Chapter 9

November 3, 2019

In [1]:
```
%load_ext sql
```

In [3]:
```
%sql postgresql://postgres:postgres@localhost:5432/analysis
```

Out[3]:
```
'Connected: postgres@analysis'
```

1 Inspecting and Modifying Data

1.1 Importing Data on Meat, Poultry, and Eggs Producers

In [4]:
```
%%sql

        CREATE TABLE meat_poultry_egg_inspect (
            est_number varchar(50) CONSTRAINT est_number_key PRIMARY KEY,
            company varchar(100),
            street varchar(100),
            city varchar(30),
            st varchar(2),
            zip varchar(5),
            phone varchar(14),
            grant_date date,
            activities text,
            dbas text
        );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[4]:
```
[]
```

In [7]:
```
%%sql

        COPY meat_poultry_egg_inspect
        FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/MPI_Directory_by_Establishment_Name.csv
        WITH (FORMAT CSV, HEADER, DELIMITER ',');
```

```
 * postgresql://postgres:***@localhost:5432/analysis
6287 rows affected.
```

Out[7]: []

In [10]:
```
%%sql

        CREATE INDEX company_idx ON meat_poultry_egg_inspect (company);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[10]: []

- we first create a table for the meat, poultry, egg inspect
- we add the natural primary key
- the activities column describes the activities of the company
- the strings are really long, so we need the text data type here
- we import our csv file and copy data into the table
- we create an index on the company column, to speed ip searches for particular companies

In [11]:
```
%%sql

        SELECT count(*) FROM meat_poultry_egg_inspect;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.
```

Out[11]: [(6287,)]

- we check the lenght of the table
  - everything is correct

1.2 Interviewing the Data Set

In [13]:
```
%%sql

        SELECT company,
               street,
               city,
               st,
               count(*) AS adress_count
        FROM meat_poultry_egg_inspect
        GROUP BY company, street, city, st
        HAVING count(*) > 1
        ORDER BY company, street, city, st;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
23 rows affected.
```

```
Out[13]: [('Acre Station Meat Farm', '17076 Hwy 32 N', 'Pinetown', 'NC', 2),
          ('Beltex Corporation', '3801 North Grove Street', 'Fort Worth', 'TX', 2),
          ('Cloverleaf Cold Storage', '111 Imperial Drive', 'Sanford', 'NC', 2),
          ('Crete Core Ingredients, LLC', '2220 County Road I', 'Crete', 'NE', 2),
          ('Crider, Inc.', '1 Plant Avenue', 'Stillmore', 'GA', 3),
          ('Dimension Marketing & Sales, Inc.', '386 West 9400 South', 'Sandy', 'UT', 2),
          ('Foster Poultry Farms, A California Corporation', '6648 Highway 15 North', 'Farmervi
          ('Freezer & Dry Storage, LLC', '21740 Trolley Industrial Drive', 'Taylor', 'MI', 2),
          ('JBS Souderton Inc.', '249 Allentown Road', 'Souderton', 'PA', 2),
          ('KB Poultry Processing LLC', '15024 Sandstone Dr.', 'Utica', 'MN', 2),
          ('Lakeside Refrigerated Services', '2600 Oldmans Creek Road', 'Swedesboro', 'NJ', 2)
          ('Liberty Cold Storage', '1310 Remington Blvd.', 'Bolingbrook', 'IL', 2),
          ('M.G. Waldbaum Company', '120 Tower Street', 'Gaylord', 'MN', 3),
          ('Midway International Logistics LLC', '948 Bradley Street', 'Watertown', 'NY', 2),
          ('Nordic Logistics and Warehousing, LLC', '220 Nordic Way', 'Pooler', 'GA', 2),
          ('OK Foods, Inc.', '3921 Reed Lane', 'Fort Smith', 'AR', 2),
          ('Pacific Produce Corporation', '220 East Harmon Industrial Park Road', 'Tamuning',
          ('Payless Distribution Center (PDC)', '370 Mendioka Street', 'Dededo', 'GU', 2),
          ('Piatkowski Riteway Meats Inc.', '3555 Witmer Road', 'Niagara Falls', 'NY', 2),
          ('Preferred Freezer Services', '2700 Trade Street', 'Chesapeake', 'VA', 2),
          ('THE AMERICAN PIG', '25 MEADOW ROAD', 'ASHEVILLE', 'NC', 2),
          ('The Classic Jerky Company', '21655 Trolley Industrial Drive', 'Taylor', 'MI', 2),
          ('United States Cold Storage Inc.', '11801 NW 102 Road', 'Medley', 'FL', 2)]
```

- we group companies by unique combinations of the company, strett, city and st columns then we use count(*) which returns the number of rows for each combination of those columns and gives it the alias adress_count

1.2.1 Checking for Missing Values

```
In [14]: %%sql

         SELECT st,
                count(*) AS st_count
         FROM meat_poultry_egg_inspect
         GROUP BY st
         ORDER BY st;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
57 rows affected.
```

```
Out[14]: [('AK', 17),
          ('AL', 93),
          ('AR', 87),
```

```
('AS', 1),
('AZ', 37),
('CA', 666),
('CO', 121),
('CT', 55),
('DC', 2),
('DE', 22),
('FL', 322),
('GA', 206),
('GU', 14),
('HI', 47),
('IA', 149),
('ID', 38),
('IL', 348),
('IN', 82),
('KS', 69),
('KY', 110),
('LA', 49),
('MA', 101),
('MD', 75),
('ME', 24),
('MI', 177),
('MN', 160),
('MO', 158),
('MP', 4),
('MS', 69),
('MT', 23),
('NC', 212),
('ND', 22),
('NE', 110),
('NH', 18),
('NJ', 244),
('NM', 28),
('NV', 35),
('NY', 346),
('OH', 186),
('OK', 59),
('OR', 86),
('PA', 364),
('PR', 84),
('RI', 27),
('SC', 55),
('SD', 24),
('TN', 113),
('TX', 387),
('UT', 71),
('VA', 111),
('VI', 2),
```

```
 ('VT', 27),
 ('WA', 139),
 ('WI', 184),
 ('WV', 23),
 ('WY', 1),
 (None, 3)]
```

- this query is a simple count it counts each state postal code (st) and gives it a alias

```
In [15]: %%sql

         SELECT est_number,
                company,
                city,
                st,
                zip
         FROM meat_poultry_egg_inspect
         WHERE st IS NULL;

 * postgresql://postgres:***@localhost:5432/analysis
3 rows affected.


Out[15]: [('V18677A', 'Atlas Inspection, Inc.', 'Blaine', None, '55449'),
          ('M45319+P45319', 'Hall-Namie Packing Company, Inc', None, None, '36671'),
          ('M263A+P263A+V263A', 'Jones Dairy Farm', None, None, '53538')]
```

- we want to find out where the NULL values are coming from

1.2.2 Checking for Inconsistent Data Values

```
In [24]: %%sql

         SELECT company,
                count(*) AS company_count
         FROM meat_poultry_egg_inspect
         GROUP BY company
         ORDER BY company ASC
         LIMIT 350;

 * postgresql://postgres:***@localhost:5432/analysis
350 rows affected.


Out[24]: [('121 In-Flight Catering LLC', 1),
          ('165368 C. Corporation ', 1),
          ('1732 Meats LLC', 1),
          ('1st Original Texas Chili Company, Inc.', 1),
          ('290 West Bar & Grill', 1),
```

```
('3 Little Pigs LLC', 1),
('3-A Enterprises', 1),
('3282 Beaver Meadow Road LLC', 1),
('3D Meats, LLC', 1),
('4 Frendz Meat Market', 1),
('4-L Processing', 1),
('41 Market', 1),
('412 Meat Processing Inc.', 1),
('458 1/2 South Broadway Meat Inc', 1),
('4G Meat Processing LLC', 1),
('50th State Poultry', 1),
('58 Place Seafood Inc.', 1),
('5th Generation Adams Farm', 1),
('701 Foods, Inc.', 1),
('814 Americas, Inc.', 1),
('86-17 Northern Blvd. Corp.', 1),
('888 Food Company', 1),
('A & A Finest', 1),
('A & A Halal Distributors', 1),
('A & B Famous Gefilte Fish Inc.', 1),
('A & G Food Service LLC', 1),
('A & M Cold Storage', 1),
('A & R Packing Co., Inc.', 1),
('A & S Distributors', 1),
('A & S Produce Inc.', 1),
('A Casa Enterprises LLC', 1),
('A Full Measure Catering', 1),
('A G Specialty Foods', 1),
('A La Carte Foods, Inc.', 1),
("A To Z Kosher Meat Prod's In", 1),
('A to Z Portion Control Meats, Inc.', 1),
('A&M Packing LLC', 1),
('A&S & Son', 1),
('A&W Country Meats, Inc.', 1),
('A-1 Meat Inc.', 1),
('A. Altieri & Sons', 1),
('A. Concepcion Hnos', 1),
('A. Decoite Packing House, Inc.', 1),
('A. F. Wendling, Inc.', 1),
('A. Gemmen & Sons, Inc.', 1),
('A. Gimenez Trading LLC', 1),
('A. I. Foods', 1),
('A. N. Deringer, Inc.', 2),
('A. S. K. Foods Inc.', 1),
('A. Tarantino & Sons', 1),
('A.A. Rubashkin & Sons', 1),
('A.B. Vannoy Hams', 1),
('A.F.I. Food Service L.L.C.', 1),
```

```
("A.J. Cetak's Meat Market", 1),
('A.J. Sons, Inc.', 1),
("A.J.'s Lena Maid Meats, Inc.", 1),
('A.L. Beck & Sons, Inc.', 1),
('A.N. Deringer Inc.', 1),
('A.N. Deringer, Inc.', 1),
('A.S.K. Foods Inc.', 1),
('A.T.A. Meat Company, Inc.', 1),
('AA Meat Products Inc.', 2),
('AA Poultry Processing, LLC', 1),
('ABF Packing, Inc.', 1),
('ACC Central Kitchen LLC', 1),
('ACME Jerky, LLC', 1),
('AFS Classico, LLC', 1),
('AG Food Products Corp.', 1),
('AGA Investments Inc.', 1),
('AGRO Merchants Oakland LLC', 2),
('AGRO Merchants Oakland, LLC', 1),
('ALFREDO AIELLO ITALIAN FOODS, INC.', 1),
('AMERICAN FOOD SERVICE', 1),
('AMICK FARMS, LLC', 1),
('AMPC, Inc.', 2),
('ARC Industries', 1),
('ASU Food Safety and Product Development Lab', 1),
('ATG Transportation LLC', 1),
('ATK Foods, Inc.', 1),
('ATM International USA, Inc.', 1),
('ATPAC Inc.', 1),
('AU, LAU and Associates, Inc.', 1),
('AVA Pork Products, Inc.', 1),
('AVF Holding LLC', 1),
('AZ Gourmet Foods Inc.', 1),
('AZ Grass Fed Beef', 1),
('Aala Meat Market, Inc.', 1),
('Abanto Forwarding, Inc.', 1),
('Abattoir Associates Inc.', 1),
("Abbott's Meat, Inc.", 1),
('Abbyland Foods, Inc.', 2),
('Abbyland Pork Pack, Inc.', 1),
("Abe's Finest Meats", 1),
("Abe's Kosher Meats LLC", 1),
('Abeles & Heymann, LLC', 1),
('Able Freight Services Inc.', 1),
('Abner Snack Foods, Inc.', 2),
('Abuelito Meat Inc.', 1),
('Abundant Foods', 1),
('Academy Packing Co Inc', 1),
('Acadian Fine Foods, LLC', 1),
```

```
('Accu-Ray Inspection Services', 1),
('Achatz Handmade Pie Company LLC', 1),
('Achatz Soup From Skratzch', 1),
('Acme Delivery Service', 1),
('Acoreana Manufacturing, Co.', 1),
('Acre Station Meat Farm', 2),
('Activ International, Inc.', 1),
('Ada Valley Gourmet Foods', 1),
('Adam Farms, LLC', 1),
('Adams Farm Slaughterhouse LLC', 1),
("Addielee's Inc.", 1),
('Adena Poultry', 1),
('Adesa International LLC', 1),
('Adirondack Meat Co.', 1),
("Adolf's Meat Products", 1),
('Advance Pierre Foods, Inc', 2),
('Advance Pierre Foods, Inc.', 1),
('AdvancePierre Foods', 1),
('AdvancePierre Foods, Inc', 1),
('AdvancePierre Foods, Inc.', 7),
('Advertising Resources, Inc', 1),
('Afandina Halal', 1),
('Afya Brands International, Inc.', 1),
('Against The Grain Gourmet', 1),
('Agri Star Meat and Poultry, LLC', 1),
('Agro Merchant Group Inc.', 1),
('Agro Merchants Group Charleston LLC', 1),
('Airfrigo USA Inc/Seafrigo Coldstorage Fairmont', 1),
('Ajinomoto Toyo Frozen Noodle, Inc.', 1),
('Ajinomoto Windsor Inc.', 2),
('Ajinomoto Windsor, Inc.', 7),
('Ajuua', 1),
('Al & John Inc.', 1),
('Al - Baghdadi Food Inc.', 1),
("Al Lunardi & Son's Meat Co Inc", 1),
('Al Shabrawy Inc.', 1),
("Al's Hickory House BBQ", 1),
("Al's Wholesale Meats, Inc.", 1),
('Al-Marwa L. L. C.', 1),
('Alabama CatfishLLC', 1),
('Alaska Commercial Co.', 1),
('Alaska Natural Meats, Ltd.', 1),
('Alaska Sausage Company, Inc.', 1),
('Alatrade Foods', 2),
('Alatrade Foods LLC', 1),
('Alba Cold Storage', 1),
('Albert Lea Select Foods Inc.', 1),
("Alberto's Meat Shop", 1),
```

```
('Albertville Quality Foods', 2),
("Albie's Foods, Inc.", 1),
('Albion Locker', 1),
('Alcor Foods, Inc.', 1),
('Alderfer, LLC', 1),
('Aldon Food Corporation', 1),
('AleCon Enterprises Inc.', 1),
('Alef Sausage', 1),
('Alena Foods', 1),
("Alewel's Country Meats", 1),
('Alex & George Wholesale Meats, Inc.', 1),
('Alex Froehlich Packing Company', 1),
("Alex's Deli", 1),
("Alex's Meat & Provision", 1),
('Alexander & Hornung', 1),
("Alexander's Ham Co.", 1),
('Alexandra Foods', 1),
('Alexis Wholesale Inc.', 1),
('Alfresco Pasta, LLC', 1),
('Ali International Inc.', 1),
('All American Cold Storage', 1),
('All American Meats, Inc.', 1),
('All Foods LLC', 1),
("All People's Food", 1),
('Alle-Pia', 1),
('Alleghany Highlands Agricultural Center, LLC', 1),
('Allen Brothers, Inc', 1),
('Allen Harim Foods, LLC.', 1),
('Allentown Refrigerated Terminals Inc.', 1),
('Allied Caribbean Distribution', 1),
('Allied Frozen Storage inc.', 1),
('Allied Poultry Sales Co., Inc.', 1),
('Allonco Inc.', 1),
('Alma Cold Storage, Inc.', 1),
('Alma Foods, LLC', 1),
('Almi Inc.', 1),
('Alpha Foods Co.', 1),
('Alpha Omega Distributors, LTD', 1),
('Alpine Meats Inc.', 1),
('Alpine Sausage Company', 1),
('Alta Vista Locker', 1),
('Alto Valle Foods, Inc.', 1),
('AmBar Foods', 1),
('Amana Meat Shop & Smokehouse', 1),
('Amaro Foods Enterprise Inc', 1),
('Amba Ham Company Inc.', 1),
('Ambassador Meat Dist. Inc.', 1),
('Amboy Group, LLC', 1),
```

```
('Amend Packing Co.', 1),
('AmeriCold Logistics', 7),
('AmeriCold Logistics Crete Dist', 1),
('AmeriCold Logistics, Inc.', 1),
('AmeriCold Logistics, LLC', 2),
('AmeriQual Group, LLC', 1),
('AmeriQual Packaging', 1),
('America New York Ri Wang Food Group', 1),
("America's Best Steak", 1),
("America's Catch Inc", 1),
("America's Custom Brokers, Inc.", 1),
('American Alliance Partners LLC', 1),
('American Beef Packers, Inc.', 1),
('American Butchers, Inc.', 1),
('American Butchers, LLC', 1),
('American Cold Storage', 4),
('American Cold Storage-IN. Div.', 1),
('American Consolidation & Logistics', 1),
('American Custom Meats LLC', 1),
('American Egg Products, LLC', 1),
('American Food Export Service', 1),
('American Food Services, LLC', 1),
('American Food Supplies LLC', 1),
('American Food Systems Inc.', 1),
('American Halal Meat', 1),
('American Halal Meat Inc.', 1),
('American Kitchen Delights Inc.', 1),
('American Laboratories, Inc', 1),
('American Laboratories, Inc.', 2),
('American Meat Company', 1),
('American Meat Processing Co., Inc.', 1),
('American Outdoor Products, Inc', 1),
('American Pasteurization Co LLC', 1),
('American Pasteurization Company', 1),
('American Samoa Government', 1),
('American Skin Food Group LLC', 1),
('American Soy Products, Inc.', 1),
('American Specialty Food, Inc.', 1),
('Americold', 11),
('Americold - Ontario, Oregon', 1),
('Americold Corporation', 2),
('Americold Logistics', 21),
('Americold Logistics Inc', 2),
('Americold Logistics Inc.', 1),
('Americold Logistics LLC', 8),
('Americold Logistics, Inc.', 3),
('Americold Logistics, LLC', 9),
('Americold Logistics, LLC.', 1),
```

```
('Americold, LLC', 1),
('Ameripack Foods LLC', 1),
('Ameristar Meats, Inc', 1),
('Ameristar Meats, Inc.', 2),
('Amick Farms, LLC', 1),
("Amigo's Mexican Foods", 1),
('Amigos Frozen Solutions', 1),
('Amilas Foods', 1),
('Aminchi Foods International, Inc.', 1),
('Amity Packing Co. Inc.', 1),
('Amor Nino Foods, Inc.', 1),
('Amy Food Inc.', 1),
('Anchor Distribution Service, Inc.', 1),
('Anco Poultry Processing', 1),
('Anderson Boneless Beef, Inc.', 1),
('Anderson Produce Company, Inc.', 1),
('Andrade Slaughterhouse', 1),
("Andy's Deli & Mikolajczk", 1),
("Andy's Meats Inc.", 1),
('Angelina Foods', 1),
('Angus Meats, Inc.', 1),
("Angy's Landolfi Food Group LLC", 1),
('Anichini Bros Inc', 1),
('Ankeny Cold Storage, LLC', 1),
("Anna's Kitchen, Inc.", 1),
('Annex Food Company', 1),
('Ansaldos Sausage Corp.', 1),
('Antonio Pena Distributors', 1),
('Apache Foods LLC', 1),
('Apex Cold Storage', 1),
('Apex Cold Storage Co.', 1),
('Apollo Ship Chandlers Inc.', 1),
('Apostolic Christian World Relief', 1),
('Appalachian Meats LLC', 1),
('Appalachian Meats, LLC', 1),
('Appalachian Smoked Meats', 1),
('Appetito Provisions Co Inc', 1),
('Apple Valley Farms', 1),
('Applied Process Systems', 1),
('Aquasouth', 1),
('Ara Food Corp.', 1),
('Arapahoe Foods Inc.', 1),
('Arch Foods Inc', 1),
("Archie's Foods, Inc.", 1),
('Arck Foods, Inc.', 1),
('Arctic Cold Storage', 1),
('Arctic Foods Inc', 1),
('Arctic Frozen Foods LLC', 1),
```

```
('Arctic Logistics LLC', 1),
('Argo Merchants Oakland', 1),
('Argus Food Processing Corporation', 1),
('Ariake USA, Inc.', 1),
('Aries Prepared Beef Co.', 1),
('Aristocrat Products, Inc', 1),
('Arkansas Department of Corrections', 1),
('Arkansas Refrigerated Services', 2),
('Arko Veal Company, Inc.', 1),
('Arlindo Catering Inc.', 1),
('Arm National Foods', 1),
('Armour - Eckrich Meats, LLC', 1),
('Armour-Eckrich Meats LLC', 3),
('Armour-Eckrich Meats, Inc.', 1),
('Armour-Eckrich Meats, LLC', 2),
("Arnold's & Eddies Foods Inc.", 1),
('Arsho Meat Products', 1),
('Artisan Bread Co., LLC', 1),
('Arveybell Farm Co.', 1),
('Aryzta LLC', 2),
('Asahi Foods Inc.', 1),
('Asheville Packing Co.', 1),
('Ashland Cold Storage Co.', 1),
('Ashland Cold Storage Company', 1),
('Ashland Sausage Company', 1),
('Ashley Foods Inc.', 1),
('Ashton Farms Custom Meats', 1),
('Asiago Foods-USA, Inc.', 1),
('Asianic Inc.', 1),
('Aspen Foods Div Koch Meat Co.', 1),
('Aspen Hollow Sheep Station Mobile Harvest Unit', 1),
('Assemblers Inc.', 1),
('Associated Brands Inc.', 1),
('Associated Grocers of FL', 1),
('Astra Foods Inc.', 1),
('At Last! Gourmet Foods', 1),
('Athena Foods', 1),
('Athens Foods, Inc.', 1),
('Athens Sausage Co.', 1),
('Atkins Sheep Ranch Inc.', 1),
('Atkins Sheep Ranch, Inc.', 1),
('Atlanta Meat Company, Inc.', 1),
('Atlanta Sausage Company', 1),
('Atlantic Coast Freezers LLC', 1),
('Atlantic Coast Freezers, LLC', 1),
('Atlantic Veal & Lamb Inc', 1),
('Atlantic Veal & Lamb LLC', 1),
('Atlas Inspection, Inc.', 1),
```

```
          ('Atlas Meat Company', 1),
          ('Attilio Esposito Inc.', 1),
          ('Au Bon Canard Foie Gras, Inc.', 1),
          ('Aufschnitt Meats (W & L Kosher Meats)', 1),
          ("Augustine's Italian Village Inc.", 1),
          ("Aunt Kitty's Foods Inc", 1),
          ('Aurnish Enterprises Corp.', 1),
          ('Aurora Packing Company, Inc.', 1),
          ('Authentic Brands of Chicago', 1)]
```

- we can see that some values are inconsistent

1.2.3 Checking for Malformed Values Using lenght()

```
In [26]: %%sql

         SELECT length(zip),
                count(*) AS length_count
         FROM meat_poultry_egg_inspect
         GROUP BY length(zip)
         ORDER BY length(zip) ASC;

 * postgresql://postgres:***@localhost:5432/analysis
3 rows affected.


Out[26]: [(3, 86), (4, 496), (5, 5705)]
```

- we find out, that some zip codes are not 5 characters
- they are taken with the leading zeros and as an integer they are not taken

```
In [28]: %%sql

         SELECT st,
                count(*) AS st_count
         FROM meat_poultry_egg_inspect
         WHERE length(zip) < 5
         GROUP BY st
         ORDER BY st ASC;

 * postgresql://postgres:***@localhost:5432/analysis
9 rows affected.


Out[28]: [('CT', 55),
          ('MA', 101),
          ('ME', 24),
          ('NH', 18),
          ('NJ', 244),
```

```
('PR', 84),
('RI', 27),
('VI', 2),
('VT', 27)]
```

- using the WHERE clause we can check the details of the result to see which states there ZIP codes are coming from

- so far we have to correct these errors:
  - **missing values for three rows in the st column**
  - **inconsistent spelling of at least one company's name**
  - **inaccurate ZIP codes due to file conversion**

1.3 Modifying Tables, Columns, and Data

1.3.1 Modifying Tables with ALTER TABLE

- ALTER TABLE table ADD COLUMN colum data_type;
  - the code for adding a colum to a table
- ALTER TABLE table DROP COLUMN column;
  - remove a column
- ALTER TABLE table ALTER COLUMN column SET DATA TYPE data_type;
  - to change the data type of a colum
- ALTER TABLE table ALTER COLUMN column SET NOT NULL;
  - adding a NOT NULL constrains to a column
- ALTER TABLE table ALTER COLUMN column DROP NOT NULL;
  - removing the NOT NULL constraint

1.3.2 Modifying Values with UPDATE

- UPDATE table SET column = value;
  - modifies the data in a column in all rows or in a subset of rows tht meet the condition

- UPDATE table SET column_a = value, column_b = value;
  - we first pass UPDATE to the table to update
  - SET clause contains the values to change
  - the new value can be a string, number, the name of another column or even a query or expression that generates value

- UPDATE table SET column = value WHERE criteria;
  - we can restrict the update to particular rows, we add a WHERE clause with some criteria

- UPDATE table SET column = (SELECT colum FROM table_b WHERE table.column = table_b.column) WHERE EXISTS (SELECT column FROM table_b WHERE table.column = table_b.column);
  - we can also update one table with values from another table
  - the WHERE EXISTS clause is alternative

1.3.3 Creating Backup Tables

```
In [29]: %%sql

         CREATE TABLE meat_poultry_egg_inspect_backup AS
         SELECT * FROM meat_poultry_egg_inspect;

 * postgresql://postgres:***@localhost:5432/analysis
6287 rows affected.


Out[29]: []
```

- before modifying a table, it is a good idea to make a copy for reference and backup
- a confrmation that everything is backuped

```
In [30]: %%sql

         SELECT
             (SELECT count(*) FROM meat_poultry_egg_inspect) AS original,
             (SELECT count(*) FROM meat_poultry_egg_inspect_backup) AS backup;

 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.


Out[30]: [(6287, 6287)]
```

1.3.4 Restoring Missing Column Values

Creating a Column Copy

```
In [31]: %%sql

         ALTER TABLE meat_poultry_egg_inspect ADD COLUMN st_copy varchar(2);

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[31]: []
```

```
In [32]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET st_copy = st;

 * postgresql://postgres:***@localhost:5432/analysis
6287 rows affected.


Out[32]: []
```

- we used the **ALTER TABLE** statement which adds a column called st_copy using the same varchar data type as the original st column
- next the **UPDATE** statement's **SET** fill our newly created st_copy column with the data values in column st
- because we dont specify any criteria using WHERE every row is updated

```
In [37]: %%sql

         SELECT st,
                st_copy
         FROM meat_poultry_egg_inspect
         ORDER BY st
         LIMIT 10; -- LIMIT originally 6287 rows

 * postgresql://postgres:***@localhost:5432/analysis
10 rows affected.


Out[37]: [('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK'),
          ('AK', 'AK')]
```

Updating Rows where Values are Missing

```
In [40]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET st = 'MN'
         WHERE est_number = 'V18677A';

         UPDATE meat_poultry_egg_inspect
```

```
        SET st = 'AL'
        WHERE est_number = 'M45319+P45319';

        UPDATE meat_poultry_egg_inspect
        SET st = 'WI'
        WHERE est_number = 'M263A+P263A+V263A';
```

```
 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.
1 rows affected.
1 rows affected.
```

```
Out[40]: []
```

Restoring Original Values

```
In [39]: %%sql

        UPDATE meat_poultry_egg_inspect
        SET st = st_copy;

        UPDATE meat_poultry_egg_inspect AS original
        SET st = backup.st
        FROM meat_poultry_egg_inspect_backup AS backup
        WHERE original.est_number = backup.est_number;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
6287 rows affected.
6287 rows affected.
```

```
Out[39]: []
```

- to restore the values from the backup colum in meat_poultry_egg_inspect we created earlier (which is the first UPDATE here)
- both column should have the identical values again
- we do that with an update that sets st to the values from the backup file

1.4 Updating Values for Consistentcy

- in several cases a single company's name was entered incosistently
- if we want to aggregate data by compamy name such inconsistencies will hinder us from doing so

```
In [ ]: %%sql

        ALTER TABLE meat_poultry_egg_inspect
        ADD COLUMN company_standart varchar(100);
```

```
In [45]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET company_standart = company;

 * postgresql://postgres:***@localhost:5432/analysis
6287 rows affected.


Out[45]: []
```

- to protect our data we add a new column to our table which we name company_standart
- we then **UPDATE** our table and **SET** the new column with the old original one, we want to protect

```
In [46]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET company_standart = 'Armour-Ekrich Meats'
         WHERE company LIKE 'Armour%';

 * postgresql://postgres:***@localhost:5432/analysis
7 rows affected.


Out[46]: []
```

- in this line we use the WHERE clause that uses the LIKE keyword
- including the wildcard % at the end of the string ARMOUR, it updates all rows that start with those characters regardless of what comes after them

```
In [47]: %%sql

         SELECT company, company_standart
         FROM meat_poultry_egg_inspect
         WHERE company LIKE 'Armour%';

 * postgresql://postgres:***@localhost:5432/analysis
7 rows affected.


Out[47]: [('Armour-Eckrich Meats LLC', 'Armour-Ekrich Meats'),
          ('Armour - Eckrich Meats, LLC', 'Armour-Ekrich Meats'),
          ('Armour-Eckrich Meats LLC', 'Armour-Ekrich Meats'),
          ('Armour-Eckrich Meats LLC', 'Armour-Ekrich Meats'),
          ('Armour-Eckrich Meats, Inc.', 'Armour-Ekrich Meats'),
          ('Armour-Eckrich Meats, LLC', 'Armour-Ekrich Meats'),
          ('Armour-Eckrich Meats, LLC', 'Armour-Ekrich Meats')]
```

- the SELECT statement results the updated company_standart
- now we have standart values for Armour-Eckrich

1.5 Repairing ZIP Codes Using Concatenation

- for the issiue with the leading zeors at the beginning of the zip codes we will use UPDATE again but this time in conjunction with the double pupe string operator (||) which performs concatenation
- concatenation combines two or more strings or non-strings into one
- for example if you insert || between abc and 123 you get abc123

```
In [48]: %%sql

         ALTER TABLE meat_poultry_egg_inspect ADD COLUMN zip_copy varchar(5);

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[48]: []
```

```
In [49]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET zip_copy = zip;

 * postgresql://postgres:***@localhost:5432/analysis
6287 rows affected.


Out[49]: []
```

```
In [51]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET zip = '00' || zip
         WHERE st IN('PR', 'VI') AND length(zip) = 3;

 * postgresql://postgres:***@localhost:5432/analysis
86 rows affected.


Out[51]: []
```

- first we create and fill the copy colum
- we modify the codes in the zip column with the leading zeors
- we do that by setting the zip value to a result of a concatenation of the string and 00
- we limit the update to only those wors where the st column has the state codes 'PD' and 'VI' using **IN** comparison operator and add a test for rows where the length of the zp is 3

```
In [52]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET zip = '0' || zip
         WHERE st IN('CT', 'MA', 'ME', 'NH', 'NJ', 'RI', 'VT') AND length(zip) = 4;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
496 rows affected.
```

Out[52]: []

- for the remaining missing 1 leading zeors we do this operatopn

In [53]:
```
%%sql

        SELECT length(zip),
                count(*) AS length_count
        FROM meat_poultry_egg_inspect
        GROUP BY length(zip)
        ORDER BY length(zip) ASC;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.
```

Out[53]: [(5, 6287)]

- we do the same query again to check if everything is okay

1.5.1 Updating Values Across Tables

In [54]:
```
%%sql

        CREATE TABLE state_regions (
                st varchar(2) CONSTRAINT st_key PRIMARY KEY,
                region varchar(20) NOT NULL
        );
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[54]: []

In [57]:
```
%%sql

        COPY state_regions
        FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/state_regions-Copy1.csv'
        WITH (FORMAT CSV, HEADER, DELIMITER ',');
```

```
 * postgresql://postgres:***@localhost:5432/analysis
56 rows affected.
```

Out[57]: []

- we are creating a new table and filling it with the data from the csv file
- we have created two columns in a state_regions table
- one containing two-character state code *st* and the other containing the region name *region*
- we set the primary key for the st which holds a unique *st-key*

```
In [58]: %%sql

         ALTER TABLE meat_poultry_egg_inspect ADD COLUMN inspection_date date;

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[58]: []
```

```
In [61]: %%sql

         UPDATE meat_poultry_egg_inspect inspect
         SET inspection_date = '2019-12-01'
         WHERE EXISTS (SELECT state_regions.region
                       FROM state_regions
                       WHERE inspect.st = state_regions.st
                              AND state_regions.region = 'New England');

 * postgresql://postgres:***@localhost:5432/analysis
252 rows affected.


Out[61]: []
```

- the **ALTER TABLE** statement creates the inspection_date column in the meat_poultry_egg_inspect table
- the **UPDATE** statement, we start by naming the table using an alias of inspect to make the code easiert to read
- we set a date value for the inspection_date column
- with **WHERE EXISTS** we connect the meat_poultry_egg_inspect table to the state_regions table
- this subquery looks for rows in the state_regions table where the region column mathces the string New Englad
- at the same time it joins the meat_poultry_egg_inspect table with the state-regions table using the *st* column from both tables

```
In [62]: %%sql

         SELECT st, inspection_date
         FROM meat_poultry_egg_inspect
         GROUP BY st, inspection_date
         ORDER BY st;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
56 rows affected.
```

Out[62]:
```
[('AK', None),
 ('AL', None),
 ('AR', None),
 ('AS', None),
 ('AZ', None),
 ('CA', None),
 ('CO', None),
 ('CT', datetime.date(2019, 12, 1)),
 ('DC', None),
 ('DE', None),
 ('FL', None),
 ('GA', None),
 ('GU', None),
 ('HI', None),
 ('IA', None),
 ('ID', None),
 ('IL', None),
 ('IN', None),
 ('KS', None),
 ('KY', None),
 ('LA', None),
 ('MA', datetime.date(2019, 12, 1)),
 ('MD', None),
 ('ME', datetime.date(2019, 12, 1)),
 ('MI', None),
 ('MN', None),
 ('MO', None),
 ('MP', None),
 ('MS', None),
 ('MT', None),
 ('NC', None),
 ('ND', None),
 ('NE', None),
 ('NH', datetime.date(2019, 12, 1)),
 ('NJ', None),
 ('NM', None),
 ('NV', None),
 ('NY', None),
 ('OH', None),
 ('OK', None),
 ('OR', None),
 ('PA', None),
 ('PR', None),
 ('RI', datetime.date(2019, 12, 1)),
```

```
('SC', None),
('SD', None),
('TN', None),
('TX', None),
('UT', None),
('VA', None),
('VI', None),
('VT', datetime.date(2019, 12, 1)),
('WA', None),
('WI', None),
('WV', None),
('WY', None)]
```

- we updated the table with dates (we have been selected New England in this example)

1.6 Deleting Unnecessary Data

- without backup the data is gone for good
- **DELETE FROM**
  - for removing rows from a table
- **ALTER TABLE**
  - for remove a column from a table
- **DROP TABLE**
  - remove a while table froim the db

1.6.1 Deleting Rows from a Table

- DELETE FROM table_name;
  - we can remove all rows from a table or we can use WHERE to delete only the porion that matches an expression we supply
- DELETE FROM table_name WHERE expression;

```
In [63]: %%sql

         DELETE FROM meat_poultry_egg_inspect
         WHERE st IN('PR', 'VI');
```

```
 * postgresql://postgres:***@localhost:5432/analysis
86 rows affected.
```

```
Out[63]: []
```

- for example if we want our table of meat, poultry and egg we can remove the 2 other us states from the table

1.6.2 Deleting a Column from a Table

- ALTER TABLE table_name DROP COLUMN column_name;
  - we earlier created a zip_copy column as a backup to our table

```
In [64]: %%sql

         ALTER TABLE meat_poultry_egg_inspect DROP COLUMN zip_copy;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[64]: []
```

1.6.3 Deleting a Table from a DB

- DROP TABLE table_name;
  - you can just remove the table

```
In [65]: %%sql

         DROP TABLE meat_poultry_egg_inspect_backup;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[65]: []
```

1.7 Using Transaction Blocks to Save or Revert Changes

- the techniques in this chapter such as **UPDATE** or **DELETE** are final
- the only way to restore is from a backup
- you can check your changes before finalizing them and cancel the change if it's not what wo want
- **transaction block** can do that
  - **START TRANSACTION** signals the sart of the transaction block
  - **COMIT** signals the end of the block and saves all changes
  - **ROLLBACK** signals the end of the block and reverts all changes
- defining both transaction steps as one unit - if one steps fails, the other iscanceled too

```
In [69]: %%sql

         START TRANSACTION;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[69]: []

In [70]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET company = 'AGRO Merchants Oakland LLC'
         WHERE company = 'AGRO Merchantss Oakland, LLC';

 * postgresql://postgres:***@localhost:5432/analysis
0 rows affected.


Out[70]: []

In [71]: %%sql

         SELECT company
         FROM meat_poultry_egg_inspect
         WHERE company LIKE 'AGRO%'
         ORDER BY company;

 * postgresql://postgres:***@localhost:5432/analysis
3 rows affected.


Out[71]: [('AGRO Merchants Oakland LLC',),
          ('AGRO Merchants Oakland LLC',),
          ('AGRO Merchantss Oakland LLC',)]

In [75]: %%sql

         ROLLBACK;

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[75]: []
```

- we run each statement separately with **START TRANSACTION**
- the DB letting you know that any changes you make will not be permanent unless you **COMMIT** the command
- next we can run the **UPDATE** statement which changes the name by mistake
- when we view the names of the companies wich start with the ARGO using SELECT we see the typo
- Instead of reunning the UPDATE we can fix the typo running **ROLLBACK** command

1.8 Improving Performance When Updating Large Tables

- instead of adding a column and filling it with values, we can save disk space by copying the entire table and adding a populated column during the operation
- then we can rname the tables so the copy replaces the original

```
In [83]: %%sql

         CREATE TABLE meat_poultry_egg_inspect_backup AS
         SELECT *,
                 '2018-02-07'::date AS reviewd_date
         FROM meat_poultry_egg_inspect;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
6201 rows affected.
```

```
Out[83]: []
```

- in addition to selecting all the columns unsing the asterisk
- we also add a column called reviewd_date
- we provide this value a cast date data type and an alias

```
In [84]: %%sql

         ALTER TABLE meat_poultry_egg_inspect RENAME TO meat_poultry_egg_inspect_temp;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[84]: []
```

```
In [85]: %%sql

         ALTER TABLE meat_poultry_egg_inspect_backup
         RENAME TO meat_poultry_egg_inspect;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[85]: []
```

```
In [87]: %%sql

         ALTER TABLE meat_poultry_egg_inspect_temp
         RENAME TO meat_poultry_egg_inspect_backup;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[87]: []
```

- we swap the names
- the first statement renames the copy we made to the temp table
- the second renames the cop we made to the original name
- finall we rename the table that ends with _tmp to ending _backup
- the original is now called _backup and the copy with the added column is called _inspect

1.9 Try it Yourself

– In this exercise, you'll turn the meat_poultry_egg_inspect table into useful – information. You needed to answer two questions: How many of the companies – in the table process meat, and how many process poultry?

– Create two new columns called meat_processing and poultry_processing. Each – can be of the type boolean.

– Using UPDATE, set meat_processing = TRUE on any row where the activities – column contains the text 'Meat Processing'. Do the same update on the – poultry_processing column, but this time lookup for the text – 'Poultry Processing' in activities.

– Use the data from the new, updated columns to count how many companies – perform each type of activity. For a bonus challenge, count how many – companies perform both activities.

```
In [95]: %%sql

         ALTER TABLE meat_poultry_egg_inspect ADD COLUMN meat_processing bool;

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[95]: []
```

```
In [96]: %%sql

         ALTER TABLE meat_poultry_egg_inspect ADD COLUMN poultry_processing bool;

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[96]: []
```

```
In [97]: %%sql

         UPDATE meat_poultry_egg_inspect
         SET meat_processing = TRUE
         WHERE activities LIKE '%Meat Processing%';

 * postgresql://postgres:***@localhost:5432/analysis
4764 rows affected.
```

```
Out[97]: []
```

```
In [101]: %%sql

          UPDATE meat_poultry_egg_inspect
          SET poultry_processing = TRUE
          WHERE activities LIKE '%Poultry Processing%';
```

```
 * postgresql://postgres:***@localhost:5432/analysis
3728 rows affected.
```

```
Out[101]: []
```

```
In [103]: %%sql

          SELECT count(meat_processing), count(poultry_processing)
          FROM meat_poultry_egg_inspect;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.
```

```
Out[103]: [(4764, 3728)]
```