NORTH ROAD FLORALS
Relational Database

Designed by Kathy Coomes
For
Database Management – Alan Labouseur
Spring 2017 – due 05/02/2017
# Table of Contents

Summary .................................................................................................................................................. 4

Entity Relationship Diagram .................................................................................................................. 5

Types: phone, prod, pos ......................................................................................................................... 6

## Tables

- People table ........................................................................................................................................... 7
- Vendors table ......................................................................................................................................... 8
- VendorContacts table .......................................................................................................................... 9
- Staff table ............................................................................................................................................. 10
- Customers table .................................................................................................................................... 11
- Addresses table .................................................................................................................................... 12
- Products table ....................................................................................................................................... 13
- ProductHistory table ............................................................................................................................ 14
- Arrangements table .............................................................................................................................. 15
- ArrangementItems ............................................................................................................................... 16
- ArrangementItemsList ........................................................................................................................... 17
- Inventory table ...................................................................................................................................... 18
- Orders table .......................................................................................................................................... 19

## Stored Procedures (also see Triggers)

- GetArrListofItems ................................................................................................................................. 20
- GetProductOrderList ............................................................................................................................ 21
- AddPeople .......................................................................................................................................... 22 - 24
- AddVendor (also see Reports) ............................................................................................................... 25
- AddProducts (also see Reports) ........................................................................................................... 26 - 27
- FindAlan (created by Daniel Verite — shared on Stack Overflow) ...................................................... 28 - 29

## Triggers

- ValidPeopleInput with Stored Procedure ValidatePeopleInput() ...................................................... 30
- ValidVendorInput with Stored Procedure ValidateVendorInput() .................................................... 31
Table of Contents

Views – (each with their own reports)

VendorInfo 32
Select all
Find vendor name by entering something LIKE the name

StaffInfo 33 - 34
Select all
Find an employee by entering something LIKE the first name
List all current Staff of North Road Florals
List any staff that has left North Road Florals

CustomerInfo 35
Select all
Find all Customers by entering something LIKE a city

ProductInfo 36 - 37
Select all
List current and history costs of Products bought

Reports

To list everything included in every table 38
Arrangement Item List 38
Vendors with no Vendor Contact 38
Vendors with no Products 39
Vendor from which we order the most 40
Find a Vendor city 40
Examples for reports from Stored Procedure AddVendor 41
Examples for reports from Stored Procedure AddProducts 41

Roles and Security 42 - 43

Implementation Notes 44

Known Problems and Future Enhancements 45
Summary

North Road Florals is a large floral business that has been steadily growing. The company has found that using spreadsheets, as they have, is no longer the most accurate, fastest and easiest way to handle all the information it needs to track about the people involved in or with the company – (Staff, Customers, and Vendor Contacts along with all their information). I was contacted to create a relational database for the company. They also wanted to track the history of products bought - including the current and past history of prices. Having spoken with the owner, arrangers, manager, and sales clerks many times, I have designed such a database in PostgreSQL.

While speaking with the staff of the company, I found that they also wanted the ability when an arrangement was ordered to check the list of items needed for that arrangement against inventory giving them a list of products needed to be ordered. I was able to create a stored procedure for them where all they need do is to enter the arrangement ID and they will receive a list of products to order.
Types

**phone Type** - the phone type is used in the People table

CREATE TYPE phone as ENUM ('work', 'home', 'cell');

---

**product Type** - used in the Product Table

CREATE TYPE prod as ENUM ('flower', 'container', 'greenery', 'balloon', 'other');

---

**position TYPE** - used in the Staff Table for hiredPosition and position

CREATE TYPE pos as ENUM ('owner', 'arranger', 'manager', 'salesclerk');
Tables

--- -------------------------------------------------------------

People
This table keeps track of the basic information that VendorContacts, Customers and Staff have in common.

--- -------------------------------------------------------------

CREATE TABLE People(
    peopleId text not null,
    firstName text not null,
    lastName text not null,
    phone text,
    phoneType text,
    email text,
    primary key(peopleId)
);

Functional Dependencies:

peopleId → firstName, lastName, phone, phoneType, email

Sample Data:

<table>
<thead>
<tr>
<th>peopleId</th>
<th>firstName</th>
<th>lastName</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>people001</td>
<td>John</td>
<td>Dolan</td>
<td>800-827-3665</td>
<td>work</td>
<td>floralsupply.com</td>
</tr>
<tr>
<td>people002</td>
<td>Michael</td>
<td>Growski</td>
<td>800-773-2554</td>
<td>work</td>
<td>directfloral.com</td>
</tr>
<tr>
<td>people003</td>
<td>Jessica</td>
<td>Murray</td>
<td>877-701-7673 - ext 5465</td>
<td>work</td>
<td>globalrose.com</td>
</tr>
<tr>
<td>people004</td>
<td>Jon</td>
<td>Sitzer</td>
<td>845-555-5555</td>
<td>home</td>
<td><a href="mailto:jon.sitzer34@yahoo.com">jon.sitzer34@yahoo.com</a></td>
</tr>
<tr>
<td>people005</td>
<td>Marisa</td>
<td>Sumner</td>
<td>845-666-6666</td>
<td>cell</td>
<td>[null]</td>
</tr>
<tr>
<td>people006</td>
<td>Mandy</td>
<td>Mishra</td>
<td>845-777-7777</td>
<td>home</td>
<td><a href="mailto:mmishra456@gmail.com">mmishra456@gmail.com</a></td>
</tr>
<tr>
<td>people011</td>
<td>Gary</td>
<td>Carney</td>
<td>845-222-2222</td>
<td>home</td>
<td><a href="mailto:gary.carney@net10.com">gary.carney@net10.com</a></td>
</tr>
<tr>
<td>people008</td>
<td>Janice</td>
<td>Jones</td>
<td>845-111-1111</td>
<td>home</td>
<td><a href="mailto:janicejones58@yahoo.com">janicejones58@yahoo.com</a></td>
</tr>
<tr>
<td>people009</td>
<td>Jane</td>
<td>Doe</td>
<td>845-555-2323</td>
<td>cell</td>
<td><a href="mailto:j3d2456@gmail.com">j3d2456@gmail.com</a></td>
</tr>
<tr>
<td>people010</td>
<td>Mary</td>
<td>Marist</td>
<td>914-555-1212</td>
<td>cell</td>
<td><a href="mailto:mmarist6486@marist.edu">mmarist6486@marist.edu</a></td>
</tr>
<tr>
<td>people007</td>
<td>Alan</td>
<td>Labouse...</td>
<td>845-440-1102</td>
<td>work</td>
<td><a href="mailto:alan@labousseur.com">alan@labousseur.com</a></td>
</tr>
<tr>
<td>people012</td>
<td>Candie</td>
<td>Kane</td>
<td>846-555-4545</td>
<td>home</td>
<td><a href="mailto:candie.kane22@outlook.com">candie.kane22@outlook.com</a></td>
</tr>
</tbody>
</table>
Tables

--- -----------------------------------------------------

Vendors
This table tracks Vendors from which the company buys its products. It creates a many to many relationship between VendorContacts and Products.

--- -----------------------------------------------------

CREATE TABLE Vendors(
  vendorId text not null unique,
  vendorName text not null,
  primary key(vendorId)
);

Functional Dependencies:

vendorId → vendorName

Sample Data:
Tables

-- ---------------------------------------------------------------------

VendorContacts
This table is an extension of the People table for Vendor Contact Information.

-- ---------------------------------------------------------------------

CREATE TABLE VendorContacts (  
    peopleId  text  not null  references People(peopleId),  
    vendorId  text  not null  references Vendors(vendorId),  
    primary key(peopleId, vendorId)  
);

Functional Dependencies:

peopleId $\rightarrow$ vendorId

Sample Data:

<table>
<thead>
<tr>
<th>peopleId</th>
<th>vendorId</th>
</tr>
</thead>
<tbody>
<tr>
<td>people001</td>
<td>vendor001</td>
</tr>
<tr>
<td>people002</td>
<td>vendor002</td>
</tr>
<tr>
<td>people003</td>
<td>vendor003</td>
</tr>
</tbody>
</table>
Tables

--  Adam Developed

Staff
This table is an extension of the People table for Staff Information.

CREATE TABLE Staff (  
    peopleId text not null references People(peopleId),  
    dateHired text not null,  
    hiredPosition pos not null,  
    dateLeft text DEFAULT NULL,  
    position pos not null,  
    cellphone text,  
    primary key(peopleId)  
);

Functional Dependencies:

peopleId → dateHired, hiredPosition, dateLeft, position, cellphone

Sample Data:

<table>
<thead>
<tr>
<th>peopleId</th>
<th>dateHired</th>
<th>hiredPosition</th>
<th>dateLeft</th>
<th>position</th>
<th>cellphone</th>
</tr>
</thead>
<tbody>
<tr>
<td>people004</td>
<td>2007-05-12</td>
<td>sales clerk</td>
<td>[null]</td>
<td>manager</td>
<td>914-555-5555</td>
</tr>
<tr>
<td>people005</td>
<td>2015-02-23</td>
<td>sales clerk</td>
<td>[null]</td>
<td>sales clerk</td>
<td>914-666-6666</td>
</tr>
<tr>
<td>people006</td>
<td>2016-01-01</td>
<td>sales clerk</td>
<td>[null]</td>
<td>arranger</td>
<td>914-777-7777</td>
</tr>
<tr>
<td>people011</td>
<td>2007-05-12</td>
<td>owner</td>
<td>[null]</td>
<td>owner-arranger</td>
<td>914-222-2222</td>
</tr>
<tr>
<td>people008</td>
<td>2007-05-12</td>
<td>sales clerk</td>
<td>2007-07-30</td>
<td>sales clerk</td>
<td>914-111-1111</td>
</tr>
</tbody>
</table>
Tables

------------------

Customers
This table is an extension of the People Table for Customer Information.

CREATE TABLE Customers (  
    peopleId   text   not null   references People(peopleId),  
    primary key(peopleId)  
);  

Functional Dependencies:  

peopleId  

Sample Data:

<table>
<thead>
<tr>
<th>peopleId</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>people009</td>
<td></td>
</tr>
<tr>
<td>people010</td>
<td></td>
</tr>
<tr>
<td>people007</td>
<td></td>
</tr>
<tr>
<td>people012</td>
<td></td>
</tr>
</tbody>
</table>
Tables

Addresses
This table keeps track of the addresses of all People.

CREATE TABLE Addresses (  
    peopleId text not null references People(peopleId),  
    addressId text,  
    street1 text,  
    street2 text,  
    city text,  
    state text,  
    zip text not null references Zips(zip),  
    primary key(peopleId, addressId)  );

Functional Dependencies:
peopleId, addressId → street1, street2, city, state, zip

Sample Data:
**Tables**

-- ------------------------------------------------------------------------

**Products**
This table tracks current product information.

-- ------------------------------------------------------------------------

CREATE TABLE Products (  
  vdrProdld text not null unique,  
  vendorId text not null references Vendors(vendorId),  
  Type prod,  
  Name text,  
  Size text,  
  Color text,  
  Qty int CHECK (productQty >= 0),  
  -- # of items in currentCostUSD, might be sold singly, or by the dozen  
  currentCostUSD numeric(6,2) CHECK (currentCostUSD > 0),  
  prodDesc text,  
  primary key(vdrProdld, vendorId)  
);

**Functional Dependencies:**

vdrProdld, vendorID → Type, Name, Size, Color, Qty, currentCostUSD, prodDesc

**Sample Data:**

<table>
<thead>
<tr>
<th>vdrProdld</th>
<th>vendorId</th>
<th>type</th>
<th>name</th>
<th>size</th>
<th>color</th>
<th>qty</th>
<th>currentCostUSD</th>
<th>prodDesc</th>
</tr>
</thead>
<tbody>
<tr>
<td>G10026E2</td>
<td>vendor003</td>
<td>flower</td>
<td>rose</td>
<td>long stem</td>
<td>lavender</td>
<td>12</td>
<td>7.00</td>
<td>[null]</td>
</tr>
<tr>
<td>C56736</td>
<td>vendor001</td>
<td>container</td>
<td>country metal pail</td>
<td>6 inches tall</td>
<td>off white</td>
<td>5</td>
<td>12.00</td>
<td>[null]</td>
</tr>
<tr>
<td>N674DDA</td>
<td>vendor002</td>
<td>flower</td>
<td>alstroemeria</td>
<td>[null]</td>
<td>white</td>
<td>6</td>
<td>4.50</td>
<td>[null]</td>
</tr>
<tr>
<td>N673GST</td>
<td>vendor002</td>
<td>flower</td>
<td>statice</td>
<td>[null]</td>
<td>white</td>
<td>6</td>
<td>3.00</td>
<td>[null]</td>
</tr>
<tr>
<td>N675GHF</td>
<td>vendor002</td>
<td>flower</td>
<td>daisy</td>
<td>[null]</td>
<td>yellow</td>
<td>12</td>
<td>12.00</td>
<td>[null]</td>
</tr>
<tr>
<td>S97854</td>
<td>vendor001</td>
<td>other</td>
<td>floral foam</td>
<td>9in long x 4.5in wide x 3in high</td>
<td>green</td>
<td>48</td>
<td>25.00</td>
<td>dry brick</td>
</tr>
<tr>
<td>S97657</td>
<td>vendor001</td>
<td>other</td>
<td>floral tape</td>
<td>110yds long x 1in wide</td>
<td>green</td>
<td>6</td>
<td>6.75</td>
<td>rolls</td>
</tr>
</tbody>
</table>
| C56765     | vendor001| container | its a boy wagon | 9in long x 4in wide x 4in high | red | 1 | 9.99 | good for plant or an
### ProductHistory
This table tracks the history of products: ID, cost when bought, date bought.

```sql
CREATE TABLE ProductHistory (
  vдрProdId text references Products(vдрProdId),
  historyCostUSD numeric(6,2) not null CHECK (historyCostUSD > 0),
  dateBought text not null,
  primary key(vдрProdId, historyCostUSD, dateBought)
);
```

**Functional Dependencies:**

```
vдрProdId → historyCostUSD, dateBought
```

**Sample Data:**

<table>
<thead>
<tr>
<th>vдрProdId</th>
<th>historyCostUSD</th>
<th>dateBought</th>
</tr>
</thead>
<tbody>
<tr>
<td>G10026BG</td>
<td>10.00</td>
<td>2016-02-02</td>
</tr>
<tr>
<td>C78457</td>
<td>12.00</td>
<td>2016-02-02</td>
</tr>
<tr>
<td>N67454C3</td>
<td>5.00</td>
<td>2016-02-02</td>
</tr>
<tr>
<td>C97485</td>
<td>6.00</td>
<td>2016-02-02</td>
</tr>
<tr>
<td>G87465</td>
<td>13.50</td>
<td>2016-02-02</td>
</tr>
<tr>
<td>S97657</td>
<td>5.75</td>
<td>2016-02-02</td>
</tr>
<tr>
<td>N674DDA</td>
<td>6.00</td>
<td>2016-02-02</td>
</tr>
<tr>
<td>N67454G5</td>
<td>8.25</td>
<td>2016-02-15</td>
</tr>
</tbody>
</table>
Tables

Arrangements
This table tracks arrangements: ID, name, cost, and category. Category can be any holiday, birthday, birth of a child, thank you, etc.

CREATE TABLE Arrangements(
    arrId  text not null unique,
    arrName text not null,
    arrCostUSD numeric (6,2) not null CHECK (arrCostUSD > 0),
    category text not null,
    primary key(arrId)
);

Functional Dependencies:

arrId \rightarrow arrName, arrCostUSD, category

Sample Data:

<table>
<thead>
<tr>
<th>arrId</th>
<th>arrName</th>
<th>arrCostUSD</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr001</td>
<td>Blooming Pail</td>
<td>50.00</td>
<td>Spring</td>
</tr>
<tr>
<td>arr002</td>
<td>Wow Wagon - boy</td>
<td>43.00</td>
<td>baby boy</td>
</tr>
<tr>
<td>arr003</td>
<td>Wow Wagon - girl</td>
<td>43.00</td>
<td>baby girl</td>
</tr>
<tr>
<td>arr004</td>
<td>Fly Away Labouseur</td>
<td>53.00</td>
<td>birthday</td>
</tr>
<tr>
<td>arr005</td>
<td>lovely Ladybug Bouquet</td>
<td>40.00</td>
<td>general</td>
</tr>
</tbody>
</table>
Tables

--- -----------------------------------------------

ArrangementItems
This table gives a list of items that can be used in an arrangement.

--- -----------------------------------------------

CREATE TABLE ArrangementItems(
    itemId        text  not null  unique,
    itemName      text  not null,
    primary key(itemId)
);

Functional Dependencies:

itemId \rightarrow itemName

Sample Data:

<table>
<thead>
<tr>
<th>itemId</th>
<th>itemname</th>
</tr>
</thead>
<tbody>
<tr>
<td>item001</td>
<td>Country Metal Pail by Alan</td>
</tr>
<tr>
<td>Item002</td>
<td>Lavender Roses</td>
</tr>
<tr>
<td>item003</td>
<td>Alstroemeria</td>
</tr>
<tr>
<td>item004</td>
<td>Static</td>
</tr>
<tr>
<td>item005</td>
<td>Yellow Daisies</td>
</tr>
<tr>
<td>item006</td>
<td>Floral Foam</td>
</tr>
<tr>
<td>item007</td>
<td>Floral tape</td>
</tr>
<tr>
<td>item008</td>
<td>Its a Rav Red Wanon</td>
</tr>
</tbody>
</table>
Tables

-- ---------------------------------------------------------------------

ArrangementItemsList
This table brings together the arrangement and arrangementItems tables
giving the number of each item to be used in each arrangement.
-- ---------------------------------------------------------------------

CREATE TABLE ArrangementItemsList(
    arrId   text not null references Arrangements(arrId),
    itemId  text not null references ArrangementItems(itemId),
    itemQty int not null CHECK (itemQty >= 0),
    primary key(arrId, itemId, itemQty)
);

Functional Dependencies:

arrId, itemId → itemQty

Sample Data:

<table>
<thead>
<tr>
<th>arrId</th>
<th>itemId</th>
<th>itemQty</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr001</td>
<td>item001</td>
<td>1</td>
</tr>
<tr>
<td>arr001</td>
<td>item002</td>
<td>8</td>
</tr>
<tr>
<td>arr001</td>
<td>item003</td>
<td>5</td>
</tr>
<tr>
<td>arr001</td>
<td>item004</td>
<td>6</td>
</tr>
<tr>
<td>arr001</td>
<td>item005</td>
<td>5</td>
</tr>
<tr>
<td>arr001</td>
<td>item006</td>
<td>1</td>
</tr>
<tr>
<td>arr001</td>
<td>item007</td>
<td>1</td>
</tr>
<tr>
<td>arr001</td>
<td>item015</td>
<td>4</td>
</tr>
<tr>
<td>arr002</td>
<td>item005</td>
<td>3</td>
</tr>
<tr>
<td>arr002</td>
<td>item006</td>
<td>1</td>
</tr>
<tr>
<td>arr002</td>
<td>item007</td>
<td>1</td>
</tr>
</tbody>
</table>
Tables

-- -----------------------------------------------------------------------------

Inventory
This table is a list of how many of each item the company has on hand. It creates a many to many relationship between Products and ArrangementItems.

-- -----------------------------------------------------------------------------

CREATE TABLE Inventory (  
vdrProdId text references Products(vdrProdId),  
itemId text references ArrangementItems(itemId),  
invQty int CHECK (invQty >= 0),  
primary key(vdrProdId, itemId)
);

Functional Dependencies:

vdrProdId, itemId → invQty

Sample Data:

<table>
<thead>
<tr>
<th>vdrprodid</th>
<th>itemid</th>
<th>invqty</th>
</tr>
</thead>
<tbody>
<tr>
<td>G10026B2</td>
<td>item002</td>
<td>10</td>
</tr>
<tr>
<td>C56736</td>
<td>item001</td>
<td>4</td>
</tr>
<tr>
<td>N674DDA</td>
<td>item003</td>
<td>0</td>
</tr>
<tr>
<td>N673GST</td>
<td>item004</td>
<td>0</td>
</tr>
<tr>
<td>N675GH</td>
<td>item005</td>
<td>10</td>
</tr>
<tr>
<td>S97854</td>
<td>item006</td>
<td>45</td>
</tr>
<tr>
<td>S97657</td>
<td>item007</td>
<td>6</td>
</tr>
<tr>
<td>C56765</td>
<td>item008</td>
<td>1</td>
</tr>
</tbody>
</table>
Tables

Orders
This table is a list of orders of arrangements from the Customer.
It creates a many to many relationship from Customers to Arrangements.

CREATE TABLE Orders (  
  peopleId text references People(peopleId),  
  arrId text references Arrangements(arrId),  
  orderDate date CHECK (orderDate <= deliveryDate),  
  deliveryDate date CHECK (deliveryDate >= orderDate),  
  deliveryName text,  
  deliveryStreet1 text,  
  deliveryStreet2 text,  
  deliveryCity text,  
  customerPhone text,  
  deliveryPhone text,  
  cardScript text not null,  
  specInstruct text not null DEFAULT 'none',  
  primary key(peopleId, arrId, orderDate, deliveryDate, deliveryName)  
);  

Functional Dependencies:
peopleId, arrId, orderDate, deliveryDate, deliveryName → deliveryStreet1, deliveryStreet2, deliveryCity, customerPhone, deliveryPhone, cardScript, specInstruct

Sample Data:

<table>
<thead>
<tr>
<th>text</th>
<th>text</th>
<th>date</th>
<th>date</th>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>people009</td>
<td>arr004</td>
<td>2017-04-21</td>
<td>2017-04-22</td>
<td>Jodi Walker</td>
<td>Park Condos</td>
<td>78 Eastwood Drive</td>
<td>Hyde Park</td>
<td>845-632-5896</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>text</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>914-258-3571</td>
<td>Jodi - Have a Wonderful EBirthday. We love you, Grammy and Popoy none</td>
</tr>
</tbody>
</table>
Stored Procedures

GetArrListofItems - Stored Procedure - enter arrId
To obtain a list of items and quantities of each for that arrangement

CREATE OR REPLACE FUNCTION GetArrListofItems(TEXT, REFCURSOR)
RETURNS refcursor AS
$$
DECLARE
    arrIdEntered TEXT := $1;
    resultset REFCURSOR := $2;
BEGIN
    OPEN resultset FOR
    SELECT a.arrId, p.itemId, p.itemName, apl.itemQty
    FROM Arrangements a INNER JOIN ArrangementItemsList apl ON a.arrId = apl.arrId
    INNER JOIN ArrangementItems p ON apl.itemId = p.itemId
    WHERE a.arrId = arrIdEntered
    ORDER BY p.itemId ASC;
    RETURN resultset;
END;
$$
LANGUAGE plpgsql;

SELECT GetArrListofItems('arr002', 'results');
FETCH all FROM results;

Sample Data:

<table>
<thead>
<tr>
<th>arrId</th>
<th>itemId</th>
<th>itemName</th>
<th>itemQty</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr002</td>
<td>item005</td>
<td>Yellow Daisies</td>
<td>3</td>
</tr>
<tr>
<td>arr002</td>
<td>item006</td>
<td>Floral Foam</td>
<td>1</td>
</tr>
<tr>
<td>arr002</td>
<td>item007</td>
<td>Floral tape</td>
<td>1</td>
</tr>
<tr>
<td>arr002</td>
<td>item016</td>
<td>Its a Girl Red Wag...</td>
<td>1</td>
</tr>
<tr>
<td>arr002</td>
<td>item009</td>
<td>Yellow Spray Roses</td>
<td>2</td>
</tr>
<tr>
<td>arr002</td>
<td>item010</td>
<td>Red Mini Gerbera</td>
<td>3</td>
</tr>
<tr>
<td>arr002</td>
<td>item011</td>
<td>Asters</td>
<td>2</td>
</tr>
<tr>
<td>arr002</td>
<td>item012</td>
<td>White Daisies</td>
<td>3</td>
</tr>
<tr>
<td>arr002</td>
<td>item013</td>
<td>White Button Mums</td>
<td>2</td>
</tr>
<tr>
<td>arr002</td>
<td>item014</td>
<td>Soldago</td>
<td>4</td>
</tr>
<tr>
<td>arr002</td>
<td>item015</td>
<td>Greenery</td>
<td>4</td>
</tr>
</tbody>
</table>
Stored Procedures

GetProductOrderList
A customer has ordered an arrangement - Enter the arrId and this stored procedure compares the items needed for the arrangement against what is in Inventory and gives a list of items that need to be ordered.

CREATE OR REPLACE FUNCTION GetProductOrderList(TEXT, REFCURSOR) RETURNS refcursor AS $$
DECLARE
  arrIdEntered TEXT := $1;
  resultset      REFCURSOR := $2;
BEGIN
  OPEN resultset FOR
    SELECT i.itemId, i.vdrProdID, v.vendorName, pe.firstName, pe.lastName, pe.phone
    FROM Inventory i
    INNER JOIN ArrangementItems ai ON i.itemId = ai.itemId
    INNER JOIN ArrangementItemsList ail ON ail.itemId = ai.itemId
    INNER JOIN Arrangements a ON ail.arrId = a.arrId
    INNER JOIN Products pr ON i.vdrProdID = pr.vdrProdID
    INNER JOIN Vendors v ON pr.vendorID = v.vendorId
    INNER JOIN VendorContacts vc ON v.vendorId = vc.vendorId
    INNER JOIN People pe ON vc.peopleId = pe.peopleId
    WHERE a.arrId = arrIdEntered AND i.invQty <= ail.itemQty
    ORDER BY p.itemId ASC;
  RETURN resultset;
END;
$$
LANGUAGE plpgsql;

SELECT GetProductOrderList('arr002', 'results2');
FETCH all FROM results2;

Sample Data:

<table>
<thead>
<tr>
<th>ItemId</th>
<th>itemQty</th>
<th>invQty</th>
<th>venderproductid</th>
<th>vendorname</th>
<th>firstname</th>
<th>lastname</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item005</td>
<td>3</td>
<td>0</td>
<td>N675GTF</td>
<td>DirectFloral.com</td>
<td>Michael</td>
<td>Growski</td>
<td>800-773-2554</td>
</tr>
<tr>
<td>Item010</td>
<td>3</td>
<td>0</td>
<td>N67454C3</td>
<td>GlobalRose.com</td>
<td>Jessica</td>
<td>Murray</td>
<td>877-701-7673 - ext 5465</td>
</tr>
<tr>
<td>Item011</td>
<td>2</td>
<td>0</td>
<td>N67454G5</td>
<td>GlobalRose.com</td>
<td>Jessica</td>
<td>Murray</td>
<td>877-701-7673 - ext 5465</td>
</tr>
<tr>
<td>Item013</td>
<td>2</td>
<td>0</td>
<td>N697BHG</td>
<td>DirectFloral.com</td>
<td>Michael</td>
<td>Growski</td>
<td>800-773-2554</td>
</tr>
<tr>
<td>Item014</td>
<td>4</td>
<td>0</td>
<td>N9865GKB</td>
<td>DirectFloral.com</td>
<td>Michael</td>
<td>Growski</td>
<td>800-773-2554</td>
</tr>
<tr>
<td>Item016</td>
<td>1</td>
<td>1</td>
<td>C56766</td>
<td>Floral Supply Wholesale</td>
<td>John</td>
<td>Dolan</td>
<td>800-827-3665</td>
</tr>
</tbody>
</table>
Stored Procedures

AddPeople -
select addPeople(enter info with NULL for data not needed);
procedure then adds people and address information, and uses IF statements to decide whether to enter Staff or Customer or VendorContact information

CREATE OR REPLACE FUNCTION AddPeople
    (peopleType text, peopleId text, firstName text, lastName text, phone text, phoneType phone, email text, addressId text, street1 text, street2 text, city text, state text, zip text, dateHired date, hiredPosition pos, dateLeft date, currentPosition pos, cellPhone text, vendorId text)
RETURNS void AS
$$
BEGIN
    INSERT INTO People
        VALUES (peopleId, firstName, lastName, phone, phoneType, email);
    INSERT INTO addresses
        VALUES(peopleId, addressId, street1, street2, city, state, zip);

    IF peopleType = 'S' THEN
        INSERT INTO Staff
            VALUES(peopleId, dateHired, hiredPosition, dateLeft, currentPosition, cellPhone);
    END IF;

    IF peopleType = 'C' THEN
        INSERT INTO Customers
            VALUES(peopleId);
    END IF;

    IF peopleType = 'V' THEN
        INSERT INTO VendorContacts
            VALUES(peopleId, vendorId);
    END IF;
END
$$
LANGUAGE plpgsql;

-- TEST DATA for AddPeople - Staff:
Select AddPeople('S','people013', 'Kathy', 'Coomes', '555-555-5555', 'cell',
    'kathy.coomes@marist.edu', NULL, 'address013', '19 Church Street', 'Red Hook', 'New York', '12571', '04/30/2017', 'arranger', NULL, 'arranger', '555-555-5555', NULL);
select * from StaffInfo;
Stored Procedures

-- TEST DATA for AddPeople - Customers:

Select AddPeople('C', 'people014', 'Jennie', 'Masters', '555-555-5555', 'home', 'jennie3456@gmail.com', NULL, 'address014', '100 Main Street', 'Rhinecliff', 'New York', '12573', NULL, NULL, NULL, NULL, NULL);

SELECT * FROM CustomerInfo;
Stored Procedures

-- TEST DATA for AddPeople - VendorContacts:
Select AddPeople('V', 'people025', 'George', 'Baker', '555-555-5555', 'work',
    'george@directfloral.com', NULL, 'address025', '3657 Floral Drive', 'Pleasant Valley','New York',
    '12569', NULL, NULL, NULL, NULL, 'vendor002');
SELECT * FROM VendorInfo;

<table>
<thead>
<tr>
<th>vendorid</th>
<th>vendorname</th>
<th>peopleid</th>
<th>firstname</th>
<th>lastname</th>
<th>phone</th>
<th>phonenumber</th>
<th>type</th>
<th>phoneext</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor001</td>
<td>Floral Supply Wholesaler</td>
<td>people001</td>
<td>John</td>
<td>Dolan</td>
<td>800-827-3665</td>
<td>work</td>
<td>fl</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vendor002</td>
<td>DirectFloral.com</td>
<td>people002</td>
<td>Michael</td>
<td>Growski</td>
<td>800-773-2554</td>
<td>work</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vendor002</td>
<td>DirectFloral.com</td>
<td>people025</td>
<td>George</td>
<td>Baker</td>
<td>555-555-5555</td>
<td>work</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vendor003</td>
<td>GlobalRose.com</td>
<td>people003</td>
<td>Jessica</td>
<td>Murray</td>
<td>877-701-7673 - ext 5465</td>
<td>work</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>emailtext</th>
<th>street1text</th>
<th>street2text</th>
<th>citytext</th>
<th>statetext</th>
<th>zip</th>
<th>texttext</th>
<th>phonetext</th>
</tr>
</thead>
<tbody>
<tr>
<td>floralsupply.com</td>
<td>[null]</td>
<td>15 Applewood Drive</td>
<td>Fruit Heights</td>
<td>Utah</td>
<td>84037</td>
<td></td>
<td></td>
</tr>
<tr>
<td>directfloral.com</td>
<td>[null]</td>
<td>750 Killian Road</td>
<td>Akron</td>
<td>Ohio</td>
<td>44319</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="mailto:george@directfloral.com">george@directfloral.com</a></td>
<td>[null]</td>
<td>3657 Floral Drive</td>
<td>Pleasant Valley</td>
<td>New York</td>
<td>12569</td>
<td></td>
<td></td>
</tr>
<tr>
<td>globalrose.com</td>
<td>[null]</td>
<td>7225 NW 25th St</td>
<td>Miami</td>
<td>Florida</td>
<td>33122</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Stored Procedures**

---

**AddVendor** -
enter select AddVendor(enter spots with NULL for data not needed)

---

CREATE OR REPLACE FUNCTION AddVendor
  (vendorId text, vendorName text)
RETURNS void AS
$$
BEGIN
  INSERT INTO Vendors
  VALUES (vendorId, vendorName);
END
$$
LANGUAGE plpgsql;

-- Test Data:
Select AddVendor('vendor005', 'Florist ExtraOrdinary');
Select * FROM Vendors;

Data Output:  
<table>
<thead>
<tr>
<th>vendorid</th>
<th>vendorname</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor001</td>
<td>Floral Supply Wholesale</td>
</tr>
<tr>
<td>vendor002</td>
<td>DirectFloral.com</td>
</tr>
<tr>
<td>vendor003</td>
<td>GlobalRose.com</td>
</tr>
<tr>
<td>vendor004</td>
<td>Flowers by Alan</td>
</tr>
<tr>
<td>vendor005</td>
<td>Florist ExtraOrdinary</td>
</tr>
</tbody>
</table>
Stored Procedures

---

AddProducts -
adds information to Products, ProductHistory and Inventory once a Product order has been received -- updates or inserts new

---

CREATE OR REPLACE FUNCTION AddProducts
  (productNew text, newVdrProdId text, vendorId text, newType text,
   newName text, newSize text, newColor text, newQty int,
   newCurrentCostUSD numeric, newProdDesc text, historyCostUSD numeric,
   dateBought date, newItemId text, NewItemName text, newInvQty int)
RETURNS void AS $$
BEGIN
  IF productNew = 'Y' THEN
    INSERT INTO Products
      VALUES (newVdrProdId, vendorId, newType,
              newName, newSize, newColor, newQty, newCurrentCostUSD, newProdDesc);
    INSERT INTO ArrangementItems
      VALUES (newItemId, newItemName);
    INSERT INTO Inventory
      VALUES(newVdrProdId, newItemId, newInvQty);
    INSERT INTO ProductHistory
      VALUES(newVdrProdId, historyCostUSD, dateBought);
  END IF;

  IF productNew = 'N' THEN
    UPDATE Products SET
      productType = newProductType,
      productName = newProductName,
      productSize = newProductSize,
      productColor = newProductColor,
      productQty = newProductQty,
      currentCostUSD = newCurrentCostUSD,
      productDesc = newProductDesc
    WHERE vendorProductId = newVendorProductId;
    INSERT INTO ProductHistory
      VALUES(newVendorProductId, historyCostUSD, dateBought);
    UPDATE Inventory SET invQty = invQty + newInvQty
      WHERE vendorproductId = newVendorProductId and itemId = newItemId;
  END IF;
END
$$ LANGUAGE plpgsql;
Stored Procedures

-- TEST DATA – Add new product:
SELECT AddProducts('Y', 'G10026BH', 'vendor003', 'flower', 'rose', 'long stem',
'red', 12, 15, NULL, 15, '04/30/2017', 'item034', 'red roses', 12);

SELECT * FROM Products;

SELECT * FROM ProductHistory;
SELECT * FROM Inventory;
SELECT * FROM ArrangementItems;

-- TEST DATA – Update a product:
SELECT AddProducts('N', 'G10026B2', 'vendor003', 'flower', 'rose', 'long stem',
'lavender', 12, 15, NULL, 15, '04/30/2017', 'item002', 'lavender roses', 12);

SELECT * FROM Products;
SELECT * FROM ProductHistory;
SELECT * FROM Inventory;
SELECT * FROM ArrangementItems;
Stored Procedures

------------------------------------------

Find Alan Report - Finds Alan or Labouseur anywhere in the database

Created by Daniel Verite and shared on Stack Overflow on 4/12/2014 under the title of 'How to search a specific value in all tables (PostgreSQL)?' works in version 9.1 or newer
He stated the following:
"Here is a plpgsql function that locates records where any column contains a specific value. It takes as arguments the value to search in text format, an array of table names to search into (defaults to all tables) and an array of schema names (public by default. It returns a table structure with schema, name of table, name of column and pseudo-column ctid (non-durable physical location of the row in the table.)"

CREATE OR REPLACE FUNCTION search_columns(
    needle text,
    haystack_tables name[] default '{}',
    haystack_schema name[] default '{public}'
)
RETURNS table(schemaname text, tablename text, columnname text, rowctid text)
AS $$
begin
    FOR schemaname, tablename, columnname IN
        SELECT c.table_schema, c.table_name, c.column_name
        FROM information_schema.columns c
        JOIN information_schema.tables t ON
            (t.table_name=c.table_name AND t.table_schema=c.table_schema)
        WHERE (c.table_name=ANY(haystack_tables) OR
            haystack_tables='{}')
        AND c.table_schema=ANY(haystack_schema)
        AND t.table_type='BASE TABLE'
    LOOP
        EXECUTE format('SELECT ctid FROM %I.%I WHERE cast(%I as text)=%L',
            schemaname, tablename, columnname, needle)
        INTO rowctid;
        IF rowctid is not null THEN
            RETURN NEXT;
        END IF;
    END LOOP;
END;
$$ language plpgsql;
Stored Procedures

SELECT * FROM search_columns('Alan');

```
<table>
<thead>
<tr>
<th>schemaname_text</th>
<th>tablename_text</th>
<th>columnname_text</th>
<th>rowctid_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>people</td>
<td>firstname</td>
<td>(0,11)</td>
</tr>
<tr>
<td>public</td>
<td>vendors</td>
<td>vendornoame</td>
<td>(0,4)</td>
</tr>
<tr>
<td>public</td>
<td>arrangementitems</td>
<td>itemname</td>
<td>(0,1)</td>
</tr>
</tbody>
</table>
```

SELECT * FROM search_columns('Labouseur');

```
<table>
<thead>
<tr>
<th>schemaname_text</th>
<th>tablename_text</th>
<th>columnname_text</th>
<th>rowctid_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>people</td>
<td>lastname</td>
<td>(0,11)</td>
</tr>
<tr>
<td>public</td>
<td>arrangements</td>
<td>arname</td>
<td>(0,4)</td>
</tr>
</tbody>
</table>
```
Triggers

--- -----------------------------------------------------

Trigger – **validPeopleInput**
Stored Procedure - **ValidatePeopleInput()**
Triggers on a new entry or update in the People table

--- -----------------------------------------------------

CREATE OR REPLACE FUNCTION ValidatePeopleInput()
RETURNS TRIGGER AS
$$
BEGIN
  IF NEW.firstName IS NULL THEN
    RAISE EXCEPTION 'firstName may not be NULL';
  END IF;
  IF NEW.lastName IS NULL THEN
    RAISE EXCEPTION 'lastName may not be NULL';
  END IF;
  RETURN NEW;
END
$$
LANGUAGE plpgsql;

--- -----------------------------------------------------

CREATE TRIGGER validPeopleInput
BEFORE INSERT OR UPDATE ON People
FOR EACH ROW
EXECUTE PROCEDURE ValidatePeopleInput();

--- -----------------------------------------------------

**Test Data**

INSERT INTO People (peopleId, firstName, lastName)
VALUES('people013', NULL, NULL);
INSERT INTO People (peopleId, firstName, lastName)
VALUES('people013' Joyce, NULL);

**Results:**

ERROR:  firstName may not be NULL
CONTEXT:  PL/pgSQL function validatepeopleinput() line 4 at RAISE

ERROR:  lastName may not be NULL
CONTEXT:  PL/pgSQL function validatepeopleinput() line 7 at RAISE
Triggers

--- --------------------------------------------------

Trigger - ValidVendorInput
Stored Procedure - ValidateVendorInput
Triggers on insert or update of the Vendor table
--- --------------------------------------------------

CREATE OR REPLACE FUNCTION ValidateVendorInput()
RETURNS TRIGGER AS
$$
BEGIN
  IF NEW.vendorId IS NULL THEN
    RAISE EXCEPTION 'vendorId may not be NULL';
  END IF;
  IF NEW.vendorName IS NULL THEN
    RAISE EXCEPTION 'vendorName may not be NULL';
  END IF;
  RETURN NEW;
END
$$
LANGUAGE plpgsql;
--- --------------------------------------------------

CREATE TRIGGER ValidVendorInput
BEFORE INSERT OR UPDATE ON Vendors
FOR EACH ROW
EXECUTE PROCEDURE ValidateVendorInput();
--- --------------------------------------------------

Test Data
INSERT INTO Vendors (vendorId, vendorName)
  VALUES(NULL, 'LabouseursPlace');
INSERT INTO Vendors (vendorId, vendorName)
  VALUES('vendor234', NULL);

Results:
ERROR: vendorId may not be NULL
CONTEXT: PL/pgSQL function validatevendorinput() line 4 at RAISE
ERROR: vendorName may not be NULL
CONTEXT: PL/pgSQL function validatevendorinput() line 7 at RAISE
Views

---

VendorInfo – A view with 2 reports to obtain Vendor Information from People, Addresses, Zips, Vendors, VendorContacts

---

CREATE VIEW VendorInfo
AS
SELECT v.vendorId, v.vendorName, vc.peopleId, p.firstName, p.lastName, p.phone, p.phoneType, p.email, a.addressId, a.street1, a.street2, a.zip, z.city, z.state
FROM Vendors v,
    VendorContacts vc,
    People p,
    Addresses a,
    Zips z
WHERE v.vendorId = vc.vendorId and
    vc.peopleId = p.peopleId and
    p.peopleId = a.peopleId and
    a.zip = z.zip
ORDER BY v.vendorId ASC;

---

query 1 - all Vendor Information

SELECT * FROM VendorInfo;

Sample Data:

<table>
<thead>
<tr>
<th>vendorId</th>
<th>vendorName</th>
<th>peopleId</th>
<th>firstName</th>
<th>lastName</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
<th>addressId</th>
<th>street1</th>
<th>street2</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor001</td>
<td>Floral Support</td>
<td>people01</td>
<td>John</td>
<td>Dolan</td>
<td>800-827</td>
<td>work</td>
<td>floralsp...</td>
<td>address01</td>
<td>15 App...</td>
<td>Fruit Htl</td>
<td>Utah</td>
<td>84037</td>
<td></td>
</tr>
<tr>
<td>vendor002</td>
<td>DirectFloral</td>
<td>people02</td>
<td>Michael</td>
<td>Growski</td>
<td>800-773</td>
<td>work</td>
<td>directfl...</td>
<td>address02</td>
<td>760 Kill...</td>
<td>Akron</td>
<td>Ohio</td>
<td>44319</td>
<td></td>
</tr>
<tr>
<td>vendor003</td>
<td>GlobalRose</td>
<td>people03</td>
<td>Jessica</td>
<td>Murray</td>
<td>877-701</td>
<td>work</td>
<td>globalrose...</td>
<td>address03</td>
<td>7225 NW 25...</td>
<td>Miami</td>
<td>Florida</td>
<td>33122</td>
<td></td>
</tr>
</tbody>
</table>

---

query 2 - search for an unknown name

SELECT vendorId, vendorName, firstName, LastName, phone, email
FROM VendorInfo
WHERE vendorName Like '%Rose%';

Sample Data:

<table>
<thead>
<tr>
<th>vendorId</th>
<th>vendorName</th>
<th>firstName</th>
<th>lastName</th>
<th>phone</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor003</td>
<td>GlobalRose.com</td>
<td>Jessica</td>
<td>Murray</td>
<td>877-701</td>
<td>globalrose.com</td>
</tr>
</tbody>
</table>
Views

-- -------------------------------------------------

StaffInfo – A view with 4 reports to obtain Vendor Information from People, Addresses, Zips, Staff
-- -------------------------------------------------

CREATE VIEW StaffInfo
AS
SELECT s.peopleId, s.dateHired, s.hiredPosition, s.dateLeft, s.position, s.cellPhone,
    p.firstName, p.lastName, p.phone, p.phoneType, p.email,
    a.address, a.street1, a.street2,
    a.zip, z.city, z.state
FROM Staff s,
    People p,
    Addresses a,
    Zips z
WHERE s.peopleId = p.peopleId and
    p.peopleId = a.peopleId and
    a.zip = z.zip
ORDER BY s.peopleId ASC;

-- ---- query 1 - all Staff Information -----------------------------------
SELECT * FROM StaffInfo;

Sample Data:

<table>
<thead>
<tr>
<th>peopleId</th>
<th>dateHired</th>
<th>hiredPosition</th>
<th>dateLeft</th>
<th>currentPosition</th>
<th>cellPhone</th>
<th>firstName</th>
<th>lastName</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
<th>address</th>
<th>street1</th>
<th>street2</th>
<th>city</th>
<th>state</th>
<th>zip</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
<th>address</th>
<th>street1</th>
<th>street2</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>people004</td>
<td>2007-05-12</td>
<td>salesclerk</td>
<td>[null]</td>
<td>manager</td>
<td>914-555-55...</td>
<td>Jon</td>
<td>Sitzer</td>
<td>845-555...</td>
<td>home</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>people005</td>
<td>2015-02-23</td>
<td>salesclerk</td>
<td>[null]</td>
<td>salesclerk</td>
<td>914-666-66...</td>
<td>Marisa</td>
<td>Sumter</td>
<td>845-666...</td>
<td>cell</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>people006</td>
<td>2016-01-01</td>
<td>salesclerk</td>
<td>[null]</td>
<td>salesclerk</td>
<td>914-777-77...</td>
<td>Mandy</td>
<td>Mishra</td>
<td>845-777...</td>
<td>home</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>people008</td>
<td>2007-05-12</td>
<td>salesclerk</td>
<td>2007-07-30</td>
<td>salesclerk</td>
<td>914-111-11...</td>
<td>Janice</td>
<td>Jones</td>
<td>845-111...</td>
<td>home</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>people011</td>
<td>2007-05-12</td>
<td>owner</td>
<td>[null]</td>
<td>owner</td>
<td>914-222-22...</td>
<td>Gary</td>
<td>Carney</td>
<td>845-222...</td>
<td>home</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>email</th>
<th>addressId</th>
<th>street1</th>
<th>street2</th>
<th>city</th>
<th>state</th>
<th>status</th>
<th>zip</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
<th>address</th>
<th>street1</th>
<th>street2</th>
<th>city</th>
<th>state</th>
<th>zip</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
<th>address</th>
<th>street1</th>
<th>street2</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>jon.sitzer</td>
<td>address004</td>
<td>[null]</td>
<td>456 Manor</td>
<td>Poughkeepsie</td>
<td>New York</td>
<td>12601</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[null]</td>
<td>address005</td>
<td>[null]</td>
<td>23 Alamo</td>
<td>Rhinebeck</td>
<td>New York</td>
<td>12572-1234</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m.mishra45</td>
<td>address006</td>
<td>Royal Crest Apart</td>
<td>22B Royal Crest Pl</td>
<td>Hyde Park</td>
<td>New York</td>
<td>12538-2256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JaniceJones</td>
<td>address008</td>
<td>[null]</td>
<td>45 Zachary Way</td>
<td>Poughkeepsie</td>
<td>New York</td>
<td>12602-4545</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gary.carne</td>
<td>address011</td>
<td>[null]</td>
<td>868 Church Street</td>
<td>Poughkeepsie</td>
<td>New York</td>
<td>12602</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Views

-- ---- query 2. - search for Staff (when you know part of the first name)
-- list staff info for first names that have the letters 'ma' in it
SELECT firstName, LastName, cellPhone, phone, phoneType, email
FROM StaffInfo
WHERE firstName Like 'Ma%'
ORDER BY firstName DESC;

Sample Data:

-- ---- query 3. - search for current Staff
-- list current staff
SELECT firstName, LastName, dateHired, hiredPosition, position
FROM StaffInfo
WHERE dateLeft is Null
ORDER BY dateHired DESC;

Sample Data:

-- ---- query 4. - search for Staff who have left
-- list staff who are no longer with the company
SELECT firstName, LastName, dateHired, hiredPosition, position
FROM StaffInfo
WHERE dateLeft is not Null;

Sample Data:
Views

CustomerInfo – A view with 4 reports to obtain Customer Information from People, Addresses, Zips, Customers

CREATE VIEW CustomerInfo
AS
SELECT c.peopleId, p.firstName, p.lastName, p.phone, p.phoneType, p.email, a.addressId, a.street1, a.street2, a.zip, z.city, z.state
FROM Customers c,
    People p,
    Addresses a,
    Zips z
WHERE c.peopleId = p.peopleId and
    p.peopleId = a.peopleId and
    a.zip = z.zip
ORDER BY c.peopleId ASC;

-- ---- query 1 - all Customer Information -----------------------------
SELECT * FROM CustomerInfo;

Sample Data:

```
<table>
<thead>
<tr>
<th>peopleId</th>
<th>firstName</th>
<th>lastName</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
<th>address</th>
<th>street1</th>
<th>street2</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>people007</td>
<td>Alan</td>
<td>Laboureur</td>
<td>845-440-1...</td>
<td>work</td>
<td>alan@laboureur...</td>
<td>address007</td>
<td>Park Manor Ap...</td>
<td>4558 Spring</td>
<td>Hyde Park</td>
<td>New York</td>
<td>12538-45...</td>
</tr>
<tr>
<td>people009</td>
<td>Jane</td>
<td>Doe</td>
<td>845-555-2...</td>
<td>cell</td>
<td>3d2456@gmail...</td>
<td>address009</td>
<td>Cherry Condomi...</td>
<td>21 Cherry Hill</td>
<td>Red Hook</td>
<td>New York</td>
<td>12571</td>
</tr>
<tr>
<td>people10</td>
<td>Mary</td>
<td>Marist</td>
<td>914-555-1...</td>
<td>cell</td>
<td>mmarist6496...</td>
<td>address10</td>
<td>[null]</td>
<td>62 Hilltop Road</td>
<td>Rhinecliff</td>
<td>New York</td>
<td>12573</td>
</tr>
<tr>
<td>people12</td>
<td>Candie</td>
<td>Kane</td>
<td>846-555-4...</td>
<td>home</td>
<td>candie.kane12...</td>
<td>address12</td>
<td>[null]</td>
<td>91-208 Main...</td>
<td>Beacon</td>
<td>New York</td>
<td>12508-19...</td>
</tr>
</tbody>
</table>
```

-- ---- query 2 - By city -----------------------------------------------
-- list customer info for cities that start with 'R'
SELECT firstName, LastName, phone, phoneType, email, city
FROM CUSTOMERINFO
where city Like 'R%';

Sample Data:

```
<table>
<thead>
<tr>
<th>firstName</th>
<th>lastName</th>
<th>phone</th>
<th>phoneType</th>
<th>email</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>Doe</td>
<td>845-555-23...</td>
<td>cell</td>
<td><a href="mailto:3d2456@gmail.com">3d2456@gmail.com</a></td>
<td>Red Hook</td>
</tr>
<tr>
<td>Mary</td>
<td>Marist</td>
<td>914-555-12...</td>
<td>cell</td>
<td><a href="mailto:mmarist6496@marist.edu">mmarist6496@marist.edu</a></td>
<td>Rhinecliff</td>
</tr>
</tbody>
</table>
```
Views

---

ProductInfo – A view with 2 reports to obtain Product Information from Vendors, Products, ProductHistory
---

CREATE VIEW ProductInfo
AS
SELECT c.vendorId, c.vendorName, h.vendorProductId, p.productType, p.productName, p.productSize, p.productColor, p.productQty, p.currentCostUSD, p.productDesc, h.dateBought, h.historyCostUSD
FROM Vendors c,
    Products p,
    ProductHistory h
WHERE c.vendorId = p.vendorId and
    p.vendorProductId = h.vendorProductId;

--- ---- query 1 - all Product Information ----------------------

SELECT * FROM ProductInfo;

Sample Data:

![Table 1](image1.png)

![Table 2](image2.png)
Views

-- ---- query 2 - List of History of cost and current cost for each product -------

SELECT vdrProdId, Name, Size, Qty, currentCostUSD, dateBought, historyCostUSD
FROM ProductInfo
ORDER BY vdrProdId, dateBought;

Sample Data:
Reports

Reports that list all from each of the tables. Results were listed with each table create:

---

SELECT * FROM People;
SELECT * FROM Vendors;
SELECT * FROM VendorContacts;
SELECT * FROM Staff;
SELECT * FROM Customers;
SELECT * FROM Zips;
SELECT * FROM Addresses;
SELECT * FROM Products;
SELECT * FROM ProductHistory;
SELECT * FROM Arrangements;
SELECT * FROM ArrangementItems;
SELECT * FROM ArrangementItemsList;
SELECT * FROM Inventory;

---

List Items used in an Arrangement and their quantity
Report to obtain a list of items needed for an arrangement.

---

SELECT a.arrId, p.itemId, p.itemName, apl.itemQty
FROM Arrangements a INNER JOIN ArrangementItemsList apl ON a.arrId = apl.arrId
INNER JOIN ArrangementItems p ON apl.itemId = p.itemId
WHERE a.arrId = 'arr001'
ORDER BY p.itemId ASC;

Sample Data:

<table>
<thead>
<tr>
<th>arrId</th>
<th>itemId</th>
<th>itemName</th>
<th>itemQty</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr01</td>
<td>item01</td>
<td>Country Metal Pall</td>
<td>1</td>
</tr>
<tr>
<td>arr01</td>
<td>item02</td>
<td>Lavender Roses</td>
<td>8</td>
</tr>
<tr>
<td>arr01</td>
<td>item03</td>
<td>Alstroemeria</td>
<td>5</td>
</tr>
<tr>
<td>arr01</td>
<td>item04</td>
<td>Statice</td>
<td>6</td>
</tr>
<tr>
<td>arr01</td>
<td>item05</td>
<td>Yellow Daisies</td>
<td>5</td>
</tr>
<tr>
<td>arr01</td>
<td>item06</td>
<td>Floral Foam</td>
<td>1</td>
</tr>
<tr>
<td>arr01</td>
<td>item07</td>
<td>Floral tape</td>
<td>1</td>
</tr>
<tr>
<td>arr01</td>
<td>item015</td>
<td>Greenery</td>
<td>4</td>
</tr>
</tbody>
</table>
Reports

--- -----------------------------

**Vendors with no contact person**
Report finds Vendors who have no Vendor Contacts

```sql
SELECT *
FROM Vendors
WHERE vendorId not in
  (SELECT vendorId
   FROM VendorContacts)
ORDER BY vendorName;
```

Sample Data:

<table>
<thead>
<tr>
<th>vendorid</th>
<th>vendorname</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor004</td>
<td>Flowers by Alan</td>
</tr>
</tbody>
</table>

--- -----------------------------

**Vendors with no products**
Report finds Vendors who we have not ordered from

```sql
SELECT *
FROM Vendors
WHERE vendorId not in
  (SELECT vendorId
   FROM Products)
ORDER BY vendorName;
```

Sample Data:

<table>
<thead>
<tr>
<th>vendorid</th>
<th>vendorname</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor004</td>
<td>Flowers by Alan</td>
</tr>
</tbody>
</table>
Reports

--- ----------------------------------------------------------

Vendor Most Used
Report finds Vendor who we order the most from
--- ----------------------------------------------------------

SELECT v.vendorId as "Vendor ID",
    v.vendorName as "Vendor Name",
    count(v.vendorId) as "Product Orders Numbers"
FROM Vendors v INNER JOIN Products p ON v.vendorId = p.vendorId
    INNER JOIN ProductHistory ph ON p.vdrProdId = ph.vdrProdId
GROUP BY v.vendorId
ORDER BY count(v.vendorId) DESC
limit 1;

--- ----------------------------------------------------------

Find Vendor City
Finds Vendors in a particular City
--- ----------------------------------------------------------

SELECT v.vendorName as "Vendor Name"
FROM Vendors v
WHERE v.vendorId in
    (SELECT distinct vc.vendorId
    FROM VendorContacts vc
    WHERE vc.peopleId in
        (SELECT p.peopleId
        FROM People p
        WHERE p.peopleId in
            (SELECT a.peopleId
            FROM Addresses a
            WHERE a.city in ('Miami')
        )
    )
ORDER BY v.vendorName ASC;
Reports

---  ---------------------------------------------------------------------------------------------------

**AddVendors** - Calls the Stored Procedure AddVendors (test data is there)
---  ---------------------------------------------------------------------------------------------------

-- example:
-- Select AddVendor('vendor004', 'WonderlandFlorist.com');

-- Use this below - remove the dashes first, replace the information
-- SELECT AddPeople('vendorId', 'vendorName');

---  ---------------------------------------------------------------------------------------------------

**AddProducts** - Calls the Stored Procedure AddProducts (test data there)
---  ---------------------------------------------------------------------------------------------------

-- example1:
-- SELECT AddProducts('Y', 'G10026BH', 'vendor003', 'flower', 'rose', 'long stem', 'red', 12, 15, NULL, 15, '04/30/2017', 'item034', 'red roses', 12);
-- SELECT AddProducts('N', 'G10026B2', 'vendor003', 'flower', 'rose', 'long stem', 'lavender', 12, 15, NULL, 15, '04/30/2017', 'item002', 'lavendar roses', 12);

-- Use this below - remove the dashes first, replace the information
-- SELECT AddProducts('productNew', 'newVendorId', 'vendorId', 'newType', 'newName', 'newSize', 'newColor', 'newQty', 'newCurrentCostUSD', 'newDesc', 'historyCostUSD','dateBought', 'newItemId', 'newItemName', 'newInvQty');

---  ---------------------------------------------------------------------------------------------------

-- There are more reports attached to all the views and stored procedures
---  ---------------------------------------------------------------------------------------------------
Roles and Security

-- Security restart for all roles, tables, views, and grants --

DROP ROLE IF EXISTS admin;
DROP ROLE IF EXISTS owner;
DROP ROLE IF EXISTS manager;
DROP ROLE IF EXISTS salesclerk;

DROP TABLE IF EXISTS People, VendorContacts, Staff, Customers, Vendors, Addresses, Products, Producthistory, Inventory, Arrangements, ArrangementItems, ArrangementItemsList, Orders CASCADE;

DROP VIEW IF EXISTS VendorInfo, StaffInfo, CustomerInfo, ProductInfo CASCADE;

DROP FUNCTION IF EXISTS GetArrListofItems(text, refcursor) CASCADE;
DROP FUNCTION IF EXISTS GetProductOrderList(text, refcursor) CASCADE;
DROP FUNCTION IF EXISTS AddPeople(peopleType text, peopleId text, firstName text, lastName text, phone text, phoneType phone, email text, addressId text, street1 text, street2 text, city text, state text, zip text, dateHired date, hiredPosition pos, dateLeft date, currentPosition pos, cellPhone text, vendorId text) CASCADE;
DROP FUNCTION IF EXISTS AddVendor(vendorId text, vendorName text) CASCADE;
DROP FUNCTION IF EXISTS AddProducts(productNew text, newVdrProdId text, vendorId text, newType text, newName text, newSize text, newColor text, newQty int, newCurrentCostUSD numeric, newprodDesc text, historyCostUSD numeric, dateBought date, newItemId text, NewItemName text, newInvQty int) CASCADE;

DROP TYPE IF EXISTS phone, prod, pos CASCADE;

DROP TRIGGER IF EXISTS validPeopleInput ON People CASCADE;
DROP TRIGGER IF EXISTS validVendorInput ON Vendors CASCADE;

-- Administrator Role - has full access --

CREATE ROLE admin;
GRANT ALL ON ALL TABLES IN SCHEMA public TO admin;

-- Owner Role - Owns the business, so has full access --

CREATE ROLE owner;
GRANT ALL ON ALL TABLES IN SCHEMA public TO owner;
-- ------------------------------------------------------------------------
-- Manager Role - the manager has access to insert or update in all tables
-- ------------------------------------------------------------------------
CREATE ROLE manager;
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM manager;
GRANT ALL ON ALL TABLES IN SCHEMA public TO manager;
-- ------------------------------------------------------------------------
-- Salesclerk Role - has ability to select all
-- ------------------------------------------------------------------------
CREATE ROLE salesclerk;
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM salesclerk;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO salesclerk;
Implementation Notes

With a much larger data sample (which would take more than the time given and would make this project extremely unwieldy), there would be more chances to create more complex queries.

Also (to keep the project down to size), I have only included validation for input or update into two of the tables: People and Vendors as it would only be repetition.

I have included AddPeople, AddVendors, and AddProducts, but again due to size and time, I have not included AddArrangements (which would add to Arrangements, ArrangementItems, ArrangementItemsList, Inventory, Products and possibly Vendors, VendorContacts, People, and Addresses. When adding, it is required that the information for each be entered in its entirety. This leaves less chance for missing information.

I would like to thank Daniel Verite for the Stored Procedure from Stack Overflow (see my documentation in the sql) that I used to find “Alan” or “Labouseur” anywhere in the database.

Each arrangement uses particular items. As an arrangement is ordered a report can be run to check inventory to obtain a list of products that are need to be ordered. Once products are received, if they are new, they are added to Products, ProductHistory, Inventory, and ArrangementItems. If they are products that we have previously ordered, they are added to ProductHistory, the quantity is added to the Inventory quantity, and items in Products that may have changed are updated. As an arrangement is made, the items used will be subtracted from the Inventory quantity. *NOTE* Information in Products is from the last time they were ordered. Information in product history is a list of every time we order the product.

If anyone utilizing this database finds that a report that they would like to have is not available, please contact the administrator who will endeavor to write such a report in a timely fashion.
Known Problems and Future Enhancements

There are a few things missing and others that should be added, such as:

**More validation of all input** - When adding information there is no validation to ensure that information fits the criteria for each column in each table. Set up more Types for anything that doesn’t have a long list of possibilities.

**AddArrangements** – a stored procedure for adding arrangements should also be added.

**ArrangementInfo** - a view for all arrangement information

**Financial Information** - as the company is currently using QuickBooks to keep its accounts, there was no financial information included in this database other than the cost of an arrangement, and the cost of a product. I would like to add the financials to the database, but it is, of course, up the company.

**Missing information** – another procedure should be added that finds anything that is NULL in each column in each table.

**Update missing, changed or incorrect information** – another procedure that changes the information that was missing or incorrect from each column of each table (once that information has been found).

**Subtract from Inventory** – once an arrangement has been made the items used need to be subtracted from Inventory.

The **Orders table**, added at the last minute, should also have an OrdersHistory table, an OrdersInfo view and reports.

As this was already much too long, I added the Order table because it was a must.