

Introduction

The United States clothing manufacturing industry has been in decline for many years. Nearly half of the private clothing manufacturers have gone out of business in the last decade, and they have dropped approximately 80% of their employees in the last two decades. In order for any clothing manufacturing business to stay afloat, an efficient and concise method of finding production costs is needed. One effective cost management method can be found through the use of a detailed database. Such a database is designed with the purpose of presenting costs, its attributors, and any important information pertaining to them, where the information stored is easily accessible and comprehensible for any user. This document describes such a database.

Report Overview

The following report details the design and implementation of a database that is centered around the production costs of a clothing manufacturer. The report structure and the database design follows an entity-relationship(ER) model. The ER model is used to depict the relationship of certain components necessary in creating the clothing manufacturer's pivotal product, a garment. To allow ease of grasp and comprehension, the overall model is partitioned into seven core sections. Each section of the report is elaborated with descriptions and input formats showing how one could input new data. The sections after the descriptions of the seven parts of the model will contain further implementation details, decisions on certain entities and their attributes, and test results for maintaining integrity.

Table of Contents

Introduction and Report Overview

page 1

ER Schema

Part 1: Piece Diagram	page 3
Part 2: Notions Diagram	page 7
Part 3: Dye	page 10
Part 4: Seam	page 13
Part 5: Processing	page 17
Part 6: FOB	page 19
Part 7: Photo	page 22
Complete ER Diagram	page 25

Planning Process and Distribution of Work

page 26

Further Implementation Details

Data Input	page 28
Integrity Testing	page 37
Additional Tools	page 62
Web Interface	page 63

Student Contributions

Part 1: Piece Diagram

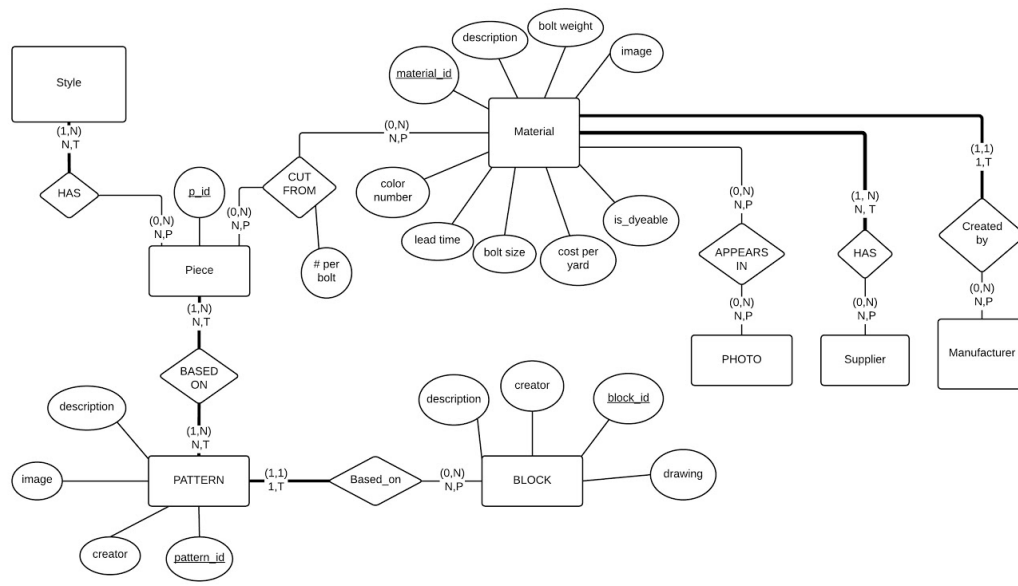


Diagram Overview

The central entities here are *Piece*, *Material*, *Pattern*, and *Block*. Every garment, or *style*, is made up of pieces, and each *piece* is *based on* a *pattern* design and may be *cut from* a certain *material*. The pattern design is based on *blocks*, which are essentially generic patterns in and of themselves. The *Material* also has a *supplier*, a *manufacturer* and *photos* for viewing. The four central entities come together in order to make standard items such as sleeves, collars, blouses, etc.

Piece Diagram: Piece Entity

Piece Entity Implementation

```
CREATE TABLE piece
(
  p_id serial NOT NULL,
  CONSTRAINT piece_pkey PRIMARY KEY (p_id)
);
```

Piece Description

- *piece* is a table that contains and provides individual identification numbers for every piece of garment. It can be used in conjunction with other entities in order to determine the overall product in question.
- *serial* in PostgreSQL is an automatic incrementing integer.

Piece Diagram: Pattern Entity

Pattern Entity Implementation

```
CREATE TABLE pattern
(
  pattern_id serial NOT NULL,
  creator character varying(30) NOT NULL,
  description character varying(512),
  block_id integer,
  image character varying(100),
  CONSTRAINT pattern_pkey
    PRIMARY KEY (pattern_id),
  CONSTRAINT pattern_block_id_fkey
    FOREIGN KEY (block_id)
      REFERENCES block (block_id)
    MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE RESTRICT
);
```

Pattern Description

- *pattern* holds the list of templates that can be used for cutting out a piece of garment.
- *Pattern* must exist alongside a *Block* entity, as *Block* is the generic template of which *Pattern* is based on.
- *Pattern* contains the attributes *pattern_id*, *creator*, *description*, *block_id*, *image*.
 - *block_id*, references to the *Block* entity and *image* is a web link to an image of the pattern itself.

Piece Diagram: Block Entity

Block Entity Implementation

```
CREATE TABLE block
(
  block_id serial NOT NULL,
  creator character varying(30) NOT NULL,
  description character varying(512),
  drawing character varying(100),
  CONSTRAINT block_pkey
    PRIMARY KEY (block_id)
);
```

Block Description

- *Blocks* are simplistic shapes and cuts of a *Piece* which effectively act as an outline to develop *Patterns*.
 - *block* holds a list of these basic templates.
- *Block* attributes are *block_id*, *creator*, *description*, *drawing*.
 - *drawing* is a web link to an image of the specific block.

Piece Diagram: *Material* Entity

Material Entity Implementation

```
CREATE TABLE material
(
    material_id serial NOT NULL,
    man_id integer,
    description character varying(512),
    color_number integer NOT NULL,
    bolt_size character varying(30)
        NOT NULL,
    bolt_weight character varying(15)
        NOT NULL,
    cost_per_yard money NOT NULL,
    is_dyeable boolean,
    lead_time time without time zone,
    image character varying(100),
    CONSTRAINT material_pkey
        PRIMARY KEY (material_id),
    CONSTRAINT material_man_id_fkey
        FOREIGN KEY (man_id)
        REFERENCES manufacturer (man_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE RESTRICT
);
```

Material Description

- *Material* refers to things like leather, wool, cotton, etc. It also contains the length of the material and supplier information. It must also come from a *Supplier* and a *Manufacturer*.
- The *Material* table contains a *material_id*, an id for each unique type of material, as its primary key. It also contains *man_id* which is a foreign key that references *Manufacturer*. Every material has a manufacturer, so the ON DELETE RESTRICT constraint is applied to the foreign key to prevent deletion. The material's *color_number*, *bolt_size*, *bolt_weight*, and *cost_per_yard* are set to NOT NULL because every material requires these.

Piece Diagram: *Piece_Pattern* Relationship

Piece_Pattern Relationship Implementation

```
CREATE TABLE piece_pattern
(
    p_id integer NOT NULL,
    pattern_id integer NOT NULL,
    CONSTRAINT piece_pattern_pkey
        PRIMARY KEY (p_id, pattern_id),
    CONSTRAINT piece_pattern_p_id_fkey
        FOREIGN KEY (p_id)
        REFERENCES piece (p_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT piece_pattern_pattern_id_fkey
        FOREIGN KEY (pattern_id)
        REFERENCES pattern (pattern_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);
```

Piece_Pattern Description

- *piece_pattern* is the relationship between the *Piece* entity and *Pattern* entity. On the diagram, it is displayed as the relationship "*based on*".
- *Piece_Pattern* references the specific template used to cut the piece of the garment. This relationship relies the *Piece* primary key, *p_id*, and *Pattern* primary key, *pattern_id*. The two are used as composite primary key in this relationship to identify itself.

Piece Diagram: Piece_Material Relationship

Piece_Material Implementation

```
CREATE TABLE piece_material
(
  p_id integer NOT NULL,
  material_id integer NOT NULL,
  number_per_bolt integer,
  CONSTRAINT piece_material_pkey
    PRIMARY KEY (p_id, material_id),
  CONSTRAINT
    piece_material_material_id_fkey
    FOREIGN KEY (material_id)
    REFERENCES material (material_id)
    MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT piece_material_p_id_fkey
    FOREIGN KEY (p_id)
    REFERENCES piece (p_id)
    MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);
```

Piece_Material Description

- *piece_material* is the relationship between the *Piece* entity and *Material* entity. It is displayed on the diagram as the relationship “*cut from*”.
- The *Piece_material* table contains the foreign key *p_id*, which references *p_id* from the *Piece* table. It also contains the foreign key *material_id* which references *material_id* from the *Material* table. The composite primary key for this table is the combination of *p_id* and *material_id*. Finally the *Piece* to material relation contains an attribute , *number_per_bolt*, which is stored in this table.

Piece Diagram: Style_Piece Relationship

Style_Piece Implementation

```
CREATE TABLE style_piece
(
  style_id integer NOT NULL,
  p_id integer NOT NULL,
  CONSTRAINT style_piece_pkey
    PRIMARY KEY (style_id, p_id),
  CONSTRAINT style_piece_p_id_fkey
    FOREIGN KEY (p_id)
    REFERENCES piece (p_id)
    MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT style_piece_style_id_fkey
    FOREIGN KEY (style_id)
    REFERENCES style (style_id)
    MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE RESTRICT
);
```

Style_Piece Description

- *Style_Piece* represents the “has” relationship between *Piece* and *Style*.
- *Style_Piece* holds the links between what style a piece used.
- This table relates the style entity with the Piece entity. As such, it contains the primary keys of both *Style* and *Piece* as foreign keys. The primary key consists of both foreign keys. *Style* must contain at least one piece, so it has the constraint ON DELETE RESTRICT.

Part 2: Notion Diagram

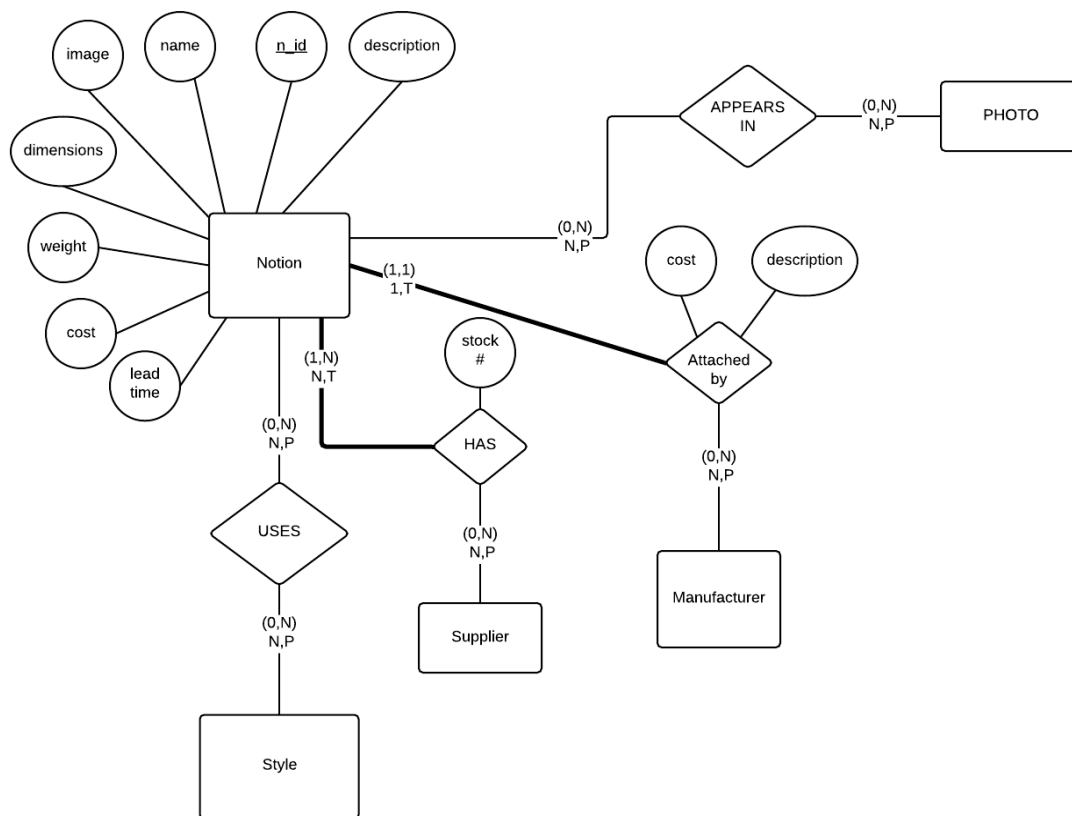


Diagram Overview

The main entity in this diagram is *Notion*. Notions in fashion refer to things such as zippers, buttons, beads, etc. A notion may be a part of any *style*. Each notion may also be modeled in a *photo*. A notion must have a *supplier* and be attached by a *manufacturer*. Collectively, this information helps deduce part of the cost of a garment.

Notion Diagram: Notion Entity

Notion Entity Implementation

```
CREATE TABLE Notion
(
  n_id SERIAL PRIMARY KEY,
  man_id INT REFERENCES
    Manufacturer(man_id)
    ON DELETE RESTRICT,
  name VARCHAR(50) NOT NULL,
  description VARCHAR(512),
  image VARCHAR(100),
  dimensions VARCHAR(30) NOT NULL,
  weight VARCHAR(15) NOT NULL,
  cost MONEY NOT NULL,
  attach_cost MONEY NOT NULL,
  attach_description VARCHAR(512),
  lead_time TIME
);
```

Notion Description

- The `Notion` table contains `n_id`, an id for each unique type of notion, as its primary key. It also contains `man_id` which is a foreign key that references *Manufacturer*. Every notion has a manufacturer, so the ON DELETE RESTRICT constraint has been applied to the foreign key to prevent deletion.
- NOT NULL has been set to the notion's `name`, `dimensions`, `weight`, `cost`, and `attach_cost`, because every notion requires these.

Notion Diagram: Material_Notion_Photo Relationship

Material_Notion_Photo Implementation

```
CREATE TABLE Material_Notion_Photo
(
  photo_id INT REFERENCES
    Photo(photo_id),
  material_id INT REFERENCES
    Material(material_id),
  n_id INT REFERENCES Notion(n_id),
  PRIMARY KEY
    (photo_id, material_id, n_id)
);
```

Material_Notion_Photo Description

- The `Material_Notion_Photo` table links *Material*, *Notion*, and *Photo* together. This table contains the `photo_id`, `material_id`, and `n_id` from the notion. The composite primary key consists of all three of these combined.
- This relationship describes the photo in which a notion and material appear.

Notion Diagram: Style_Notion Relationship

Style_Notion Implementation

```
CREATE TABLE Style_Notion
(
  style_id INT REFERENCES
    Style(style_id),
  n_id INT REFERENCES Notion(n_id),
  PRIMARY KEY (style_id, n_id)
);
```

Style_Notion Description

- The `Style_Notion` table contains the primary keys of the *Style* and *Notion* tables. This table relates *Style* and *Notion* in a many-to-many relationship.
- This relationship describes what notion is used in what style.

Notion Diagram: Supplier_Notion Relationship

Supplier_Notion Implementation

```
CREATE TABLE Supplier_notion
(
    stock_number INT,
    sup_id INT REFERENCES
        Supplier(sup_id),
    n_id INT REFERENCES Notion(n_id)
        ON DELETE RESTRICT,
    PRIMARY KEY (sup_id, n_id)
);
```

Supplier_Notion Description

- The `Supplier_notion` has a `stock_number` attribute. This table also has two foreign keys one that references the *Supplier's* primary key, and the other references the *Notion's* primary key. This table has a composite primary key, whose components are foreign keys referring to the primary keys of the associated entities' relations. Also, the key from *Notion* has the constraint ON DELETE RESTRICT placed on it.

Part 3: Dye Diagram

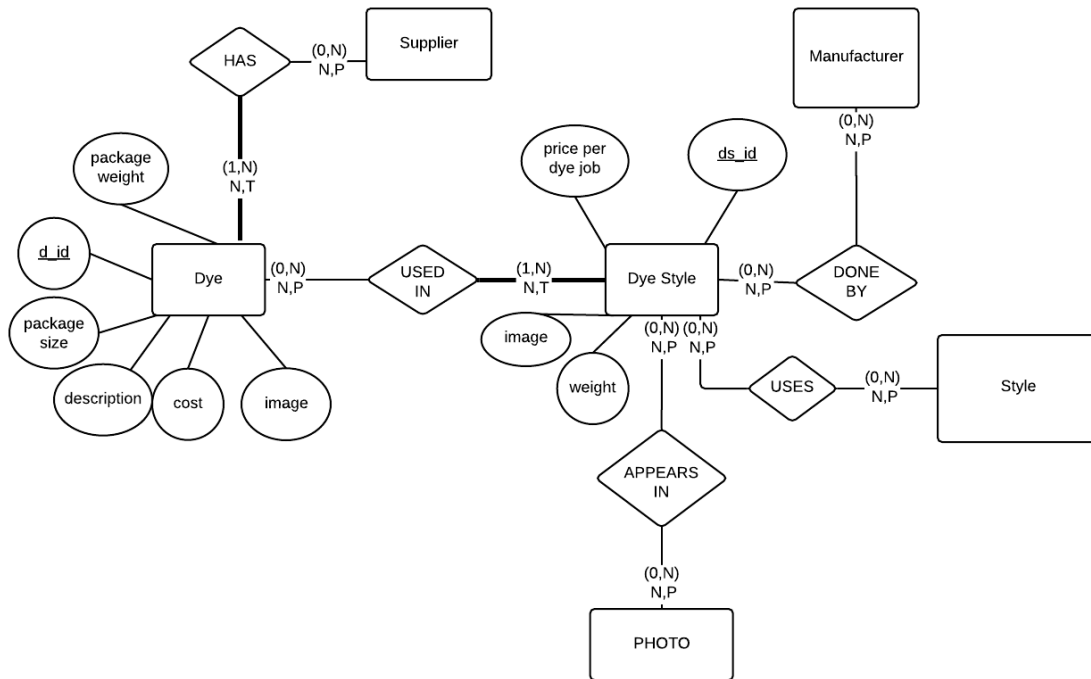


Diagram Overview

The main entities in this section are *Dye* and *Dye Style*. *Dye styles* are used in the *style*, and each dye style uses one or more *dyes*. Dye styles are done by a *manufacturer* and can appear in *photos*. Dyes are provided by a *supplier*.

Dye Diagram: Dye Style Entity

Dye Style Entity Implementation

```

CREATE TABLE Dye_Style
(
    ds_id SERIAL PRIMARY KEY,
    price_per_dye_job MONEY,
    image BYTEA,
    weight VARCHAR(20)
);
  
```

Dye Style Description

- The `Dye_Style` table contains a unique `ds_id` (dye style identifier) as its primary key. In addition, it contains the `price per dye job`, an `image` of the dye style and the `weight` of the dye used in grams.

Dye Diagram: Dye Entity

Dye Entity Implementation

```
CREATE TABLE Dye
(
    d_id SERIAL PRIMARY KEY,
    package_weight VARCHAR(20) NOT NULL,
    package_size VARCHAR(20) NOT NULL,
    description TEXT,
    cost MONEY NOT NULL,
    image BYTEA
);
```

Dye Description

- The *Dye* table utilizes a unique *d_id* (dye identifier) as its primary key. It also contains additional attributes corresponding to the dye's package weight, package size, description, cost and sample image.

Dye Diagram: Style_Dye_Style Relationship

Style_Dye_Style Relationship Implementation

```
CREATE TABLE Style_Dye_Style
(
    style_id INT REFERENCES
        Style(style_id),
    ds_id INT REFERENCES Dye_Style(ds_id),
    PRIMARY KEY (style_id, ds_id)
);
```

Style_Dye_Style Description

- This table relates the *Style* entity with the *Dye Style* entity, holding their primary keys *style_id* and *ds_id*, respectively, as foreign keys. These two foreign keys are used together to make this table's composite primary key.

Dye Diagram: Dye_Style_Manufacturer Relationship

Dye_Style_Manufacturer Relationship Implementation

```
CREATE TABLE Dye_Style_Manufacturer
(
    ds_id INT REFERENCES Dye_Style(ds_id),
    man_id INT REFERENCES
        Manufacturer(man_id),
    PRIMARY KEY (ds_id, man_id)
);
```

Dye_Style_Manufacturer Description

- This table relates the *Dye Style* entity with the *Manufacturer* entity, so it contains the primary keys of both *Dye Style*, *ds_id*, and *Manufacturer*, *man_id*, as foreign keys. Together, these two foreign keys are used to create this table's composite primary key.

Dye Diagram: Dye_Style_Dye Relationship

Dye_Style_Dye Relationship Implementation

```
CREATE TABLE Dye_Style_Dye
(
  ds_id INT REFERENCES Dye_Style(ds_id)
    ON DELETE RESTRICT,
  d_id INT REFERENCES Dye(d_id),
  PRIMARY KEY (ds_id, d_id)
);
```

Dye_Style_Dye Description

- This table relates the *Dye Style* entity with the *Dye* entity, containing the primary keys of both Dye Style, *ds_id*, and Dye, *d_id*. These two primary keys (stored as foreign keys) are used to create the composite primary key of this relation.
- Since each dye style must contain at least one dye, *Dye_Style* has the constraint ON DELETE RESTRICT placed on it.

Dye Diagram: Photo_DyeStyle Relationship

Photo_DyeStyle Relationship Implementation

```
CREATE TABLE Photo_DyeStyle
(
  photo_id INT REFERENCES
    Photo(photo_id),
  ds_id INT REFERENCES Dye_Style(ds_id),
  PRIMARY KEY (photo_id, ds_id)
);
```

Photo_DyeStyle Description

- The *Photo_DyeStyle* table links a *Dye Style* to a *Photo*. This table contains a *photo_id* for the photo and *ds_id* for the Dye Style. The composite primary key consists of the *photo_id* and *ds_id* keys combined.

Dye Diagram: Supplier_dye Relationship

Supplier_dye Relationship Implementation

```
CREATE TABLE Supplier_dye
(
  sup_id INT REFERENCES
    Supplier(sup_id),
  d_id INT REFERENCES Dye(d_id)
    ON DELETE RESTRICT,
  PRIMARY KEY(sup_id, d_id)
);
```

Supplier_dye Description

- This table contains two foreign keys. *sup_id* references the primary key of the *Supplier* entity and *d_id* references the primary key of the *Dye* entity. The *Dye* table has the ON DELETE RESTRICT placed on it. The *Supplier_dye* table has a composite primary key made up of the two participating entities' keys.

Part 4: Seam Diagram

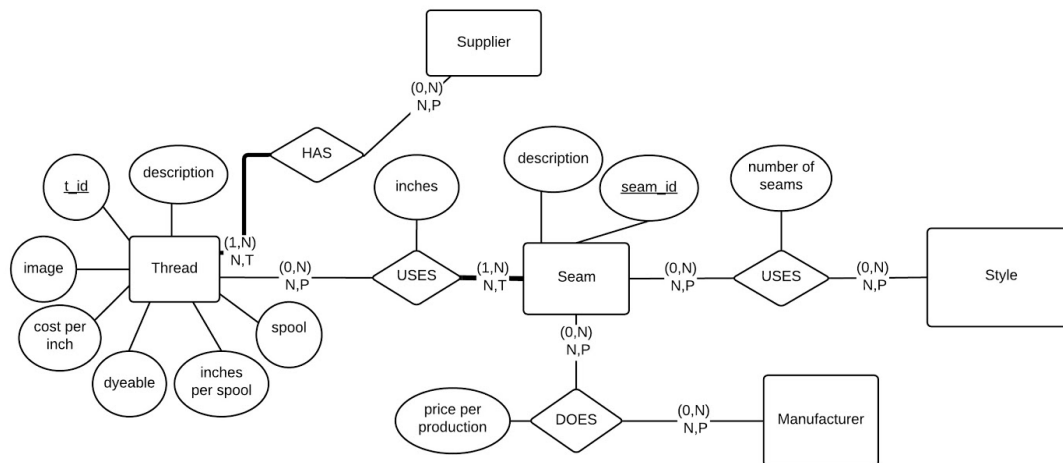


Diagram Overview

The core entities in this section are *Seam* and *Thread*. Every garment may have number of *seams*, which is where two or more pieces of cloths are joined together. The seams themselves require a number of *threads* for stitching the clothes. The threads are bought from certain *suppliers*. Of course, every seam is done by a *manufacturer* and are used in certain *styles*. *Seams* and *threads* come together to create linings on items like jeans, the fingers on gloves, to stylize bed covers, and so on.

Seam Diagram: *Seam* Entity

Seam Entity Implementation

```
CREATE TABLE seam
(
    seam_id serial NOT NULL,
    description text,
    CONSTRAINT seam_pkey
        PRIMARY KEY (seam_id)
)
```

Seam Description

- The Seam table simply contains a unique `seam_id` (seam identifier) as its primary key, and an additional attribute containing a description of the seam.

Seam Diagram: *Thread* Entity

Thread Entity Implementation

```
CREATE TABLE thread
(
    t_id serial NOT NULL,
    description text,
    cost_per_inch money NOT NULL,
    dyeable boolean NOT NULL,
    inches_per_spool numeric NOT NULL,
    spool integer NOT NULL,
    image character varying(100),
    CONSTRAINT thread_pkey PRIMARY KEY (t_id)
);
```

Thread Description

- The Thread table first holds a unique primary key entitled “`t_id`” (thread identifier). Extra attributes include the thread’s description, sample image, `cost_per_inch` and whether the thread is `dyeable`. The final two attributes relate to the thread’s `spool`, specifically the `inches per spool` and the actual spool itself.

Seam Diagram: *style_seam* Relationship

style_seam Relationship Implementation

```
CREATE TABLE style_seam
(
    style_id integer NOT NULL,
    seam_id integer NOT NULL,
    num_seams integer NOT NULL,
    CONSTRAINT style_seam_pkey
        PRIMARY KEY (style_id, seam_id),
    CONSTRAINT style_seam_seam_id_fkey
        FOREIGN KEY (seam_id)
        REFERENCES seam (seam_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT style_seam_style_id_fkey
        FOREIGN KEY (style_id)
        REFERENCES style (style_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE RESTRICT
);
```

style_seam Description

- This relationship is set between *Seam* and *Style*, representing “USES” on the diagram.
 - This table contains the list of seams paired with styles.
- This table relates the *Style* entity with the *Seam* entity, containing the primary keys of *Style* and *Seam* as foreign keys. These two foreign keys combined make up the table’s primary key. The table also contains an additional attribute representing the number of seams.

Seam Diagram: *seam_manufacturer* Relationship

seam_manufacturer Relationship Implementation

```
CREATE TABLE seam_manufacturer
(
    seam_id integer NOT NULL,
    man_id integer NOT NULL,
    price_per_prod money NOT NULL,
    CONSTRAINT seam_manufacturer_pkey
        PRIMARY KEY (seam_id, man_id),
    CONSTRAINT seam_manufacturer_man_id_fkey
        FOREIGN KEY (man_id)
        REFERENCES manufacturer (man_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT
        seam_manufacturer_seam_id_fkey
        FOREIGN KEY (seam_id)
        REFERENCES seam (seam_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);
```

seam_manufacturer Description

- This relationship is between *Manufacturer* entity and *Seam* which is represented by “DOES” on the diagram.
 - This table holds the manufacturer of each seam and the cost as invoiced by the manufacturer.
- This table relates the Seam entity with the Manufacturer entity, containing the primary keys of both Seam and Manufacturer as foreign keys. These two foreign keys are combined to form the table’s primary key. In addition, the table also carries an attribute for the price of each seam produced.

Seam Diagram: *thread_seam* Relationship

thread_seam Relationship Implementation

```
CREATE TABLE thread_seam
(
    t_id integer NOT NULL,
    seam_id integer NOT NULL,
    inches numeric NOT NULL,
    CONSTRAINT thread_seam_pkey
        PRIMARY KEY (t_id, seam_id),
    CONSTRAINT thread_seam_seam_id_fkey
        FOREIGN KEY (seam_id)
        REFERENCES seam (seam_id) MATCH
        SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT thread_seam_t_id_fkey
        FOREIGN KEY (t_id)
        REFERENCES thread (t_id)
        MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
);
```

thread_seam Description

- This relationship details which *thread* is applied to which *seam*. It also holds values such as the length of the thread.
 - This is represented by the phrase “USES” in the diagram between *Seam* and *Thread*.
 - Every *Seam* must have a *thread*.
- This table relates the Thread entity with the Seam entity, holding the primary keys of both Thread and Seam as foreign keys. These two foreign keys together create the table’s primary key. Finally, the table carries an additional attribute representing the number of inches within each seam.

Seam Diagram: *thread_supplier* Relationship

***thread_supplier* Relationship**

Implementation

```
CREATE TABLE thread_supplier
(
  t_id integer NOT NULL,
  sup_id integer NOT NULL,
  CONSTRAINT thread_supplier_pkey
    PRIMARY KEY (t_id, sup_id),
  CONSTRAINT thread_supplier_sup_id_fkey
    FOREIGN KEY (sup_id)
      REFERENCES supplier (sup_id)
      MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE NO ACTION,
  CONSTRAINT thread_supplier_t_id_fkey
    FOREIGN KEY (t_id)
      REFERENCES thread (t_id)
      MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE NO ACTION
);
```

***thread_supplier* Description**

- This relationship is represented by the “HAS” relation in the diagram.
 - This table holds the supplier of each thread.
- A *Thread* must have a *Supplier*.
- This table relates the *Thread* entity with the *Supplier* entity, containing the primary keys of both Thread and Supplier. Combined, these two primary keys (stored as foreign keys) create the primary key of this relation. Since each thread must have at least one supplier, Thread holds the constraint ON DELETE RESTRICT.

Part 5: Processing Diagram

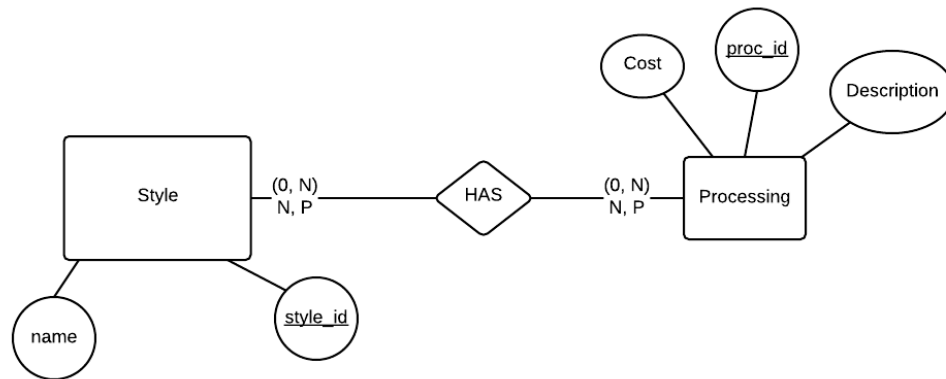


Diagram Overview

The main entities in this section are *Processing* and *Style*. A style may have additional processing costs. These are costs which are not accounted for in other sections of the database. Processing costs are stored in the processing entity along with a description of the processing done.

Processing Diagram: Processing Entity

Processing Implementation

```

CREATE TABLE Processing
(
    proc_id SERIAL PRIMARY KEY,
    cost MONEY,
    description TEXT
);
  
```

Processing Description

- The `Processing` table contains a `proc_id` as the primary key. The `cost` of the processing is an attribute with the type `MONEY`. The `description` attribute contains additional information about the processing, such as the type of special processing, and any additional instructions.

Processing Diagram: Style Entity

Style Entity Implementation

```
CREATE TABLE Style
(
    style_id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL
);
```

Style Description

- Each `Style` contains a unique `style_id` as the primary key. The type for this is `SERIAL`, which is an automatically incrementing integer. Every style also has a `name` which may not be unique, but must not be `NULL`.
- The `Style` entity was created to connect all the parts of the *style*, and since each style has a unique `style_id`, this warrants the creation of `Style` as a separate entity.

Processing Diagram: Style_Processing Relationship

Style_Processing Relationship Implementation

```
CREATE TABLE Style_Processing
(
    style_id INT REFERENCES
        Style(style_id),
    proc_id INT REFERENCES
        Processing(proc_id),
    PRIMARY KEY (style_id, proc_id)
);
```

Style_Processing Description

- This table contains foreign keys from the `Style` and `Processing` tables to relate the two entities in a many-to-many relationship. The two foreign keys, `style_id` and `proc_id`, are combined to form a composite key.

Part 6: FOB

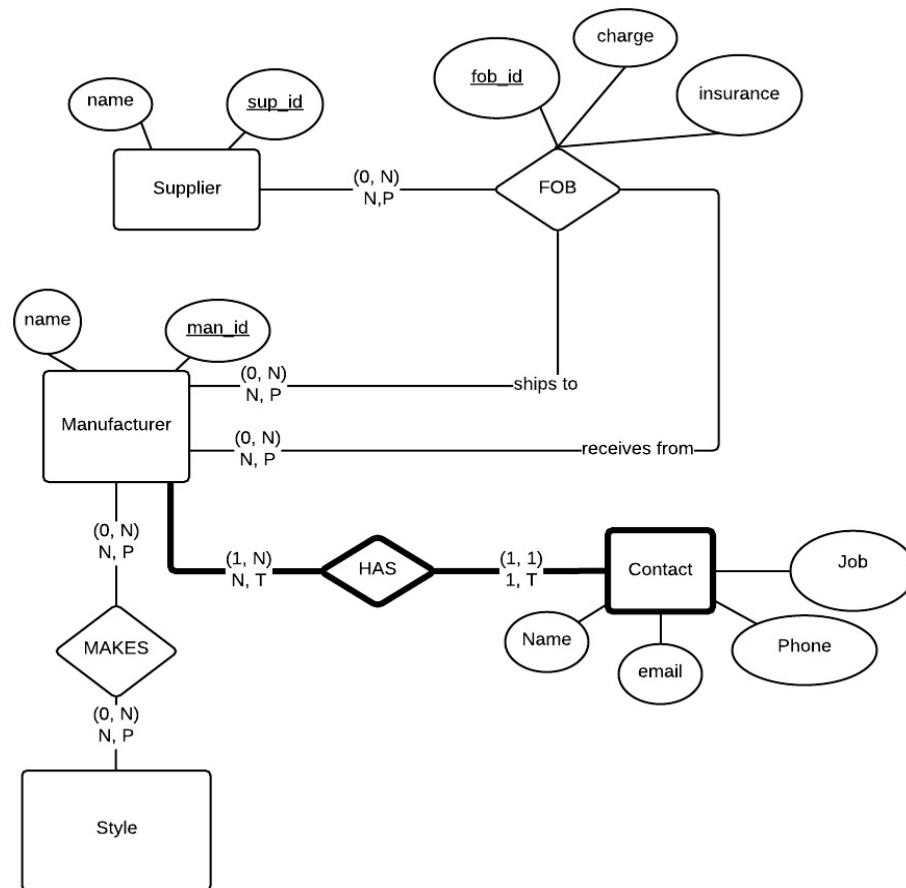


Diagram Overview

The central entities here are *Supplier*, *Manufacturer*, and *Contact*. A style is made by a manufacturer. Manufacturers receive items, such as notions, from suppliers to begin creating the style. This relationship is expressed through the term FOB. It represents the costs between a manufacturer and a supplier and the cost between two different manufacturers. This detail is essential in calculating the total cost per garment. Also, manufacturers must have contact information.

FOB Diagram: Manufacturer Entity

Manufacturer Implementation

```
CREATE TABLE Manufacturer
(
    man_id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

Manufacturer Description

- Every `Manufacturer` contains a `man_id` as the primary key and a `name`, which must not be NULL.

FOB Diagram: Supplier Entity

Supplier Implementation

```
CREATE TABLE Supplier
(
    sup_id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL
);
```

Supplier Description

- The `Supplier` table has a `sup_id` attribute, which is the primary key. This table also has a `name` attribute reserved for the name of `supplier`. The name attribute has a NOT NULL constraint because in real life all suppliers must have a name.

FOB Diagram: Contact Entity

Contact Implementation

```
CREATE TABLE Contact
(
    man_id INT REFERENCES
        Manufacturer(man_id)
        ON DELETE RESTRICT,
    name VARCHAR(50) NOT NULL,
    phone VARCHAR(20)
        NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    job VARCHAR(50),
    PRIMARY KEY (man_id, email)
);
```

Contact Description

- The `Contact` table contains a reference to the primary key of the related manufacturer. Every manufacturer must contain at least one contact, so the integrity constraint ON DELETE RESTRICT has been added to the foreign key, disallowing delete queries. `Contact name` is the name of the person who is the contact, and must not be NULL. They have a `phone` and `email` which both must be unique and NOT NULL.
- Contact is a weak entity, so its primary key consists of the related manufacturer id, and the contact's email.

FOB Diagram: Fob Relationship

Fob Implementation

```
CREATE TABLE Fob
(
  sup_id INT REFERENCES
    Supplier(sup_id),
  man_id INT REFERENCES
    Manufacturer(man_id),
  PRIMARY KEY(sup_id, man_id ),
  charge MONEY NOT NULL,
  insurance INT NOT NULL
);
```

Fob Description

- This table has two foreign keys, `sup_id` and `man_id`. The `sup_id` key references the primary key of Supplier and `man_id` key references the primary key of Manufacturer. The primary key of this relation is a composite key made up of `man_id` and `sup_id`. the Fob Table has `charge` and `insurance` attributes, both which have NOT NULL constraint.

FOB Diagram: Style_Manufacturer Relationship

Style_Manufacturer Implementation

```
CREATE TABLE Style_Manufacturer
(
  style_id INT REFERENCES
    Style(style_id),
  man_id INT REFERENCES
    Manufacturer(man_id),
  PRIMARY KEY (style_id, man_id)
);
```

Style_Manufacturer Description

- This table links a style to one or more manufacturers, so it contains the primary keys of the style table and manufacturer table.

Photo Diagram: Model Entity

Model Entity Implementation>

```
CREATE TABLE model
(
  model_id serial NOT NULL,
  agency character(50),
  name character(50),
  CONSTRAINT model_pkey
    PRIMARY KEY (model_id)
);
```

Model Description

- *Model* holds a list of human fashion models who wear various garments for photo shoots. They essentially act as a reference for others to judge clothing styles.
- The Model table contains the `agency` of the model, `name` of the model, and a primary key called `model_id`.

Photo Diagram: Photo Entity

Photo Entity Implementation

```
CREATE TABLE photo
(
  photo_id serial NOT NULL,
  location character(50),
  description text,
  date date,
  photographer character(50),
  model_id integer,
  image character varying(100),
  CONSTRAINT photo_pkey
    PRIMARY KEY (photo_id),
  CONSTRAINT photo_model_id_fkey
    FOREIGN KEY (model_id)
      REFERENCES model (model_id)
      MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE NO ACTION
);
```

Photo Description

- *Styles* from various sorts of assets involved in making the garment has a *Photo* taken with it.
- The *Photo* entity also holds the reference the *Model* with the model's ID. The relationship between the two is shown as "appeared in" in the diagram.
 - Each *Photo* needs a *Model* for it to be relevant in reviewing a garment piece.
- Each Photo has a `location`, `description`, `date`, `photographer`, `image`. It also has a foreign key called `model_id` which references the model who is in the photo. The primary key is `photo_id`.

Photo Diagram: style_photo Relationship

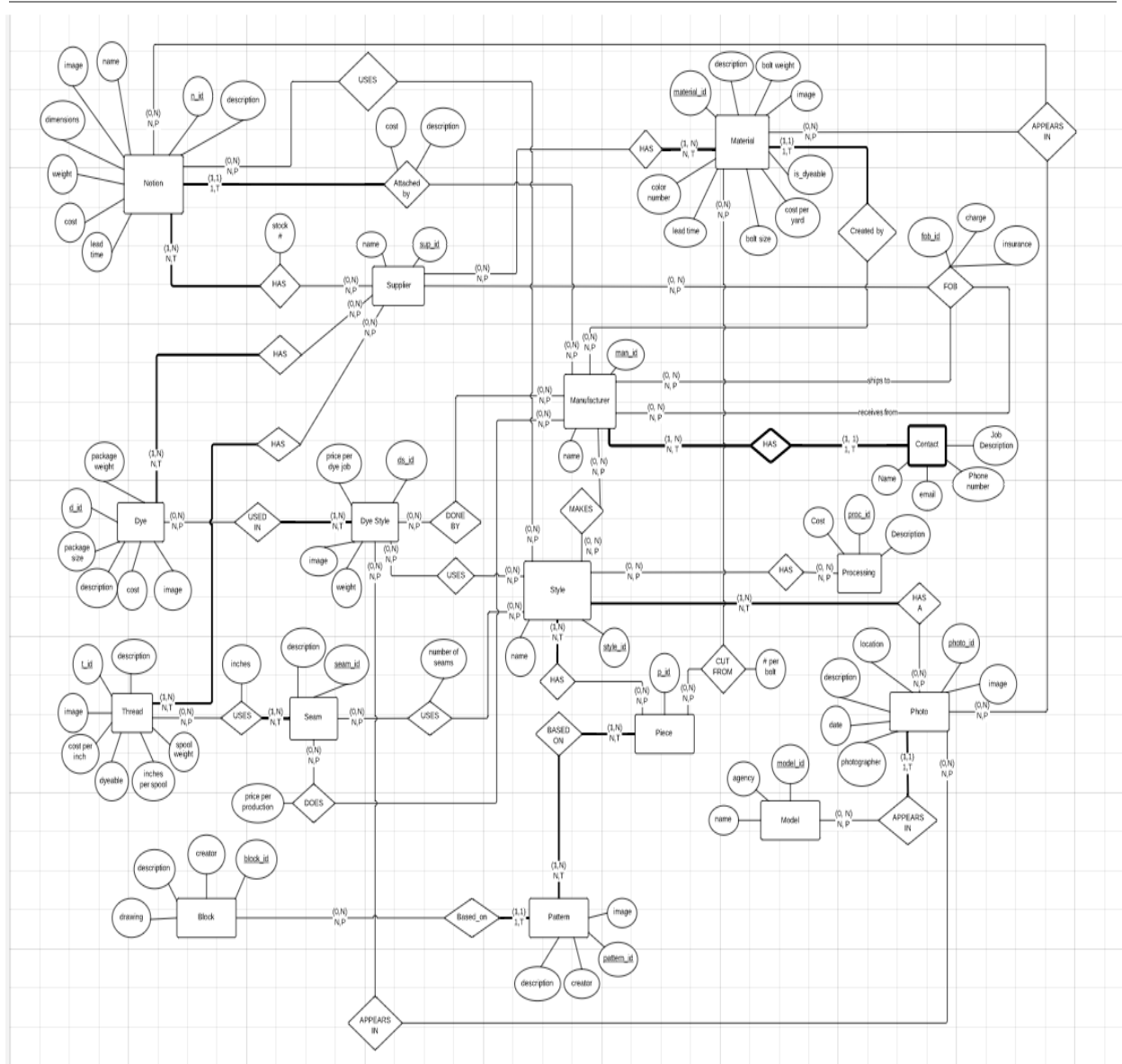
style_photo Relationship Implementation

```
CREATE TABLE style_photo
(
  style_id integer NOT NULL,
  photo_id integer NOT NULL,
  CONSTRAINT style_photo_pkey
    PRIMARY KEY (style_id, photo_id),
  CONSTRAINT style_photo_photo_id_fkey
    FOREIGN KEY (photo_id)
      REFERENCES photo (photo_id)
      MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE NO ACTION,
  CONSTRAINT style_photo_style_id_fkey
    FOREIGN KEY (style_id)
      REFERENCES style (style_id)
      MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE RESTRICT
)
```

style_photo Description

- This relationship details the styles of various garments which are shown in each photo.
 - This relationship is expressed in the diagram as the “has a” relation between *Photo* and *Style*.
- The *style_photo* table links a photo to a style, so it contains the `style_id` from the style, a `photo_id` from the photo, and the primary key being the `style_id` and `photo_id` combined.

Complete ER Diagram



Planning Process and Distribution of Work

As a class, we decided to place people into groups to divide up the work. Our first job was to come up with a complete E/R diagram that the class would agree on. Each group was given the task of creating their own complete E/R diagram of the system based on the project description. The class would then share their diagrams during the next meeting and decide on the final diagram to use to start working. During the class meeting, it became clear that we would not be able to all agree on the same E/R diagram and evenly dividing up the parts of the system to each group was not possible. One group had anticipated this problem and came to the meeting with a solution already prepared.

The solution involved splitting the work of producing the complete database system and documentation into phases. The proposed teams were: the E/R Diagram team, the Create Table team, the Insert Into Table team, the QA Testing team, the Editing team, and an optional Web Interface team. We had a total of eight groups, so we decided to allocate an extra group to both the E/R Diagram team and the Create Table team because we saw those two jobs as the ones which were more important. Each team was also responsible for providing documentation of their work for the Editing team to compile and edit.

The E/R Diagram team was responsible for collecting the E/R diagrams created by each team and deciding on a final E/R diagram from which the database would be based on. They also had to create E/R schema and divide up the E/R diagram into smaller parts for readability.

The Create Table team was responsible for turning the E/R diagram and schema into tables in the database. They wrote CREATE TABLE statements to create the structure of the database and documented their work to allow other teams to understand the structure.

The Insert Into Table team was responsible for populating the tables in the database with data. They did this by writing INSERT INTO queries and documenting their work so that other teams would be able to use and test the data.

The QA team was responsible for writing up a test plan and executing it on the populated database. They had to run detailed tests on the database to ensure the integrity of the database. They also had to record and document their tests and results.

The Editing team was responsible for compiling the documentation provided by the other teams and editing them into one document. They had to format and edit the document for consistency and professionalism.

The Web Interface team was responsible for designing and creating a CRUD web application for the working database.

Further Implementation Details: Data Input and Integrity Testing

Creation of Tables

The creation of tables for every entity and relationship sets had to be done in a certain order. This order disallows any table from referencing a non-existing table at the time of its own creation. The table creation team initialized the tables in the following way:

<u>Entity Tables</u>	<u>Relation Tables</u>
1. CREATE TABLE Style ();	17. CREATE TABLE Style_Piece ();
2. CREATE TABLE Manufacturer();	18. CREATE TABLE Style_Manufacturer ();
3. CREATE TABLE Processing ();	19. CREATE TABLE Style_Processing ();
4. CREATE TABLE Contact ();	20. CREATE TABLE Style_Photo ();
5. CREATE TABLE Model ();	21. CREATE TABLE Material_Notion_Photo();
6. CREATE TABLE Photo ();	22. CREATE TABLE Photo_DyeStyle ();
7. CREATE TABLE Dye_Style ();	23. CREATE TABLE Style_Seam ();
8. CREATE TABLE Dye ();	24. CREATE TABLE Style_Dye_Style ();
9. CREATE TABLE Thread ();	25. CREATE TABLE Dye_Style_Manufacturer();
10. CREATE TABLE Seam ();	26. CREATE TABLE Dye_Style_Dye ();
11. CREATE TABLE Notion ();	27. CREATE TABLE Seam_Manufacturer ();
12. CREATE TABLE Material ();	28. CREATE TABLE Thread_Seam ();
13. CREATE TABLE Supplier ();	29. CREATE TABLE Thread_Supplier ();
14. CREATE TABLE Piece ();	30. CREATE TABLE Style_Notion ();
15. CREATE TABLE Block ();	31. CREATE TABLE Material_Photo ();
16. CREATE TABLE Pattern ();	32. CREATE TABLE Supplier_notion();
	33. CREATE TABLE Supplier_dye ();
	34. CREATE TABLE Fob ();
	35. CREATE TABLE Piece_pattern ();
	36. CREATE TABLE Piece_material();

Insertion of Data Into the Tables

The data insertion team was tasked with inputting values into every table inside the database. Later, the website user interface team utilized the insertion team's data to create a web interface. Here is a listing of the tables as well as examples of data insertions for each:

block:

This table describes the block on which patterns are based. Field 'block_id' is the primary key which uniquely identifies each row in the table. Other fields include its description, creator and an image of what the block looks like. Below is the insert query for one row:

Insert query:

```
Insert into block(block_id,creator,description,drawing) values(1, 'Jane', describing block', 'http://sewingplums.files.wordpress.com/2011/05/lfs-women.jpg');
```

dye:

This table describes the different types of dyes that are present. Each dye has a primary key d_id which uniquely identifies it in the table. Below is the insert query for one row:

Insert query:

```
Insert into dye(d_id,package_weight,package_size,description,cost,image) values(1, 40, 50, 'dye description one', 'http://i.imgur.com/yw8l9J4.jpg?1');
```

dye_style:

This table describes the different dye styles that are present. Each dye style has a primary key ds_id which uniquely identifies it in the table. Below is the insert query for one row:

Insert query:

```
Insert into dye_style(ds_id, price_per_dye,weight,image) values(1, 30, 40, 'https://img0.etsystatic.com/003/0/6268189/il_340x270.398143468_l0ne.jpg');
```

manufacturer:

This table contains information regarding the manufacturer. Each manufacturer has a primary key man_id which uniquely identifies it in the table. Below is the insert query for one row:

Insert query:

```
Insert into manufacturer(man_id, name) values(1, 'Smith Corp');
```

model:

This table contains the information regarding the model wearing the garment, such as the name of the model and the agency that the model belongs to. Each model has a model_id as the primary key. Below is the insert query for one row:

Insert query:

```
Insert into model(model_id, agency, name) values(1, 'Jane Corp Ltd.', 'John');
```

piece:

This table contains the the ids of the pieces that make up a style. Below is the insert query for one row:

Insert query:

```
Insert into piece(p_id) values(5);
```

processing:

This table contains information regarding the different kinds of processing along with their costs. Each processing has a proc_id as its primary key. Below is the insert query for one row:

Insert query:

```
Insert into processing(proc_id, cost, description) values(1, '40.50', 'sewing');
```

seam:

This table contains information regarding different types of seams. Each seam has a seam_id as its primary key. Below is the insert query for one row:

Insert query:

```
Insert into seam(seam_id, description) values(1, 'beading');
```

style:

Each garment is also known as a style. The style table consists of the style_id which is the primary key and the name of the style. Below is the insert query for one row:

Insert query:

```
Insert into style(style_id, name) values(1, 'cross-lock');
```

supplier :

This table contains data which describes the supplier. It includes a unique supplier ID, which is also the primary key of the table, and the supplier name. Below is the insert query for one row:

Insert Query:

```
insert into supplier(sup_id,name) values(1,'Melodie');
```

thread:

This table includes a detailed information about the thread. Each thread is identified by its unique t_id, which is also the primary key of the table. The other fields in the table include description, cost_per_inch, dyeable, inches_per_spool, spool, and image. Below is the insert query for one row:

Insert Query:

```
insert into thread(t_id,description,cost_per_inch,dyeable,inches_per_spool,spool,image)
values(12,black color,78,'no',450,16,http://i.imgur.com/477GBtz.jpg);
```

contact:

This table contains the contact information about every manufacturer. It uses the foreign key man_id which references the manufacturer table. It uses man_id and email as a composite key. Below is the insert query for one row:

Insert query:

```
insert into
contact(man_id,name,phone,email,job)values(1,'Kylee','1-297-402-9787','Kylee@elitecorp.
com','Textile technologist');
```

dye_style_dye

This table describes the relationship between dye_style and dye. It references the primary keys of both dye and dye_style. Below is the insert query for one row:

Insert query:

```
insert into dye_style_dye (ds_id,d_id) values(1,1);
```

dye_style_manufacturer:

This table describes the relationship between dye_style and manufacturer. It references the primary keys of both dye_style and manufacturer. Below is the insert query for one row:

Insert query:

```
insert into dye_style_manufacturer (ds_id,man_id) values(3,2);
```

fob:

This table contains shipping costs from suppliers to the garment manufacturers. It uses the foreign keys man_id and sup_id from the manufacturer and supplier tables as its composite key to uniquely identify each row. Below is the insert query for one row:

Insert query:

```
insert into fob(sup_id,man_id,charge,insurance) values(1,1,300,5000);
```

material:

This table contains information about the material and the manufacturer which created it. It uses the foreign key man_id which references manufacturer table. Each row in the table is uniquely identified by a material_id field which is the primary key of the table. Below is the insert query for one row:

Insert query:

```
insert into
material(material_id,man_id,description,color_number,bolt_size,bolt_weight,cost_per_yard,
is_dyeable,lead_time,image) values (2,4,'purple color',18,80,35,'$12.02','no','22:00:00','
http://i.imgur.com/f95WOfY.jpg');
```

material_notion_photo:

This table describes the relationship between material, notion and photo. It references the primary keys of each of the above tables as its foreign keys to form its composite key: photo_id, material_id and n_id. Below is the insert query for one row:

Insert query:

```
insert into material_notion_photo(photo_id, material_id,n_id) values(1,2,3);
```

material_photo:

This table describes the relationship between material and photo. It references the primary keys of both material and photo and uses photo_id and material as its composite key. Below is the insert query for one row:

Insert query:

```
insert into material_photo(material_id,photo_id) values(5,4);
```

notion:

This table contains the notions or attachments that are used. It references the primary key of the manufacturer table, man_id. Below is the insert query for one row:

Insert query:

```
insert into notion(n_id,man_id,name,description,dimensions,weight,cost,attach_cost
money,attach_description,lead_time,image) values(1,5,'button',3,6,'28.6', '38.4',
'description', '06:45:42', 'http://i.imgur.com/qFJu2Cr.jpg' );
```

pattern:

This table describes the details regarding the pattern. It references the primary key of the block table, block_id. Below is the insert query for one row:

Insert query:

```
insert into pattern(pattern_id,creator, description, block_id, image ) values(1, john, 'pattern one', 1, 'http://i.imgur.com/HiuKSft.gif');
```

photo:

This table contains information regarding the photos. It references the primary key of the model table, model_id. Below is the insert query for one row:

Insert query:

```
insert into photo(photo_id, location, description, date, photographer, model_id, image) values(1, 'New york', 'describing one', '2012-01-27', 'john', 1, 'http://i.imgur.com/HiuKSft.gif');
```

photo_Dyestyle:

This table describes the relationship between the photo and dye_style tables. It references the primary key of the photo table, photo_id, and the primary key of the dye_style table, ds_id. Below is the insert query for one row:

Insert query:

```
insert into photo_dyestyle(photo_id, ds_id) values (1,2);
```

piece_material:

This table describes the relationship between the piece table and the material table. It references the primary key of the piece table, p_id, and the primary key of the material table, material_id. Below is the insert query for one row:

Insert query:

```
insert into piece_material(p_id, material_id, number_per_bolt) values(1, 2, 5);
```

piece_pattern:

This table describes the relationship between the piece table and the pattern table. It references the primary key of the piece table, p_id, and the primary key of the pattern table, pattern_id. Below is the insert query for one row:

Insert query:

```
insert into piece_pattern(p_id, pattern_id) values(1,2);
```

seam_manufacturer:

This table describes the relationship between the seam table and the manufacturer table. It references the primary key of the seam table, seam_id, and the primary key of the manufacturer table, man_id. Below is the insert query for one row:

Insert query:

```
insert into seam_manufacturer(seam_id, man_id, price_per_prod money) values(1,3, '20.50');
```

style_dye_style:

This table describes the relationship between the style table and the dye_style table. It references the primary key of the style table, style_id, and the primary key of the dye_style table, ds_id. Below is the insert query for one row:

Insert query:

```
insert into style_dye_style(style_id, ds_id) values(1,3);
```

style_manufacturer:

This table describes the relationship between the style table and the manufacturer table. It references the primary key of the style table, style_id, and the primary key of the manufacturer table, man_id. Below is the insert query for one row:

Insert query:

```
Insert into style_manufacturer(style_id, man_id) values(3,4);
```

style_notion:

This table describes the relationship between the style table and the notion table. It references the primary key of the style table, style_id, and the primary key of the notion table, n_id. Below is the insert query for one row:

Insert query:

```
insert into style_notion(style_id, n_id) values(1,3);
```

style_photo:

This table describes the relationship between the style table and the photo table. It references the primary key of the style table, style_id, and the primary key of the photo table, photo_id. Below is the insert query for one row:

Insert query:

```
insert into style_photo(style_id, photo_id) values(1,3);
```

style_piece:

This table describes the relationship between the style table and the piece table. It references the primary key of the style table, style_id, and the primary key of the piece table, p_id.

Below is the insert query for one row:

Insert query:

```
insert into style_piece(style_id, p_id) values(1,3);
```

style_processing:

This table describes the relationship between the style table and the processing table. It references the primary key of the style table, style_id, and the primary key of the processing table, proc_id. Below is the insert query for one row:

Insert query:

```
insert into style_processing(style_id, proc_id) values(1,3);
```

style_seam:

This table describes the relationship between the style table and the seam table. It references the primary key of the style table, style_id, and the primary key of the seam table, seam_id.

Below is the insert query for one row:

Insert query:

```
insert into style_seam(style_id, seam_id) values(1,3);
```

supplier_dye:

This table describes the relationship between the supplier table and the dye table. It references the primary key of the supplier table, sup_id, and the primary key of the dye table, d_id.

Below is the insert query for one row:

Insert query:

```
insert into supplier_dye(sup_id, d_id) values(1,3);
```

supplier_notion:

This table describes the relationship between the supplier table and the notion table. It references the primary key of the supplier table, sup_id, and the primary key of the notion table, n_id. Below is the insert query for one row:

Insert query:

```
insert into supplier_notion(sup_id, n_id) values(1,3);
```

thread_seam:

This table describes the relationship between the thread table and the seam table. It references the primary key of the thread table, t_id, and the primary key of the seam table, seam_id.

Below is the insert query for one row:

Insert query:

```
insert into thread_seam(t_id, seam_id) values(1,3);
```

thread_supplier:

This table describes the relationship between the thread table and the supplier table. It references the primary key of the thread table, t_id, and the primary key of the supplier table, sup_id. Below is the insert query for one row:

Insert query:

```
insert into thread_supplier(t_id, sup_id) values(1,3);
```

QA Team Tests and Results

The following section contains a detailed documentation of the tests performed by the QA team.

1. Foreign Keys Test:

(performed by Pratik Jaiswal)

Ensure that rows being inserted into 'Table2', which has a foreign key 'k' referenced from Table1, that do not have a matching entry in 'Table1' will not be inserted.

Table2: Foreign Key references Table1	RESULTS AFTER CHECK WITH POSTGRESQL
CONTACT: man_id references Manufacturer	fashion=> insert into contact values (21, 'jam', '2222', 'p@l.com', 'manager'); ERROR: insert or update on table "contact" violates foreign key constraint "contact_man_id_fkey" DETAIL: Key (man_id)=(21) is not present in table "manufacturer".-----> PASSED
Photo: model_id references Model	fashion=> insert into Photo values (881, 'Japan', 'desp', '2014-06-09', 'Sam', '1010', 287); ERROR: insert or update on table "photo" violates foreign key constraint "photo_model_id_fkey" DETAIL: Key (model_id)=(1010) is not present in table "model". -----> PASSED
Notion: man_id references Manufacturer	fashion=> insert into Notion values (33, 21, 'Posey', 'tempus risus', '3', '3', '\$28.53', '\$54.84', 'Nullaaliquet', '11:11:11'); ERROR: insert or update on table "notion" violates foreign key constraint "notion_man_id_fkey" DETAIL: Key (man_id)=(21) is not present in table "manufacturer". -----> PASSED
Material: man_id references Manufacturer	fashion=> insert into Material values (199, 1212, 'black', 10, '77', '39', '\$81.73', 't', '0011', '19:00:00'); ERROR: insert or update on table "material" violates foreign key constraint "material_man_id_fkey" DETAIL: Key (man_id)=(1212) is not present in table "manufacturer". -----> PASSED
Pattern: block_id references Block	fashion=> insert into Pattern values (1111, 'justo', 'vulputate', 'lacus. Cras', 1109); ERROR: insert or update on table "pattern" violates foreign key constraint "pattern_block_id_fkey" DETAIL: Key (block_id)=(1109) is not present in table "block". -----> PASSED
Style_Piece: style_id references Style	fashion=> insert into Style_piece values(31,31); ERROR: insert or update on table "style_piece" violates foreign key constraint "style_piece_style_id_fkey"

	DETAIL: Key (style_id)=(31) is not present in table "style". -----> PASSED
Style_Processing: style_id references Style	fashion=> insert into Style_Processing values(84,2); ERROR: insert or update on table "style_processing" violates foreign key constraint "style_processing_style_id_fkey" DETAIL: Key (style_id)=(84) is not present in table "style". -----> PASSED
Style_Photo: style_id references Style	fashion=> insert into Style_photo values(21,21); ERROR: insert or update on table "style_photo" violates foreign key constraint "style_photo_style_id_fkey" DETAIL: Key (style_id)=(21) is not present in table "style". -----> PASSED
Material_Notion_Photo: photo_id references Photo	fashion=> insert into Material_Notion_Photo values(100,9,3); ERROR: insert or update on table "material_notion_photo" violates foreign key constraint "material_notion_photo_photo_id_fkey" DETAIL: Key (photo_id)=(100) is not present in table "photo". -----> PASSED
Photo_DyeStyle: photo_id references Photo	fashion=> insert into Photo_DyeStyle values(67,9); ERROR: insert or update on table "photo_dyestyle" violates foreign key constraint "photo_dyestyle_photo_id_fkey" DETAIL: Key (photo_id)=(67) is not present in table "photo". -----> PASSED
Style_Seam: style_id references Style	fashion=> insert into Style_Seam values(31,31,2); ERROR: insert or update on table "style_seam" violates foreign key constraint "style_seam_style_id_fkey" DETAIL: Key (style_id)=(31) is not present in table "style". -----> PASSED
Style_Dye_Style: style_id references Style	fashion=> insert into Style_dye_style values(31,31); ERROR: insert or update on table "style_dye_style" violates foreign key constraint "style_dye_style_style_id_fkey" DETAIL: Key (style_id)=(31) is not present in table "style". -----> PASSED
Dye_Style_Manufacturer: ds_id references Dye_Style	fashion=>insert into Dye_Style_Manufacturer values (110,5); ERROR: insert or update on table "dye_style_manufacturer" violates foreign key constraint "dye_style_manufacturer_ds_id_fkey" DETAIL: Key (ds_id)=(110) is not present in table "dye_style". -----> PASSED
Dye_Style_Manufacturer: man_id references Manufacturer	fashion=> insert into Dye_Style_Manufacturer values (3,21); ERROR: insert or update on table "dye_style_manufacturer" violates foreign key constraint "dye_style_manufacturer_man_id_fkey" DETAIL: Key (man_id)=(21) is not present in table "manufacturer". -----> PASSED
Dye_Style_Dye: ds_id references Dye_Style	fashion=> insert into Dye_Style_Manufacturer values(333,2);

	ERROR: insert or update on table "dye_style_manufacturer" violates foreign key constraint "dye_style_manufacturer_ds_id_fkey" DETAIL: Key (ds_id)=(333) is not present in table "dye_style". -----> PASSED
Seam_Manufacturer: seam_id references Seam	fashion=> insert into Seam_Manufacturer values (83, 5, '\$5.42'); ERROR: insert or update on table "seam_manufacturer" violates foreign key constraint "seam_manufacturer_seam_id_fkey" DETAIL: Key (seam_id)=(83) is not present in table "seam". -----> PASSED
Thread_Seam: seam_id references Seam	fashion=> insert into Thread_Seam values(2, 14, 2.3); ERROR: insert or update on table "thread_seam" violates foreign key constraint "thread_seam_seam_id_fkey" DETAIL: Key (seam_id)=(14) is not present in table "seam". -----> PASSED
Thread_Supplier: t_id references Thread	fashion=> insert into Thread_Supplier values(25,21); ERROR: insert or update on table "thread_supplier" violates foreign key constraint "thread_supplier_t_id_fkey" DETAIL: Key (t_id)=(25) is not present in table "thread". -----> PASSED
Thread_Supplier: sup_id references Supplier	fashion=> insert into Thread_Supplier values(2,21); ERROR: insert or update on table "thread_supplier" violates foreign key constraint "thread_supplier_sup_id_fkey" DETAIL: Key (sup_id)=(21) is not present in table "supplier". -----> PASSED
Style_Notion: style_id references Style	fashion=> insert into Style_Notion values(25,3); ERROR: insert or update on table "style_notion" violates foreign key constraint "style_notion_style_id_fkey" DETAIL: Key (style_id)=(25) is not present in table "style". -----> PASSED
Style_Notion: n_id references Notion	fashion=> insert into Style_Notion values(3,28); ERROR: insert or update on table "style_notion" violates foreign key constraint "style_notion_n_id_fkey" DETAIL: Key (n_id)=(28) is not present in table "notion". -----> PASSED
Material_Photo: material_id references Material	fashion=> insert into Material_Photo values(35,1); ERROR: insert or update on table "material_photo" violates foreign key constraint "material_photo_material_id_fkey" DETAIL: Key (material_id)=(35) is not present in table "material". -----> PASSED
Material_Photo: photo_id references Photo	fashion=> insert into Material_Photo values(7,62); ERROR: insert or update on table "material_photo" violates foreign key constraint "material_photo_photo_id_fkey" DETAIL: Key (photo_id)=(62) is not present in table "photo". -----> PASSED

Supplier_notion: sup_id references Supplier	fashion=> insert into Supplier_notion values(8,39,8); ERROR: insert or update on table "supplier_notion" violates foreign key constraint "supplier_notion_sup_id_fkey" DETAIL: Key (sup_id)=(39) is not present in table "supplier". -----> PASSED
Supplier_dye: sup_id references Supplier	fashion=> insert into Supplier_dye values (27,4); ERROR: insert or update on table "supplier_dye" violates foreign key constraint "supplier_dye_sup_id_fkey" DETAIL: Key (sup_id)=(27) is not present in table "supplier". -----> PASSED
Fob: sup_id references Supplier	fashion=> insert into Fob values(87, 1, '\$300.00', 5000); ERROR: insert or update on table "fob" violates foreign key constraint "fob_sup_id_fkey" DETAIL: Key (sup_id)=(87) is not present in table "supplier". -----> PASSED
Fob: man_id references Manufacturer	fashion=> insert into Fob values(7, 91, '\$60.00', 100); ERROR: insert or update on table "fob" violates foreign key constraint "fob_man_id_fkey" DETAIL: Key (man_id)=(91) is not present in table "manufacturer". -----> PASSED
Piece_pattern: p_id references Piece	fashion=> insert into Piece_pattern values(104,3); ERROR: insert or update on table "piece_pattern" violates foreign key constraint "piece_pattern_p_id_fkey" DETAIL: Key (p_id)=(104) is not present in table "piece". -----> PASSED
Piece_pattern: pattern_id references Pattern	fashion=> insert into Piece_pattern values(9,108); ERROR: insert or update on table "piece_pattern" violates foreign key constraint "piece_pattern_pattern_id_fkey" DETAIL: Key (pattern_id)=(108) is not present in table "pattern". -----> PASSED
Piece_material: p_id references Piece	fashion=> insert into Piece_material values(40,39,38); ERROR: insert or update on table "piece_material" violates foreign key constraint "piece_material_material_id_fkey" DETAIL: Key (material_id)=(39) is not present in table "material". -----> PASSED
Piece_material: material_id references Material	fashion=> insert into Piece_material values(3, 104, 8); ERROR: insert or update on table "piece_material" violates foreign key constraint "piece_material_material_id_fkey" DETAIL: Key (material_id)=(104) is not present in table "material". -----> PASSED

2. Primary key constraints, Data type, Not Null constraints Test:

(performed by Luv Ahuja)

Table Name	Test performed	Result
block	Not Null Constraint insert into block values (1001,NULL,1,1);	ERROR: null value in column "creator" violates not-null constraint Result: Passed
Block	Primary key constraint insert into block values (1,1,1,1);	ERROR: duplicate key value violates unique constraint "block_pkey" Result: Passed
Pattern	Not Null constraint insert into pattern values (10000,NULL,1,1,987);	ERROR: null value in column "creator" violates not-null constraint Result: Passed
Pattern	Primary key insert into pattern values (1,8,1,1,987);	ERROR: duplicate key value violates unique constraint "pattern_pkey" Result: Passed
Piece	Insert into piece values (1);	ERROR: duplicate key value violates unique constraint "piece_pkey" Result: Passed
piece_pattern	Insert into piece_pattern values (1,68);	ERROR: duplicate key value violates unique constraint "piece_pattern_pkey" Result: Passed
piece_material	Insert into piece_material values (1,1,1);	ERROR: duplicate key value violates unique constraint "piece_material_pkey" Result: Passed
Supplier	insert into supplier values (1,1);	ERROR: duplicate key value violates unique constraint "supplier_pkey" Result: Passed

Style	Insert into style values (1,'test');	ERROR: duplicate key value violates unique constraint "style_pkey" Result: Passed
Manufacturer	Insert into manufacturer values (1,'test_data');	ERROR: duplicate key value violates unique constraint "manufacturer_pkey" Result: Passed
Processing	Insert into Processing values (1,23,'this is a test data');	ERROR: duplicate key value violates unique constraint "processing_pkey" Result: Passed
Contact	Insert into contact values (1000,NULL,4444444444,'test@email.com','tester');	ERROR: null value in column "name" violates not-null constraint Result: Passed
	Insert into contact values (1000,'tester_1',4444444444,NULL,'tester');	ERROR: null value in column "email" violates not-null constraint Result: Passed
	Insert into contact values (1000,'tester_1',NULL,'test@email.com','tester');	ERROR: null value in column "phone" violates not-null constraint Result: Passed
Style_Piece	Insert into Style_Piece values (1,1);	ERROR: duplicate key value violates unique constraint "style_piece_pkey" Result: Passed
Style_Manufacturer	Insert into Style_Manufacturer values (1,3);	ERROR: duplicate key value violates unique constraint "style_manufacturer_pkey" Result: Passed

Style_Processing	Insert into Style_Processing values (1,1);	ERROR: duplicate key value violates unique constraint "style_processing_pkey" Result: Passed
Photo	Insert into Photo values (1,'xyz','test','89-989-9999','ss','sss');	ERROR: date/time field value out of range: "89-989-9999" Result: Passed
	Insert into Photo values (1,'xyz','test','10-10-2005','ss','sss');	ERROR: invalid input syntax for integer: "sss" Result: Passed
	Insert into Photo values (1,'xyz','test','10-10-2005','ss',1);	ERROR: duplicate key value violates unique constraint "photo_pkey" Result: Passed
Style_Photo	Insert into Style_Photo values (1,1);	ERROR: duplicate key value violates unique constraint "style_photo_pkey" Result: Passed
Model	Insert into Model values (1,'test','text');	ERROR: duplicate key value violates unique constraint "model_pkey" Result: Passed
Material_Notion_Photo	Insert into Material_Notion_Photo values (1,2,3);	ERROR: duplicate key value violates unique constraint "material_notion_photo_pkey" DETAIL: Key (photo_id, material_id, n_id)=(1, 2, 3) already exists. Result: Passed
Photo_DyeStyle	Insert into Photo_DyeStyle values (1,23);	ERROR: duplicate key value violates unique constraint "photo_dyestyle_pkey" Result: Passed

Dye_Style	INSERT INTO Dye_Style values (1,23,'w',11);	ERROR: duplicate key value violates unique constraint "dye_style_pkey" Result: Passed
Dye	Insert into Dye values (1,NULL,'ss','this is a text',5555,'image');	ERROR: null value in column "package_weight" violates not-null constraint Result: Passed
	Insert into Dye values (1,'ss',NULL,'this is a text',5555,'image');	ERROR: null value in column "package_size" violates not-null constraint Result: Passed
	Insert into Dye values (1,'ss','hello','this is a text',NULL,'image');	ERROR: null value in column "cost" violates not-null constraint Result: Passed
	Insert into Dye values (1,'ss','hello','this is a text',34,'image');	ERROR: duplicate key value violates unique constraint "dye_pkey" Result: Passed
Thread	Insert into Thread values (1,'this is a text','image',30,4,3.00,3);	ERROR: invalid input syntax for type money: "image" Result: Passed
	Insert into Thread values (1,'this is a text',30,4,3.00,3,'image');	ERROR: column "dyeable" is of type boolean but expression is of type integer Result: Passed
	Insert into Thread values (1,'this is a text',30,TRUE,3.00,3,'image');	ERROR: duplicate key value violates unique constraint "thread_pkey" Result: Passed
Seam	Insert into Seam values (1,'thi is a text');	ERROR: duplicate key value violates unique constraint "seam_pkey" Result: Passed

Style_Seam	Insert into Style_Seam values (1,1,20000);	ERROR: duplicate key value violates unique constraint "style_seam_pkey" Result: Passed
	Insert into Style_Seam values (1,1222,NULL);	ERROR: null value in column "num_seams" violates not-null constraint Result: Passed
Style_Dye_Style	Insert into Style_Dye_Style values (9,80);	ERROR: duplicate key value violates unique constraint "style_dye_style_pkey" Result: Passed
Dye_Style_Manufacturer	Insert into Dye_Style_Manufacturer values (1,1);	ERROR: duplicate key value violates unique constraint "dye_style_manufacturer_pkey" Result: Passed
Dye_Style_Dye	Insert into Dye_Style_Dye values (1,1);	ERROR: duplicate key value violates unique constraint "dye_style_dye_pkey" Result: Passed
Seam_Manufacturer	Insert into Seam_Manufacturer values (1,1,NULL);	ERROR: null value in column "price_per_prod" violates not-null constraint Result: Passed
	Insert into Seam_Manufacturer values (8,20,222);	ERROR: duplicate key value violates unique constraint "seam_manufacturer_pkey" Result: Passed

Thread_Seam	INSERT INTO Thread_Seam values (1,1,NULL);	ERROR: null value in column "inches" violates not-null constraint Result: Passed
	Insert into thread_seam values (12,13,23);	ERROR: duplicate key value violates unique constraint "thread_seam_pkey" Result: Passed
Thread_Supplier	Insert into Thread_Supplier values (12,13);	ERROR: duplicate key value violates unique constraint "thread_supplier_pkey" Result: Passed
Style_Notion	Insert into Style_Notion values (1,4);	ERROR: duplicate key value violates unique constraint "style_notion_pkey" Result: Passed
Notion	Insert into Notion values (1,1,NULL,'text','image',12,12,12,12,2,2);	ERROR: column "lead_time" is of type time without time zone but expression is of type integer Result: Passed
	Insert into Notion values (1,1,NULL,'text','image',12,12,12,12,'12:00',2);	ERROR: null value in column "name" violates not-null constraint Result: Passed
	Insert into Notion values (1,1,'NAME','text','image',12,12,12,12,'12:00',2);	ERROR: duplicate key value violates unique constraint "notion_pkey" Result: Passed
Material_Photo	Insert into Material_Photo values (1,1);	ERROR: duplicate key value violates unique constraint "material_photo_pkey" Result: Passed

3. Test for Delete statements:

(performed by Kumari Sweta)

Table	Foreign Key	Table Referenced	Test case description	SQL Statement(s)	Actual result	Expected Result
Contact	man_id	Manufacturer	The record in "Manufacturer" table can't be deleted because its man_id is referenced from table "contact"	DELETE FROM manufacturer WHERE man_id = 3;	Failed	Failed
			Delete Contact record first and Manufacturer record should still exist	DELETE FROM contact WHERE man_id = 3;	Record deleted from contact	Record deleted from Contact
				SELECT * FROM manufacturer WHERE man_id = 3;	Record exists in Manufacturer	Record exists in Manufacturer
Photo	model_id	Model	The record in "Model" table can't be deleted because its model_id is still referenced from table "photo"	DELETE FROM model WHERE model_id = 3;	Failed	Failed
Notion	man_id	Manufacturer	The record in "manufacturer" table can't be deleted because its man_id is referenced from "notion"	DELETE FROM manufacturer WHERE man_id = 3;	Failed	Failed
Material	man_id	Manufacturer	The record in "manufacturer" table can't be deleted because its man_id is referenced from "material"	DELETE FROM manufacturer WHERE man_id = 2;	Failed	Failed

Pattern	block_id	Block	The record in "block" table can't be deleted because its block_id is referenced from table "pattern"	DELETE FROM block WHERE block_id = 4;	Failed	Failed
			The record from "pattern" can be deleted but block_id still exists in "block".	DELETE FROM pattern WHERE block_id = 4;	Record deleted from pattern	Record deleted from pattern
				SELECT * FROM block WHERE block_id = 4;	Record exists in Block	Record exists in Block
Style_Piece	style_id	Style	The record in "style" table can't be deleted because its style_id is referenced from table "style_piece"	DELETE FROM style WHERE style_id = 6;	Failed	Failed
			The record from "style_piece" can be deleted but its style_id will still exist in "style".	DELETE FROM style_piece WHERE style_id = 2	Record deleted from style piece	Record deleted from style piece
				SELECT * FROM style WHERE style_id = 2;	Record exists in Style	Record exists in Style
Style_Piece	p_id	Piece	The record in "piece" can't be deleted because its p_id is referenced from table "style_piece"	DELETE FROM piece WHERE p_id = 2	Failed	Failed
			The record from "style_piece" can be deleted but its p_id still exists in "piece".	DELETE FROM style_piece WHERE p_id = 1	Record deleted from style piece	Record deleted from style piece

				SELECT * FROM piece WHERE p_id = 1;	Record exists in Piece	Record exists in Piece
Style_Processing	style_id	Style	The record in "style" table can't be deleted because its style_id is referenced from table "style_process ing"	DELETE FROM style WHERE style_id = 1;	Failed	Failed
			The record from "style_process ing" can be deleted but its style_id will still exist in "style".	DELETE FROM style_processing WHERE style_id = 4;	Record deleted from style processing	Record deleted from style processing
				SELECT * FROM style WHERE style_id = 4;	Record exists in Style	Record exists in Style
Style_Processing	proc_id	Processing	The record in "processing" table can't be deleted because its proc_id is referenced from table "style_process ing"	DELETE FROM processing WHERE proc_id = 1;	Failed	Failed
			The record from "style_process ing" can be deleted but its proc_id still exists in "processing".	DELETE FROM style_processing WHERE proc_id = 16;	Record deleted from style processing	Record deleted from style processing
				SELECT * FROM processing WHERE proc_id = 16;	Record exists in Processing	Record exists in Processing
Style_Photo	style_id	Style	The record in "style" table can't be deleted because its style_id is referenced from table "style_photo"	DELETE FROM style WHERE style_id = 10;	Failed	Failed

			The record from "style_photo" can be deleted but its style_id still exists in "style".	DELETE FROM style_photo WHERE style_id = 11;	Record deleted from style photo	Record deleted from style photo
				SELECT * FROM style WHERE style_id = 11;	Record exists in Style	Record exists in Style
Style_Photo	photo_id	Photo	The record in "photo" table can't be deleted because its photo_id is referenced from table "style_photo"	DELETE FROM photo WHERE photo_id = 9;	Failed	Failed
			Delete style_photo record first and Photo record should still exist	DELETE FROM style_photo where photo_id = 15	Record deleted from style photo	Record deleted from style photo
				SELECT * FROM photo WHERE photo_id = 15;	Record exists in Photo	Record exists in Photo
Material_Notion_Photo	photo_id	Photo	The record in "photo" table can't be deleted because its photo_id is referenced from table "material_notion_photo"	DELETE FROM photo WHERE photo_id = 9;	Failed	Failed
			Delete material_notion_photo record first and Photo record should still exist	DELETE FROM material_notion_photo WHERE photo_id = 7	Record deleted from material_notion_photo	Record deleted from material_notion_photo
				SELECT * FROM photo WHERE photo_id = 7;	Record exists in Photo	Record exists in Photo
Material_Notion_Photo	material_id	Material	The record in "material" table can't be deleted because its material_id is referenced	DELETE FROM Material WHERE material_id = 10;	Failed	Failed

			from table "material_noti on_photo"			
			Delete material_notio n_photo record first but Material record should still exist	DELETE FROM Material_notion_ph oto WHERE material_id = 4;	Record deleted from material_notion_ph oto	Record deleted from material_notion_ph oto
				SELECT * FROM material WHERE material_id = 4;	Record exists in Material	Record exists in Material
Material_Notion_P hoto	n_id	Notion	The record in "Notion" table can't be deleted because its n_id is referenced from table "material_noti on_photo"	DELETE FROM notion WHERE n_id = 4;	Failed	Failed
			Delete material_notio n_photo record first but Notion record should still exist	DELETE FROM Material_notion_ph oto WHERE n_id = 14;	Record deleted from material_notion_ph oto	Record deleted from material_notion_ph oto
				SELECT * from notion WHERE n_id = 14;	Record exists in Notion	Record exists in Notion
Photo_DyeStyle	photo_id	Photo	The record in "Photo" table can't be deleted because its photo_id is referenced from table "photo_dyestyl e"	DELETE FROM photo WHERE photo_id = 14;	Failed	Failed
			Delete Photo_DyeSty le record first but Photo record should still exist	DELETE FROM photo_dyestyle WHERE photo_id = 10;	Record deleted from Photo_DyeStyle	Record deleted from Photo_DyeStyle
				SELECT * from photo WHERE photo_id = 10;	Record exists in Photo	Record exists in Photo
Photo_DyeStyle	ds_id	Dye_Style	The record in "Dye_Style" table can't be deleted	DELETE FROM dye_style WHERE photo_id = 14;	Failed	Failed

			because its ds_id is referenced from "photo_dyestyl e"			
			Delete Photo_DyeSty le record first but Dye_Style record should still exist	DELETE FROM photo_dyestyle WHERE ds_id = 11;	Record deleted from Photo_DyeStyle	Record deleted from Photo_DyeStyle
				SELECT * from dye_style WHERE ds_id = 11;	Record exists in Dye_Style	Record exists in Dye_Style
Style_Seam	style_id	Style	The record in "Style" can't be deleted because its style_id is referenced from "Style_Seam"	DELETE FROM style WHERE style_id = 20;	Failed	Failed
			Delete Style_Seam record first but Style record should still exist	DELETE FROM style_seam WHERE style_id = 5	Record deleted from Style_Seam	Record deleted from Style_Seam
				SELECT * from style WHERE style_id = 5;	Record exists in Style	Record exists in Style
Style_Seam	seam_id	Seam	The record in "Seam" can't be deleted because its seam_id is referenced from "Style_Seam"	DELETE FROM seam WHERE seam_id = 6;	Failed	Failed
			Delete Style_Seam record first but Seam record should still exist	DELETE FROM style_seam WHERE seam_id = 8;	Record deleted from Style_Seam	Record deleted from Style_Seam
				SELECT * FROM seam WHERE seam_id = 8;	Record exists in Seam	Record exists in Seam
Style_Dye_Style	style_id	Style	The record in "Style" can't be deleted because its style_id is referenced from	DELETE FROM style WHERE style_id = 6;	Failed	Failed

			"Style_Dye_Style"			
			Delete Style_Dye_Style record first but Style record should still exist	DELETE FROM style_dye_style WHERE style_id = 10;	Record deleted from Style_Dye_Style	Record deleted from Style_Dye_Style
				SELECT * FROM style WHERE style_id = 10;	Record exists in Style	Record exists in Style
Style_Dye_Style	ds_id	Dye_Style	The record in "Dye_Style" can't be deleted because its ds_id is referenced from "Style_Dye_Style"	DELETE FROM dye_style WHERE ds_id = 6;	Failed	Failed
			Delete Style_Dye_Style record first but Dye_Style record should still exist	DELETE FROM style_dye_style WHERE ds_id = 11	Record deleted from Style_Dye_Style	Record deleted from Style_Dye_Style
				SELECT * FROM dye_style WHERE ds_id = 11;	Record exists in Dye_Style	Record exists in Dye_Style
Dye_Style_Manufacturer	ds_id	Dye_Style	The record in "Dye_Style" can't be deleted because its ds_id is referenced from "Dye_Style_Manufacturer"	DELETE FROM dye_style WHERE ds_id = 6;	Failed	Failed
			Delete Dye_Style_manufacturer record first but Dye_Style record should still exist	DELETE FROM dye_style_manufacturer WHERE ds_id = 11;	Record deleted from Dye_Style_Manufacturer	Record deleted from Dye_Style_Manufacturer
				SELECT * FROM dye_style WHERE ds_id = 11;	Record exists in Dye_Style	Record exists in Dye_Style
Dye_Style_Manufacturer	man_id	Manufacturer	The record in "Manufacturer" can't be deleted because its man_id is	DELETE FROM manufacturer WHERE man_id = 3;	Failed	Failed

			referenced from "Dye_Style_Manufacturer"			
			Delete Dye_Style_manufacturer record first but Manufacture record should still exist	DELETE FROM dye_style_manufacturer WHERE man_id = 11;	Record deleted from Dye_Style_Manufacturer	Record deleted from Dye_Style_Manufacturer
				SELECT * FROM manufacturer WHERE man_id = 11;	Record exists in Manufacturer	Record exists in Manufacturer
Dye_Style_Dye	ds_id	Dye_Style	The record in "Dye_Style" can't be deleted because its ds_id is referenced from "Dye_Style_Dye"	DELETE FROM dye_style WHERE ds_id = 6;	Failed	Failed
			Delete Dye_Style_Dye record first but Dye_Style record should still exist	DELETE FROM dye_style_dye WHERE ds_id = 11;	Record deleted from Dye_Style_Dye	Record deleted from Dye_Style_Dye
				SELECT * FROM dye_style WHERE ds_id = 11;	Record exists in dye_Style	Record exists in dye_Style
Dye_Style_Dye	d_id	Dye	The record in "Dye" can't be deleted because its d_id is referenced from "Dye_Style_Dye"	DELETE FROM dye WHERE d_id = 9;	Failed	Failed
			Delete Dye_Style_Dye record first but Dye record should still exist	DELETE FROM dye_style_dye WHERE d_id = 5;	Record deleted from Dye_Style_Dye	Record deleted from Dye_Style_Dye
				SELECT * FROM dye WHERE d_id = 5;	Record exists in Dye	Record exists in Dye
Seam_Manufacturer	seam_id	Seam	The record in "Seam" can't be deleted because its	DELETE FROM seam WHERE seam_id = 10;	Failed	Failed

			seam_id is referenced from "Seam_Manufacturer"			
			Delete Seam_Manufacturer record first but Seam record should still exist	DELETE FROM seam_manufacturer WHERE seam_id = 3;	Record deleted from Seam_Manufacturer	Record deleted from Seam_Manufacturer
				SELECT * from seam WHERE seam_id = 3;	Record exists in Seam	Record exists in Seam
Seam_Manufacturer	man_id	Manufacturer	The record in "Manufacturer" can't be deleted because its man_id is referenced from "Seam_Manufacturer"	DELETE FROM manufacturer WHERE man_id = 3;	Failed	Failed
			Delete Seam_Manufacturer record first but Manufacturer record should still exist	DELETE FROM seam_manufacturer WHERE man_id = 2	Record deleted from Seam_Manufacturer	Record deleted from Seam_Manufacturer
				SELECT * from manufacturer WHERE man_id = 2;	Record exists in Manufacturer	Record exists in Manufacturer
Thread_Seam	t_id	Thread	The record in "Thread" can't be deleted because its t_id is referenced from "Thread_Seam"	DELETE FROM thread WHERE t_id = 2;	Failed	Failed
			Delete Thread_Seam record first but Thread record should still exist	DELETE FROM thread_seam WHERE t_id = 5;	Record deleted from Thread_Seam	Record deleted from Thread_Seam
				SELECT * from thread WHERE t_id = 5;	Record exists in Thread	Record exists in Thread
Thread_Seam	seam_id	Seam	The record in "Seam" can't be deleted	DELETE FROM seam WHERE seam_id = 5;	Failed	Failed

			because its seam_id is referenced from "Thread_Seam"			
			Delete Thread_Seam record first but Seam record should still exist	DELETE FROM thread_seam WHERE seam_id = 3;	Record deleted from Thread_Seam	Record deleted from Thread_Seam
				SELECT * from seam WHERE seam_id = 3;	Record exists in Seam	Record exists in Seam
Thread_Supplier	t_id	Thread	The record in "Thread" can't be deleted because its t_id is referenced from "Thread_Supplier"	DELETE FROM thread WHERE t_id = 2;	Failed	Failed
			Delete Thread_Supplier record first but Thread record should still exist	DELETE FROM thread_supplier WHERE t_id = 13;	Record deleted from Thread_Supplier	Record deleted from Thread_Supplier
				SELECT * from thread WHERE t_id = 13;	Record exists in Thread	Record exists in Thread
Thread_Supplier	sup_id	Supplier	The record in "Supplier" can't be deleted because its sup_id is referenced from "Thread_Supplier"	DELETE FROM supplier WHERE sup_id = 14;	Failed	Failed
			Delete Thread_Supplier record first but Supplier record should still exist	DELETE FROM thread_supplier WHERE sup_id = 12;	Record deleted from Thread_Supplier	Record deleted from Thread_Supplier
				SELECT * from supplier WHERE sup_id = 12;	Record exists in Supplier	Record exists in Supplier
Style_Notion	style_id	Style	The record in "Style" can't be deleted because its	DELETE FROM style WHERE style_id = 6;	Failed	Failed

			style_id is referenced from "Style_Notion"			
			Delete Style_Notion record first but Style record should still exist	DELETE FROM style_notion WHERE style_id = 12;	Record deleted from Style_Notion	Record deleted from Style_Notion
				SELECT * from style WHERE style_id = 12;	Record exists in Style	Record exists in Style
Style_Notion	n_id	Notion	The record in "Notion" table can't be deleted because its n_id is referenced from table "Style_Notion"	DELETE FROM notion WHERE n_id = 4;	Failed	Failed
			Delete Style_Notion record first but Notion record should still exist	DELETE FROM style_notion WHERE n_id = 15;	Record deleted from Style_Notion	Record deleted from Style_Notion
				SELECT * from notion WHERE n_id = 15;	Record exists in Notion	Record exists in Notion
Material_Photo	material_id	Material	The record in "Material" table can't be deleted because its material_id is referenced from table "Material_Photo"	DELETE FROM Material WHERE material_id = 10;	Failed	Failed
			Delete Material_Photo record first but Material record should still exist	DELETE FROM Material_Photo WHERE material_id = 10;	Record deleted from material_photo	Record deleted from material_photo
				SELECT * from material WHERE material_id = 10;	Record exists in Material	Record exists in Material
Material_Photo	photo_id	Photo	The record in "Photo" table can't be deleted because its photo_id is	DELETE FROM photo WHERE photo_id = 14;	Failed	Failed

			referenced from table "Material_Photo"			
			Delete Material_Photo record first but Photo record should still exist	DELETE FROM Material_Photo WHERE photo_id = 20;	Record deleted from material_photo	Record deleted from material_photo
				SELECT * from photo WHERE photo_id = 20;	Record exists in Photo	Record exists in Photo
Supplier_notion	sup_id	Supplier	The record in "Supplier" can't be deleted because its sup_id is referenced from "Supplier_notion"	DELETE FROM supplier WHERE sup_id = 14;	Failed	Failed
			Delete Supplier_notion record first but Supplier record should still exist	DELETE FROM supplier_notion WHERE sup_id = 7;	Record deleted from Supplier_notion	Record deleted from Supplier_notion
				SELECT * from supplier WHERE sup_id = 7;	Record exists in Supplier	Record exists in Supplier
Supplier_notion	n_id	Notion	The record in "Notion" table can't be deleted because its n_id is referenced from table "Supplier_Notion"	DELETE FROM notion WHERE n_id = 4;	Failed	Failed
			Delete Supplier_notion record first but Notion record should still exist	DELETE FROM supplier_notion WHERE n_id = 7;	Record deleted from Supplier_notion	Record deleted from Supplier_notion
				SELECT * from notion WHERE n_id = 7;	Record exists in Notion	Record exists in Notion
Supplier_dye	sup_id	Supplier	The record in "Supplier" can't be deleted because its	DELETE FROM supplier WHERE sup_id = 14;	Failed	Failed

			sup_id is referenced from "Supplier_dye"			
			Delete Supplier_dye record first but Supplier record should still exist	DELETE FROM supplier_dye WHERE sup_id = 7;	Record deleted from Supplier_dye	Record deleted from Supplier_dye
				SELECT * from supplier WHERE sup_id = 7;	Record exists in Supplier	Record exists in Supplier
Supplier_dye	d_id	Dye	The record in "Dye" can't be deleted because its d_id is referenced from "Supplier_dye"	DELETE FROM dye WHERE d_id = 9;	Failed	Failed
			Delete Supplier_dye record first but Dye record should still exist	DELETE FROM supplier_dye WHERE sup_id = 17;	Record deleted from Supplier_dye	Record deleted from Supplier_dye
				SELECT * from dye WHERE sup_id = 17;	Record exists in Dye	Record exists in Dye
Fob	sup_id	Supplier	The record in "Supplier" can't be deleted because its sup_id is referenced from "Fob"	DELETE FROM supplier WHERE sup_id = 14;	Failed	Failed
			Delete Fob record first but Supplier record should still exist	DELETE FROM fob WHERE sup_id = 19;	Record deleted from Fob	Record deleted from Fob
				SELECT * from supplier WHERE sup_id = 19;	Record exists in Supplier	Record exists in Supplier
Fob	man_id	Manufacturer	The record in "Manufacturer" can't be deleted because its man_id is referenced from "Fob"	DELETE FROM manufacturer WHERE man_id = 3;	Failed	Failed

			Delete Fob record first but Manufacturer record should still exist	DELETE FROM fob WHERE man_id = 19;	Record deleted from Fob	Record deleted from Fob
				SELECT * FROM manufacturer WHERE man_id = 19;	Record exists in manufacturer	Record exists in manufacturer
Piece_pattern	p_id	Piece	The record in "Piece" can't be deleted because its p_id is referenced from "Piece_pattern"	DELETE FROM piece WHERE p_id = 19;	Failed	Failed
			Delete Piece_pattern record first but Piece record should still exist	DELETE FROM piece_pattern WHERE p_id = 13;	Record deleted from Piece_pattern	Record deleted from Piece_pattern
				SELECT * FROM piece WHERE p_id = 13;	Record exists in Piece	Record exists in Piece
Piece_pattern	pattern_id	Pattern	The record in "Pattern" can't be deleted because its pattern_id is referenced from "Piece_pattern"	DELETE FROM pattern WHERE pattern_id = 3;	Failed	Failed
			Delete Piece_pattern record first but Pattern record should still exist	DELETE FROM piece_pattern WHERE pattern_id = 7;	Record deleted from Piece_pattern	Record deleted from Piece_pattern
				SELECT * FROM pattern WHERE pattern_id = 17;	Record exists in Pattern	Record exists in Pattern
Piece_material	p_id	Piece	The record in "Piece" can't be deleted because its p_id is referenced from "Piece_material"	DELETE FROM piece WHERE p_id = 19;	Failed	Failed

			Delete Piece_material I record first but Piece record should still exist	DELETE FROM piece_material WHERE p_id = 2;	Record deleted from Piece_material	Record deleted from Piece_material
				SELECT * FROM piece WHERE p_id = 2;	Record exists in Piece	Record exists in Piece
Piece_material	material_id	Material	The record in "Material" table can't be deleted because its material_id is referenced from table "Piece_material"	DELETE FROM Material WHERE material_id = 10;	Failed	Failed
			Delete Piece_material I record first but Material record should still exist	DELETE FROM piece_material WHERE material_id = 15;	Record deleted from Piece_material	Record deleted from Piece_material
				SELECT * FROM material WHERE material_id = 15;	Record exists in Material	Record exists in Material

Additional Tools

Google Drive - Used to share documents between teams and group members

Google Hangouts - Used to communicate between teams

Lucidchart - Used to by the E/R Diagram teams to create E/R diagrams

generatedata.com - Used by the Insert into Tables team to generate data to insert into the database

pgAdmin - Graphical UI used by various groups to access and modify data

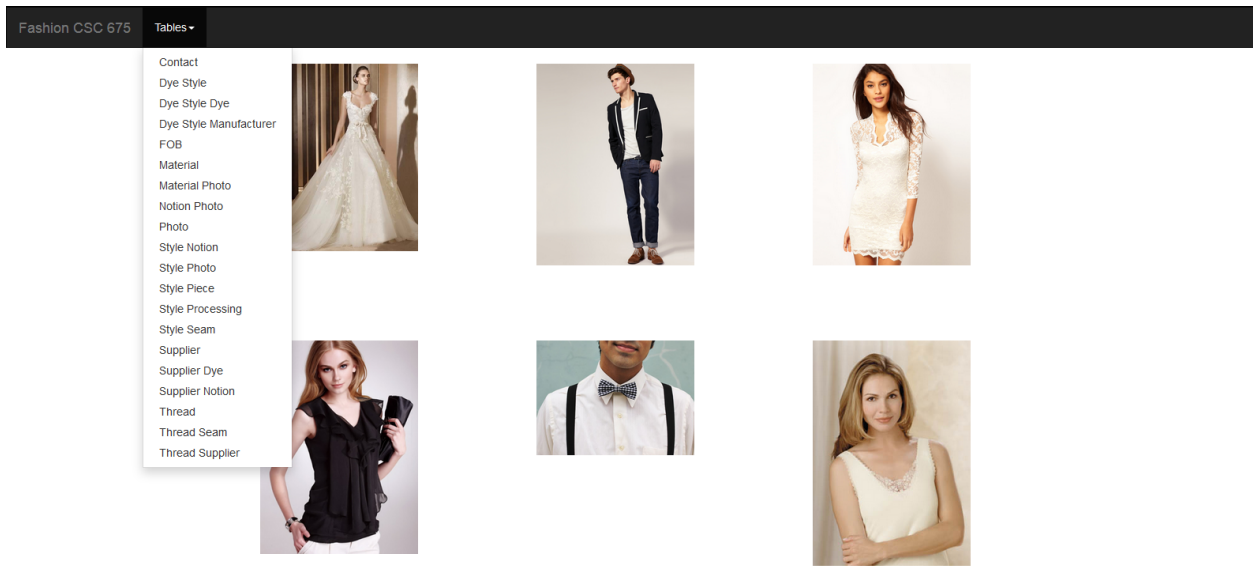
PHP - Used by the Web Interface team to connect the website to the database

HTML, CSS - Used by the Web Interface team to create the website

Web Interface

The web interface can be accessed from:

<https://tranquil-peak-5616.herokuapp.com/index.php>



- One may view the data stored in a table through the “Tables” drop list directly.
- One can also click on the photo of a model to find all the garments, styles, manufacturers, dyes, threads, materials, and patterns the model is wearing.

