



# FASHION MONTH DATABASE

**Designed by Shannon Maier**

---

# TABLE OF CONTENTS

**Executive Summary.....3**

**ER Diagram.....4**

**Tables.....5**

**Reports.....21**

**Views.....24**

**Stored Procedures.....27**

**Triggers.....29**

**Security .....31**

**Notes/Issues/Improvements.....32**

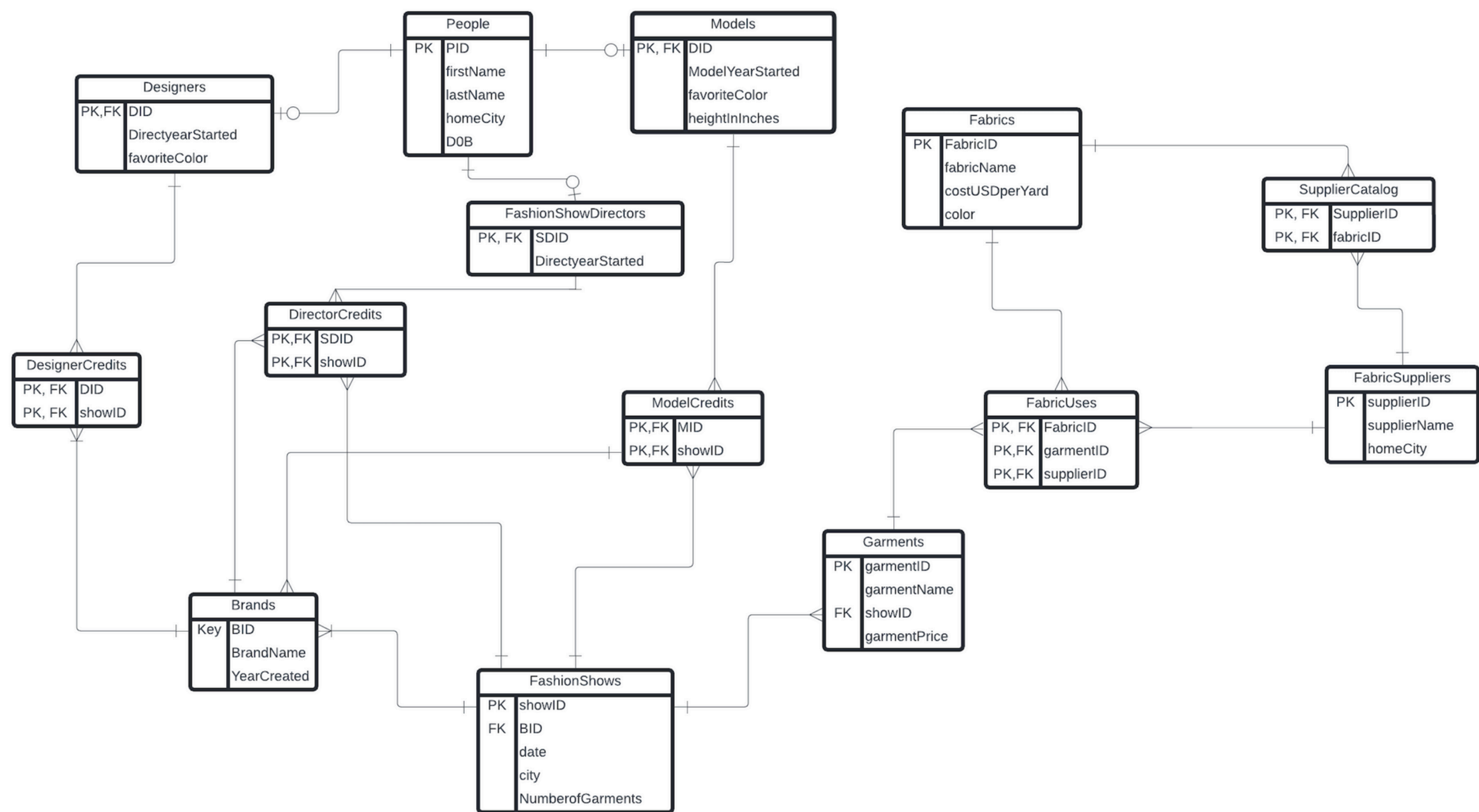
# EXECUTIVE SUMMARY -----

**This database was created in order to keep track of all the data behind Fashion Month. With thousands of people, brands, and businesses involved in this one month-long event each year, it only makes sense to have a database to organize all of it.**

**The following report provides vital information about the database in addition to various ways in which the database can be used. Some of the aspects that are reviewed include the ER diagram, create statements for the tables (with sample data for each table), reports and queries, views, stored procedures, triggers, and security roles.**

**The goal of this database is to help Fashion Month administrators and workers easily sort through the data regarding their jobs so that they can present the best possible Fashion Month each year.**

# ER DIAGRAM



The background features a large, light pink circle on the right side, partially overlapping a brown background. The brown background is divided into two horizontal sections: a lighter tan section on top and a darker brown section on the bottom.

# TABLES

# PEOPLE

This table hold all people in the database with their ID, name, home city, and DOB

```
CREATE TABLE People (  
  pid          int not null PRIMARY KEY,  
  firstName    text not null,  
  lastName     text not null,  
  homeCity     text,  
  DOB          date  
);
```

## Functional Dependencies:

- **People** → **firstName, lastName, homeCity, DOB**

	pid [PK] integer	firstname text	lastname text	homecity text	dob date
1	1	Shannon	Maier	New York	1970-05-04
2	2	Maggie	Lawrence	Paris	2003-05-02
3	3	Alan	Labouseur	Ontario	1968-07-17
4	4	Cindy	Crawford	Louisville	1969-08-18
5	5	Virgil	Abloh	London	1989-04-23
6	6	Cindy	Turlington	New York	1995-03-24
7	7	Denise	Smith	Los Angeles	1967-02-02
8	8	Colin	Burrow	Charleston	2000-01-17
9	9	Tucker	Angelo	London	2001-02-19
10	10	Kate	Sandler	Milan	1990-03-03

# DESIGNERS

This table holds all the designers with their IDs, the year they started designing, and their favorite colors. A designer needs to be a person.

```
CREATE TABLE Designers (  
  DID int not null PRIMARY KEY REFERENCES People(PID),  
  DesignyearStarted INT,  
  favoriteColor text  
);
```

	did [PK] integer	designyearstarted integer	favoritecolor text
1	1	1990	yellow
2	2	2023	blue
3	5	2010	blue
4	8	2020	orange
5	7	1995	pink



Functional Dependencies:

- DID → yearStarted, favoriteColor

# FashionShowDirectors-----

This table holds all the Fashion show directors with their IDs and the year they started. A director needs to be a person

```
CREATE TABLE FashionShowDirectors (  
  SDID int not null PRIMARY KEY REFERENCES People(PID),  
  DirectyearStarted int  
);
```

	sdid [PK] integer 	directyearstarted integer 
1	2	2020
2	3	1992
3	7	1989
4	10	2015

**Functional Dependencies:**

- SDID →yearStarted







# MODELS

This table holds all the Models with the year they started, their favorite color, and their height. A model needs to be a person.

```
CREATE TABLE Models (  
  MID int not null PRIMARY KEY REFERENCES People(PID),  
  ModelyearStarted int,  
  favoriteColor text,  
  heightInInches int  
);
```

### Functional Dependencies:

- MID → yearStarted, favoriteColor, heightInInches.

	mid [PK] integer 	modelyearstarted integer 	favoritecolor text 	heightinches integer 
1	1	1990	yellow	70
2	4	1987	blue	71
3	6	2010	blue	71
4	8	2020	orange	76
5	9	2021	pink	75

# BRANDS

This table holds all the brands with their IDs, names, and the year were created

```
CREATE TABLE Brands (  
  BID int not null PRIMARY KEY,  
  BrandName text not null,  
  YearCreated int  
);
```

	bid [PK] integer	brandname text	yearcreated integer
1	1000	Fendi	1985
2	1001	Louis Vuitton	1950
3	1002	Staud	2005
4	1003	Dior	1953
5	1004	Moschino	1999
6	1005	Coach	1974

## Functional Dependencies:

- BID → BrandName,  
YearCreated

# FashionShows

This table holds each of the shows with their ID, brand, date, city and Number of Garments in each show

```
CREATE TABLE FashionShows (  
  showID int not null PRIMARY KEY,  
  BID int REFERENCES Brands(BID),  
  date DATE ,  
  city text ,  
  NumberOfGarments int  
  
  CONSTRAINT garments_non_negative check (NumberOfGarments >= 0)  
);
```

## Functional Dependencies:

- showID → BID, date, city, NumberOfGarments

	showid [PK] integer	bid integer	date date	city text	numberofgarments integer
1	3001	1000	2010-09-10	New York	48
2	3002	1001	2020-09-29	Milan	30
3	3003	1002	2023-09-25	Paris	33
4	3004	1005	2010-09-16	New York	32
5	3005	1004	2021-09-10	London	34
6	3006	1000	2022-09-08	London	40
7	3007	1003	2020-09-22	Milan	42
8	3008	1005	2010-09-25	Paris	27



# DesignerCredits

**This table holds each designer's credits within shows --> all the shows they have designed for.**

```
CREATE TABLE DesignerCredits (  
  DID int not null REFERENCES Designers(DID),  
  showID int REFERENCES FashionShows(showID),  
  PRIMARY KEY (DID, showID)  
);
```

## Functional Dependencies:

- No Dependencies



	did [PK] integer 	showid [PK] integer 
1	1	3001
2	1	3004
3	2	3003
4	5	3002
5	8	3006
6	5	3006
7	7	3008
8	7	3007
9	8	3005

# DirectorCredits

This table holds all director credits --> what shows each director has worked for

```
CREATE TABLE DirectorCredits (  
  SDID int not null REFERENCES FashionShowDirectors(SDID),  
  showID int REFERENCES FashionShows(showID),  
  PRIMARY KEY (SDID, showID)  
);
```

- Functional Dependencies:**
- No Dependencies

	sdid [PK] integer 	showid [PK] integer 
1	2	3003
2	3	3001
3	7	3002
4	2	3007
5	7	3004
6	10	3005



# ModelCredits

This table holds all Model credits --> what shows each model has worked for

```
CREATE TABLE ModelCredits (  
  MID int not null REFERENCES Models(MID),  
  showID int REFERENCES FashionShows(showID),  
  PRIMARY KEY (MID, showID)  
);
```

## Functional Dependencies:

- No Dependencies

	mid [PK] integer 	showid [PK] integer 
1	1	3003
2	4	3007
3	6	3007
4	8	3003
5	9	3005
6	4	3005
7	6	3002
8	9	3006

# Fabrics

This table holds all fabrics with their ID, name, cost USD per Yard, and color.

```
CREATE TABLE Fabrics (  
  FabricID int not null PRIMARY KEY,  
  fabricName text not null,  
  costUSDperYard decimal(10, 2),  
  color text  
);
```

	<div><div>fabricid</div><div>[PK] integer </div></div>	<div><div>fabricname</div><div>text </div></div>	<div><div>costusdperyard</div><div>numeric (10,2) </div></div>	<div><div>color</div><div>text </div></div>
1	202	BlueSilk	15.30	blue
2	203	RedSilk	15.30	red
3	204	PinkRayon	20.50	pink
4	205	BlueMesh	10.25	Blue
5	206	OrangePolyester	9.05	Orange

## Functional Dependencies:

- FabricID → fabricName, costUSDperYard, color



# Garments

This table holds the garments with their ID, name, price, and what show they were shown in.

```
CREATE TABLE Garments (  
  garmentID int not null PRIMARY KEY,  
  garmentName text not null,  
  garmentPrice decimal(10, 2),  
  showID int REFERENCES FashionShows(showID)  
);
```

**Functional Dependencies:**

- **GarmentID → GarmentName, GarmentPrice, ShowID**

	garmentid [PK] integer	garmentname text	garmentprice numeric (10,2)	showid integer
1	303	Layla Top	100.92	3001
2	304	Sandy Pants	300.70	3002
3	305	Best Shoes	700.80	3003
4	306	Sandra Cardigan	600.20	3003
5	307	Layla Top	500.30	3001
6	308	HA Sweater	200.50	3004
7	309	Kendra Skirt	350.00	3005
8	310	Aubrey Skirt	375.25	3006
9	311	Pat Top	300.00	3007
10	312	Summer Shots	1000.25	3008



# FabricSuppliers

This table holds all the fabric suppliers with their ID, name, and home city

```
CREATE TABLE FabricSuppliers (  
  supplierID int not null PRIMARY KEY,  
  supplierName text not null,  
  homeCity text  
);
```

	supplierid [PK] integer	suppliername text	homecity text
1	701	Sender	New York
2	702	Fabrics Expert	Ontario
3	703	BuyHere	Ontario
4	704	SellerStuff	Charleston

**Functional Dependencies:**

- **SupplierID→  
SupplierName, HomeCity**

# SupplierCatalog

This table holds what Fabrics each supplier sells

```
CREATE TABLE SupplierCatalog (  
  SupplierID int not null REFERENCES FabricSuppliers(supplierID),  
  fabricID int not null REFERENCES Fabrics(FabricID),  
  PRIMARY KEY (SupplierID,fabricID)  
);
```

## Functional Dependencies:

- No Dependencies

	supplierid [PK] integer	fabricid [PK] integer
1	701	202
2	701	203
3	702	204
4	703	205
5	704	206
6	704	205
7	704	203
8	702	203
9	703	206




# FabricUses

This table hold each use of the fabrics through storing the FabricID, garmentID, and SupplierID.

```
CREATE TABLE FabricUses (  
  FabricID int not null REFERENCES Fabrics(FabricID),  
  garmentID int not null REFERENCES Garments(garmentID),  
  supplierID int not null REFERENCES FabricSuppliers(supplierID),  
  PRIMARY KEY (FabricID, garmentID, supplierID)  
);
```

## Functional Dependencies:

- No Dependencies

	fabricid [PK] integer 	garmentid [PK] integer 	supplierid [PK] integer 
1	202	303	701
2	203	303	701
3	204	304	702
4	205	305	703
5	206	306	704
6	204	307	702
7	203	308	702
8	203	309	704
9	204	310	702
10	202	311	701
11	206	312	703

REPORTS, VIEWS,  
STORED PROCEDURES,  
TRIGGERS, SECURITY



# REPORTS

**This report queries all models who have ever walked in a show in Paris. This can be used to analyze the commonality of models walking in shows based in Paris.**

	pid [PK] integer	firstname text	lastname text
1	4	Cindy	Crawford
2	9	Tucker	Angelo

```
select distinct p.pid, p.firstName, p.LastName
from
  People p
  inner join Models m on p.pid = m.MID
  inner join ModelCredits mc on m.MID = mc.MID
  inner join FashionShows fs on mc.showID = fs.showID
where fs.city = 'Paris';
```

# REPORTS

This report queries all relevant information about models who walked their first show under the age of 21 in the city of London. This can be used to analyze trends in how old the average model whose first show in London is walking in London, and can be used to compare the year they started to the year they walked their first show

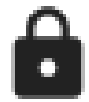
	firstname text	lastname text	heightinches integer	modelyearstarted integer	dob date	firstshowdate date	ageatfirstshow numeric
1	Tucker	Angelo	75	2021	2001-02-19	2021-09-10	20

```
select p.firstName,p.lastName,m.heightInInches,m.ModelyearStarted,p.DOB,
       min(fs.date) as firstShowDate,
       extract(year from AGE(min(fs.date), p.DOB)) as ageAtFirstShow
from People p
       inner join Models m on p.pid = m.MID
       inner join ModelCredits mc on m.MID = mc.MID
       inner join FashionShows fs on mc.showID = fs.showID
where fs.city = 'London'
group by p.pid, p.firstName, p.lastName, m.heightInInches, m.ModelyearStarted, p.DOB
Having extract(year from age(min(fs.date), p.DOB)) < 21;
```

# REPORTS

This report queries all brands that have worked with supplier 701. This can be used to analyze the popularity of this supplier

```
select distinct b.BrandName
from Brands b
where b.BID in
  (select distinct fs.BID
   from FashionShows fs
    inner join Garments g on fs.showID = g.showID
    inner join FabricUses fu on g.garmentID = fu.garmentID
   where fu.supplierID = 701);
```

	brandname 
1	Dior
2	Fendi



# VIEWS: MultipleJobs

This view is used to query all people in the database who work more than one job → these people are not exclusively models, directors, or designers, they have 2 or more roles.

```
create or replace view MultipleJobs as
select p.pid, p.firstName, p.lastName, p.homeCity, p.DOB,
       count(distinct t.job) as NumberOfJobs
from People p
  left join (
    select DID as pid, 'Designer' AS job FROM Designers
  union all
    select MID as pid, 'Model' as job from Models
  union all
    select SDID as pid, 'Director' as job from FashionShowDirectors
  ) t ON p.pid = t.pid
group by p.pid, p.firstName, p.lastName, p.homeCity, p.DOB
having count(distinct t.job) > 1;
```

```
select *
from MultipleJobs;
```

	pid integer	firstname text	lastname text	homecity text	dob date	numberofjobs bigint
1	1	Shannon	Maier	New York	1970-05-04	2
2	2	Maggie	Lawrence	Paris	2000-05-02	2
3	7	Denise	Smith	Los Angeles	1967-02-02	2
4	8	Colin	Burrow	Charleston	2000-01-17	2



# VIEWS: MostExpensiveGarmentInMilan

This view finds the highest-priced item in Milan, what brand made it, and when it was shown in a Fashion Show.

```
create or replace view MostExpensiveGarmentInMilan as
select g.garmentName, g.garmentPrice,b.BrandName,fs.city AS showCity,fs.date AS showDate
from Garments g
    inner join FashionShows fs ON g.showID = fs.showID
    inner join Brands b ON fs.BID = b.BID
Where fs.city = 'Milan'
Order By g.garmentPrice DESC
Limit 1;
```

```
select *
from MostExpensiveGarmentInMilan;
```

	garmentname text	garmentprice numeric (10,2)	brandname text	showcity text	showdate date
1	Sandy Pants	300.70	Louis Vuitton	Milan	2020-09-29

# VIEWS: CityShowsByYear

This report queries all relevant information about models who walked their first show under the age of 21 in the city of London. This can be used to analyze trends in how old the average model whose first show in London is walking in London, and can be used to compare the year they started to the year they walked their first show

```
create or replace view CityShowsByYear AS
select city, extract(year from date) as year,
count(*) as NumberOfShows
from FashionShows
Group by city,extract(year from date)
order by year ASC, city ASC;
```

```
select *
from CityShowsByYear;
```

	city text	year numeric	numberofshows bigint
1	New York	2010	2
2	Paris	2010	1
3	Milan	2020	2
4	London	2021	1
5	London	2022	1
6	Paris	2023	1

# STORED PROCEDURES: GetBrandByFabricAndSupplier

This stored procedure queries the brand names based on two inputs: FabricName and SupplierID. Using these two parameters the procedure finds what brands have used the inputted fabric supplied by the inputted supplier in any of their garments. Below I show an example using BlueSilk and Supplier 701.

```
create or replace function getBrandsByFabricAndSupplier(text, int, REFCURSOR) returns refcursor as
$$
declare
  EnterfabricName text      := $1;
  EntersupplierId  int       := $2;
  brandName       REFCURSOR := $3;
Begin
  open brandName for
    select distinct b.Brandname
    from Brands b
      inner join FashionShows fs ON b.BID = fs.BID
      inner join Garments g ON fs.showID = g.showID
      inner join FabricUses fu ON g.garmentID = fu.garmentID
      inner join Fabrics f ON fu.FabricID = f.FabricID
    where f.fabricName = EnterfabricName AND fu.supplierID = EntersupplierId;
  return brandName;
end;
$$
language plpgsql;
```

```
select getBrandsByFabricAndSupplier('BlueSilk', 701, 'results');
fetch all from results;
```

	brandname text
1	Dior
2	Fendi

# STORED PROCEDURES: GetAverageGarmentPrice

This stored procedure takes in a brand name and calculates the average price of garments displayed at their fashion shows by using the AVG function and joins. This can be helpful to analyze the trends in prices throughout the fashion industry. In my example, I use Staud as an input to query the table shown.

```
create or replace function getAverageGarmentPrice(text, REFCURSOR) returns refcursor as
$$
declare
    enterBrandName text          := $1;
    AverageGarmentPrice REFCURSOR := $2;
Begin
    open AverageGarmentPrice for
        select ROUND(avg(g.garmentPrice), 2) as AverageGarmentPrice
        from garments g
            inner join FashionShows fs on g.showID = fs.showID
            inner join Brands b on fs.BID = b.BID
        Where b.Brandname = enterBrandname;
    return AverageGarmentPrice;
end;
$$
language plpgsql;
```

```
select getAverageGarmentPrice('Staud', 'results');
fetch all from results;
```

	averagegarmentprice
	numeric
1	650.50

# TRIGGERS:CheckDirectorStartYear

**This trigger checks that any time a credit is added to DirectorCredits, the date of the show is after the year the Director started his or her job. If the date is before the recorded year, an error message will raise.**

```
create or replace function checkDirectorStartYear() returns trigger as
$$
Begin
    If (select DirectyearStarted
        from FashionShowDirectors
        where SDID = new.SDID) >
        (EXTRACT(year from (select date from FashionShows where showID = NEW.showID)))
    then raise exception 'Director cannot work a show before their career start year.';
    end if;
    return new;
end;
$$

language plpgsql;
```

```
INSERT INTO DirectorCredits (SDID, showID)
Values (002,3001);
```

```
ERROR: Director cannot work a show before their career start year.
CONTEXT: PL/pgSQL function checkdirectorstartyear() line 7 at RAISE
```

```
SQL state: P0001
```

```
Create trigger CheckingDirectorStartYear
Before insert or update on DirectorCredits
For each row
execute procedure CheckDirectorStartYear();
```



# TRIGGERS: DiorDoesNotGo

**This trigger prevents the user from deleting the Dior entry in the Brands table by checking if the delete statement calls for the deletion of the BrandName or the BID. Dior is just too iconic to ever be deleted!**

```
create or replace function DiorDoesNotGo() returns trigger as
$$
Begin
    if old.Brandname = 'Dior' or old.BID = 1003
    then raise exception 'Deletion of Dior is prohibited becuae it is just too iconic!';
    end if;

    return old;
end;
$$
language plpgsql;
```

```
DELETE from Brands
where BrandName = 'Dior';
```

```
Create trigger DiorIsntGoing
Before delete on Brands
For each row
Execute procedure DiorDoesNotGo();
```

```
ERROR:  Deletion of Dior is prohibited becuae it is just too iconic!
CONTEXT:  PL/pgSQL function diordoesnotgo() line 4 at RAISE
```

```
SQL state: P0001
```

# SECURITY

The administrator is allowed access to perform all procedures on existing tables and create new tables

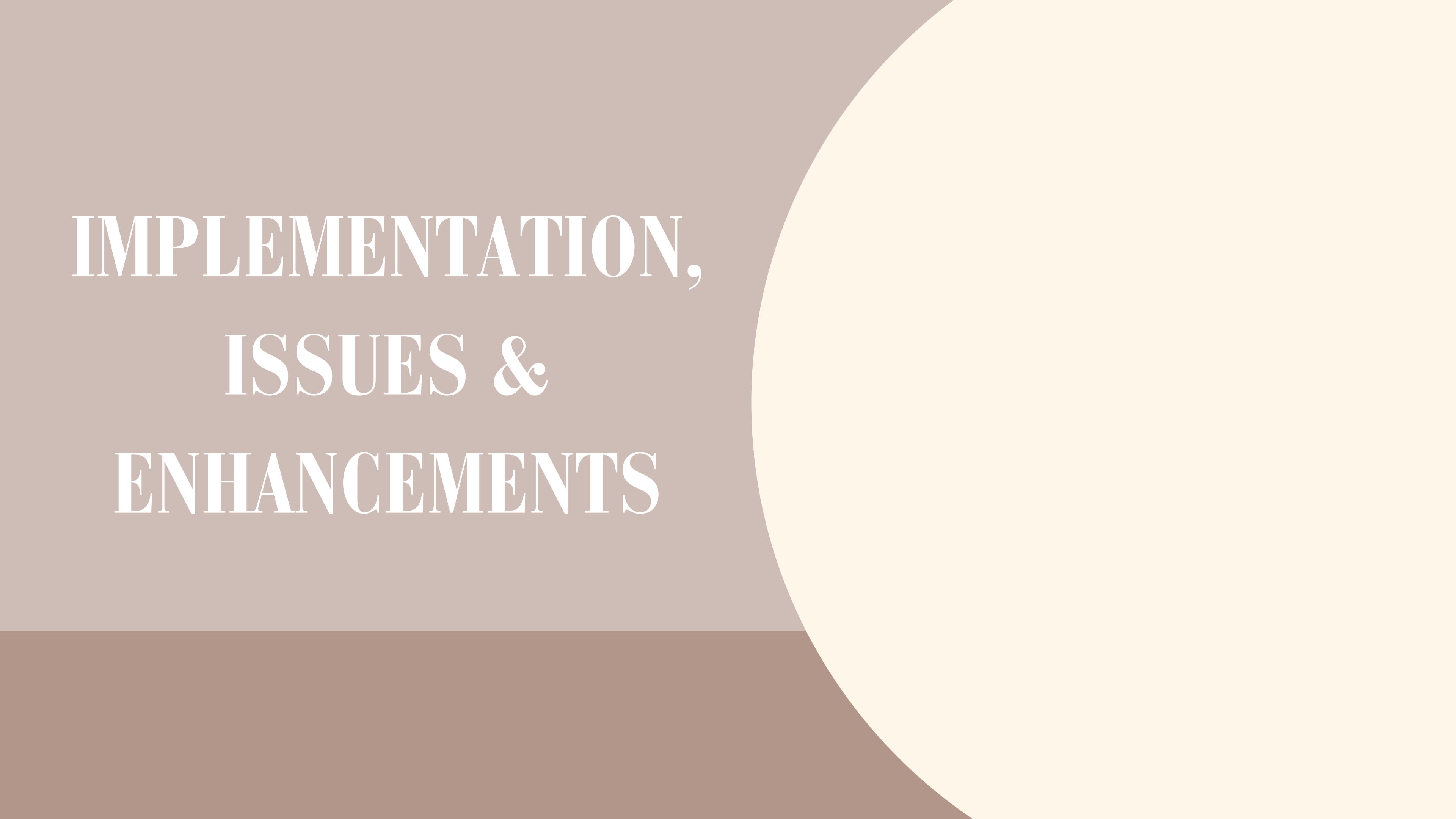
```
Create Role Admin;  
Grant all on all tables in schema public to admin;
```

The BrandCEO can access all information for tables and insert rows, but cannot update existing data.

```
Create Role BrandCEO;  
Grant select, insert on all tables in Schema Public to BrandCEO;
```

The fashion Planner has the ability to select items from the People, Designers, Models, Directors, brands, Fabrics, and Fabric Suppliers, but cannot update these tables. The tables they are allowed to access and update/insert into have to do with them styling outfits for brands.

```
Create Role FashionPlanner;  
Grant select on People, Designers, Models, FashionShowDirectors, Brands, Fabrics, FabricSuppliers to FashionPlanner;  
Grant select, insert, update on FashionShows, Garments to FashionPlanner;  
Grant select, insert on DesignerCredits, ModelCredits, DirectorCredits to FashionPlanner;
```



# IMPLEMENTATION, ISSUES & ENHANCEMENTS



# Implementation Notes/Known Problems/Future Enhancements

- **Implementation Notes**

- There is a limited amount of sample data included in this report, and with more data, one would be able to use queries to produce more valuable answers
- If this database were to be implemented there would most likely be tables added to account for other jobs within the fashion industry

- **Known Problems**

- Because the fields FabricName and SupplierName are not PKs, I cannot place them into the Supplier Catalog referencing the original inputs, and the user can not directly see the names of the fabrics and suppliers in this table, but has to go by IDs.
- A fashion show can be added at date prior to when the brand being shown was created.

- **Future Enhancements**

- This database could be improved through the addition of more check constraints
  - Ex: Check on homeCity in People and Suppliers to make sure the city exists
- More stored procedures should be made to make sorting the data easier
- Add a table/field to show how many yards of each fabric is used in each garment