, a date, and an exhibition that the event is held for.

	event_id [PK] character	name text	description text	event_date date	featured_exhibition character
1	e010	New Years Opening Event	[null]	2024-01-05	ex02
2	e011	Light and Shadow Showcase	[null]	2022-05-12	ex03

```
CREATE TABLE Events (
  event_id                char(4) not null unique,
  name                    text not null,
  description              text,
  event_date               date not null,
  featured_exhibition      char(4) not null references Exhibitions(exhibition_id),
 primary key(event_id)
);
```

Functional Dependencies

event_id → name, description, event_date, featured_exhibition

Artwork Table

The Artwork table contains the id, title, and description for each piece of artwork in the gallery. It also includes the artist, when they made the piece, whether it is for sale (if so, the starting price), and the exhibition it was placed in.

	art_id [PK] character	artist_id character	title text	description text	creation_year integer	for_sale boolean	starting_price_usd numeric (7,2)	exhibition character
1	a001	p008	Purple	[null]	2020	false	[null]	ex01
2	a002	p008	Blue	[null]	2020	false	[null]	ex01
3	a003	p008	Green	[null]	2020	false	[null]	ex01
4	a004	p008	Orb	[null]	2020	true	10000.00	ex01
5	a005	p009	Brass	[null]	2023	true	75000.00	ex02
6	a006	p009	Steel	[null]	2014	true	5000.00	ex03

```
CREATE TABLE Artwork (
  art_id                char(4) not null unique,
  artist_id       char(4) not null references Artists(artist_id),
  title                 text not null,
  description           text,
  creation_year         int not null,
  for_sale        boolean not null,
  starting_price_usd  decimal(7,2),
  exhibition            char(4) references Exhibitions(exhibition_id),
 primary key(art_id)
);
```

<u>Functional Dependencies</u>

art_id→ artist_id, title, description, creation_year, for_sale, starting_price_usd, exhibition

Paintings Table

The Paintings table contains specific details for artworks that are paintings, such as the canvas type and medium used. These details only apply to paintings, hence why they are included here and not the Artwork table.

	painting_id [PK] character	canvas_type text	medium text
1	a001	linen	oil
2	a002	linen	oil
3	a003	wodd	acrylic

```
CREATE TABLE Paintings (
  painting_id          char(4) not null references Artwork(art_id),
  canvas_type          text not null,
  medium               text not null,
 primary key(painting_id)
);
```

Functional Dependencies

painting_id → canvas_type, medium

Sculptures Table

The Sculptures table contains specific details for artworks that are sculptures, such as the material and weight in pounds. Material must be known, but it is possible to not have the weight in pounds recorded

	sculpture_id [PK] character	material text	weight_lbs integer
1	a005	brass	400
2	a006	steel	50
3	a004	glass	23

```
CREATE TABLE Sculptures (
  sculpture_id          char(4) not null references Artwork(art_id),
  material              text not null,
  weight_lbs            int,
 primary key(sculpture_id)
);
```

Functional Dependencies

sculpture_id → material, weight_lbs

Sales Table

The Sales table contains information for artwork sales. Each sale has a unique sale number, and records the art that was sold, which curator sold it, the buyer, and the final price the painting was purchased for. Many paintings go under biding wars, so it is important to keep track of this separate from the starting price.

	sale_number [PK] integer	art_sold_id character	curator_id character	buyer_id character	selling_price_usd numeric (7,2)
1	1	a004	p006	p001	14500.00
2	2	a006	p007	p003	7999.99

```
CREATE TABLE Sales (
  sale_number          int not null unique,
  art_sold_id          char(4) not null references Artwork(art_id) unique,
  curator_id           char(4) not null references Curators(curator_id),
  buyer_id             char(4) not null references Visitors(visitor_id),
  selling_price_usd    decimal(7,2) not null,
 primary key(sale_number)
);
```

Functional Dependencies

sale_number → art_sold_id, curator_id, buyer_Id, selling_price_usd

Views

Some helpful views include combining people and curators with a join, as well as combining the artwork and sales table. The information is these tables go hand in hand, meaning a user may often want to reference data from both tables at once, so creating a view will simplify upcoming queries.

```
CREATE VIEW PeopleCurators
as
select *
from Curators c inner join People p on c.curator_id = p.pid;
```

	curator_id character	years_experience text	commision_pct numeric (3,2)	date_hired date	pid character	first_name text	last_name text
1	p006	10	0.30	2018-02-15	p006	Oliver	Banks
2	p007	4	0.12	2020-04-25	p007	Jeff	Koons

```
CREATE VIEW ArtworkSales
as
select *
from Artwork a inner join Sales s on a.art_id = s.art_sold_id;
```

	art_id character	artist_id character	title text	description text	creation_year integer	for_sale boolean	starting_price_usd numeric (7,2)	exhibition character	sale_number integer	art_sold_id character	curator_id character	buyer_id character	selling_price_usd numeric (7,2)
1	a004	p008	Orb	[null]	2020	true	10000.00	ex01	1	a004	p006	p001	14500.00
2	a006	p009	Steel	[null]	2014	true	5000.00	ex03	2	a006	p007	p003	7999.99

Views Continued

These views combine the artwork table with the painting and sculpture tables. This allows for a view of all the possible details for either paintings or sculptures in one place. While these will not be used in the following stored procedures or triggers, they serve as helpful tool for getting the full details of artworks easily and quickly.

	art_id character	artist_id character	title text	description text	creation_year integer	for_sale boolean	starting_price_usd numeric (7,2)	exhibition character	painting_id character	canvas_type text	medium text
1	a001	p008	Purple	[null]	2020	false	[null]	ex01	a001	linen	oil
2	a002	p008	Blue	[null]	2020	false	[null]	ex01	a002	linen	oil
3	a003	p008	Green	[null]	2020	false	[null]	ex01	a003	wodd	acrylic

```
CREATE VIEW SculptureArtwork
as
select *
from Artwork a inner join Sculptures s on a.art_id = s.sculpture_id;
```

	art_id character	artist_id character	title text	description text	creation_year integer	for_sale boolean	starting_price_usd numeric (7,2)	exhibition character	sculpture_id character	material text	weight_lbs integer
1	a005	p009	Brass	[null]	2023	true	75000.00	ex02	a005	brass	400
2	a006	p009	Steel	[null]	2014	true	5000.00	ex03	a006	steel	50
3	a004	p008	Orb	[null]	2020	true	10000.00	ex01	a004	glass	23

```
CREATE VIEW PaintingArtwork
as
select *
from Artwork a inner join Paintings p on a.art_id = p.painting_id;
```

Report 1:

This report show the curators who have made the most money by outputting a new field called commission_usd. It also outputs relevant fields such as the name of the curator and fields that may help explain this commission amount like years of experience.

```
select first_name, last_name, years_experience,
                    art_sold_id, (commision_pct *
                    selling_price_usd) as commission_usd
from PeopleCurators pc inner join Sales s on pc.curator_id = s.curator_id
order by commission_usd DESC;
```

	first_name text	last_name text	years_experience text	art_sold_id character	commission_usd numeric
1	Oliver	Banks	10	a004	4350.0000
2	Jeff	Koons	4	a006	959.9988

Report 2:

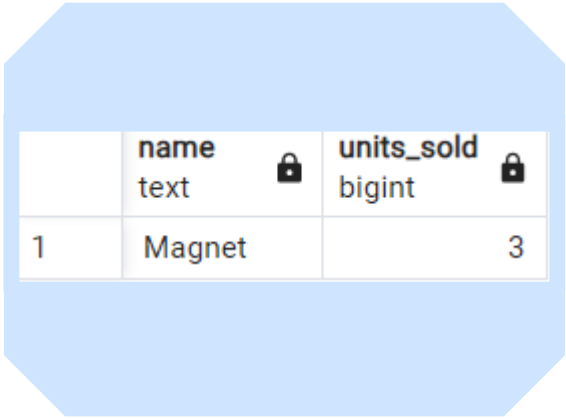
This report shows by how much over the starting price artworks sold for. Artworks are often bid on, so selling for more than anticipated is common. This report also outputs the artist, buyer, and curator that were involved with that artwork, which may lend to various insights. For example, does one buyer regularly buy over asking and by how much? Similarly, is there a popular artist whose paintings go over the starting price consistently?

```
select art_id, title, artist_id, buyer_id, curator_id,
                        (selling_price_usd - starting_price_usd)
as oversold_amt
from ArtworkSales
where starting_price_usd < selling_price_usd
order by oversold_amt DESC;
```

	art_id character	title text	artist_id character	buyer_id character	curator_id character	oversold_amt numeric
1	a004	Orb	p008	p001	p006	4500.00
2	a006	Steel	p009	p003	p007	2999.99

Report 3:

This report shows which bookstore product sells the most and how much. A lot of profit of the gallery comes from bookstore sales (as more visitors buy from the bookstore than pieces of artwork), so it's important to know which product we need to keep in stock a plenty!



	name text	units_sold bigint
1	Magnet	3

```
select p.name, sum(b.quantity) as units_sold
from Bookstore b inner join Products p on b.prod_id =
p.prod_id
group by p.prod_id
order by sum(b.quantity) DESC
LIMIT 1;
```

Stored Procedure 1:

```
create or replace function getArtworks_byId_orName(char(4), text, text, REFCURSOR) returns refcursor
as
$$
declare
  wanted_artist_id char(4)      := $1;
  first_name_lookup text                  := $2;
  last_name_lookup text     := $3;
  resultset   REFCURSOR := $4;
begin
 if (wanted_artist_id IS NOT NULL) then
                open resultset for
                select *
                                from Artwork
                                where  artist_id = wanted_artist_id;
else
                open resultset for
                                select *
                                from People p inner join Artwork a on p.pid = a.artist_id
                                where p.first_name = first_name_lookup
                                                and p.last_name = last_name_lookup;
                end if;
  return resultset;
end;
$$
language plpgsql;
```

This procedure gets all of the artworks for a certain artist. The artist can be specified by their first and last name or by their id. The purpose is to facilitate easy use of look up within the database.

	pid character	first_name text	last_name text	art_id character	artist_id character	title text	description text	creation_year integer	for_sale boolean	starting_price_usd numeric (7,2)	exhibition character
1	p008	Cecily	Brown	a001	p008	Purple	[null]	2020	false	[null]	ex01
2	p008	Cecily	Brown	a002	p008	Blue	[null]	2020	false	[null]	ex01
3	p008	Cecily	Brown	a003	p008	Green	[null]	2020	false	[null]	ex01
4	p008	Cecily	Brown	a004	p008	Orb	[null]	2020	true	10000.00	ex01

Testing

```
select getArtworks_byId_orName(NULL, ‘
                Cecily', 'Brown', 'results');
Fetch all from results;
```

Stored Procedure 2:

```
create or replace function getArtworks_inEvent(char(4),
REFCURSOR) returns refcursor as
$$
declare
  event_id_lookup   char(4)      := $1;
  resultset   REFCURSOR := $2;
begin
  open resultset for
    select a.art_id, a.title, a.description
                from Events e inner join  Exhibitions x on
e.featured_exhibition = x.exhibition_id
                                                    inner join
             Artwork a on a.exhibition = x.exhibition_id
              where e.event_id = event_id_lookup;
  return resultset;
end;
$$
language plpgsql;
```

This procedure returns all of the artworks being featured in an event using the event id. Events feature an exhibition, and exhibitions can have many artworks, so this is helpful to view all of the artworks quickly and easily.

Testing

```
select getArtworks_inEvent('e010', 'resultset');
Fetch all from resultset;
```

	art_id [PK] character	title text	description text
1	a005	Brass	[null]

Trigger 1:

```
create or replace function check_pname()
returns trigger as
$$
begin
        if (NEW.first_name IS NULL) then
                raise exception 'You must enter a first name';
        end if;
        if (NEW.last_name IS NULL) then
                raise exception 'You must enter a last name';
        end if;
        return new;
end
$$
language plpgsql;

create trigger check_pname
before insert or update on People
for each row
execute procedure check_pname();
```


A simple but important trigger is one that returns a descriptive error when no first or last name is given while trying to insert a new person into the People table. Using a trigger such as this is often more helpful than the default SQL error output.

Testing

```
insert into People
values ('p010', 'Dylan', NULL);
```

```
ERROR:  You must enter a last name
CONTEXT:  PL/pgSQL function check_pname() line 7 at RAISE
```

Trigger 2:

```
create or replace function quantity_limit()
returns trigger as
$$
begin
        if (NEW.quantity > 5) then
        delete from Bookstore where quantity =
NEW.quantity;
        end if;
        return new;
end;
$$ language plpgsql;

create trigger quantity_limit
after insert on Bookstore
for each row
execute procedure quantity_limit();
```

The Red Fox Art Gallery wants to make sure every visitor has the opportunity to take a souvenir of their visit home! So, they implemented a max quantity of 5 to any purchases made in the bookstore. Any time a quantity over the 5 limit is entered into the database it is cancelled (deleted).

Testing

```
insert into Bookstore
values (1004, 'p001', 'pr02', 6);

select *
from Bookstore;
```

	order_number [PK] integer	customer_id character	prod_id character	quantity integer
1	1001	p001	pr01	3
2	1002	p003	pr02	1
3	1003	p001	pr02	1

Security

Admin: The admin has all permissions and complete access to the database. This person would either be the owner of the art gallery, or a person reasonable for the database.

```
create role Admin;
grant all on
all tables
in schema public
to Admin;
```

Manager: The manager has access to as much of the database as they need. Their responsibilities include keeping track of artists, visitors, and such, so they have complete access for inserting and updating.

```
create role Manager;
grant select, insert, update
on all tables
in schema public
to Manager;
```

Curators: The curators have access to all the art related tables that are relevant to their job responsibilities.

```
create role Curator;
grant select, insert, update
on Artwork, Sculptures, Paintings, Exhibitions, Events, Sales
to Curator;
```

Associate: These employees work in the Bookstore and have the least amount of access to the database. They can create and manage sales.

```
create role Associate;
grant select, insert
on Bookstore
to Associate;
```

Known Problems/Future Enhancements

- Known problem: On the first stored procedure, there could potentially be problems if the name of the artist that is being searched for has the same name as a visitor. As in the people table, this is not differentiated.
- Future enhancements include:
  - Adding more fields if desired. This may include a credit card number or payment method.
  - Using joins to add more depth to queries and select people's or artist's names instead of ids
- Advanced triggers that would be helpful in this database are:
  - Not being able to sell an artwork that is already sold, changing the for sale status to sold when a sale is created
- Expanding for growth is also possible. For example, if the art gallery wants to start keeping track of the location of the artworks not just in terms of the exhibitions, but the locations of the exhibitions within the halls of the art gallery building.