Introduction to SQL

Inspect and Modify Data

Prof. Dr. Jan Kirenz

HdM Stuttgart.

2019/09/01 (updated: 2019-09-02)

Setup

- This presentation is based on the excellent book "A Beginner's Guide to Storytelling with Data" from Anthony DeBarros: DeBarros, A. (2018). Practical SQL: A Beginner's Guide to Storytelling with Data. No Starch Press.

pw = "your_password"

```
library(DBI)
library(RPostgreSQL)

drv <- dbDriver("PostgreSQL")

con <- dbConnect(drv, dbname = "postgres",
                 host = "localhost", port = 5433,
                 user = "postgres", password = pw)
```

Inspecting and modifying data

- Data: MPI_Directory_by_Estaplishment_Name.csv
- Create Table:

```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
);
```

Inspecting and modifying data

Import data

```
COPY meat_poultry_egg_inspect
FROM '/Users/jankirenz/Documents/HdM/Vorlesungen/DataScience/ProgrammingLanguages/SQL/sq
WITH (FORMAT CSV, HEADER, DELIMITER ',');
```

Inspecting and modifying data

Create index

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

```
SELECT *
FROM meat_poultry_egg_inspect
LIMIT 20
```

Show 4 entries Search:

	est_number	company	street	city	st	zip	phone	gr
1	M46712+P46712	121 In-Flight Catering LLC	45 Rason Road	Inwood	NY	11096	(718) 663-4612	20
2	M13561+P13561	165368 C. Corporation	5617 Hoover Street, Suite A	Houston	TX	77092	(713) 263-1944	20
3	M46724+P46724	1732 Meats LLC	6250 Baltimore Pike	Yeadon	PA	19050	(267) 879-7214	20
4	M7067+P7067	1st Original Texas Chili Company, Inc.	3313 N. Jones Street	Fort Worth	TX	76106	(817) 626-0983	20

Showing 1 to 4 of 5 entries Previous 1 2 Next

Inspecting and modifying data

Inspect data

- Count rows:

```
-- Count the rows imported:
SELECT count(*)
FROM meat_poultry_egg_inspect;
```

count
6287

Inspecting and modifying data

Inspect data

- Finding multiple companies at the same address

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

Show 4 entries Search:

	company	street	city	st	address_count
1	Acre Station Meat Farm	17076 Hwy 32 N	Pinetown	NC	2
2	Beltex Corporation	3801 North Grove Street	Fort Worth	TX	2
3	Cloverleaf Cold Storage	111 Imperial Drive	Sanford	NC	2
4	Crete Core Ingredients, LLC	2220 County Road I	Crete	NE	2

Showing 1 to 4 of 23 entries

Previous 1 2 3 4 5 6 Next

Inspecting and modifying data

Missing values

- Check wether any rows are missing
- How many of the companies are in each state?

```
-- Grouping and counting states
SELECT st,
    count(*) AS st_count
FROM meat_poultry_egg_inspect
GROUP BY st
ORDER BY st NULLS FIRST;    --
```

- NULL values will either appear first or last in a sorted column (depending on the database).
- You can specify NULLS FIRST or NULLS LAST to an ORDER BY

Inspecting and modifying data

Show 5 entries Search:

	st	st_count
1		3
2	AK	17
3	AL	93
4	AR	87
5	AS	1

Showing 1 to 5 of 57 entries

Previous 1 2 3 4 5 … 12 Next

Inspecting and modifying data

Find missing values

- Using IS NULL to find missing values in the st column.

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

Show 5 entries Search:

	est_number	company	city	st	zip
1	V18677A	Atlas Inspection, Inc.	Blaine		55449
2	M45319+P45319	Hall-Namie Packing Company, Inc			36671
3	M263A+P263A+V263A	Jones Dairy Farm			53538

Showing 1 to 3 of 3 entries Previous 1 Next

Inspecting and modifying data

- We’ve discovered that we’ll need to add 3 missing values to the st column to clean up this table.
- Let’s look at what other issues exist in our data set and make a list of cleanup tasks.

Inspecting and modifying data

Checking inconsistent data values

- Using GROUP BY and count() to find inconsistent names

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

Show 8 entries Search:

	company	company_count
1	121 In-Flight Catering LLC	1
2	165368 C. Corporation	1
3	1732 Meats LLC	1
4	1st Original Texas Chili Company, Inc.	1
5	290 West Bar & Grill	1
6	3 Little Pigs LLC	1
7	3-A Enterprises	1
8	3282 Beaver Meadow Road LLC	1

Showing 1 to 8 of 1,000 entries

Previous 1 2 3 4 5 … 125 Next

Inspecting and modifying data

Checking for malformed values

- length() is a string function that counts the number of characters in a string

Inspecting and modifying data

Checking for malformed values

- Using length() and count() to test the zip column

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

length	length_count
3	86
4	496
5	5705

- What happend here?

Inspecting and modifying data

Checking for malformed values

- Question: What happens if you store the value "0174" as
  - text?
  - integer?

Inspecting and modifying data

Checking for malformed values

- Filtering with length() to find short zip values

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

st	st_count
CT	55
MA	101
ME	24
NH	18
NJ	244
PR	84
RI	27
VI	2
VT	27

Inspecting and modifying data

Items to correct

- 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

Inspecting and modifying data

Modifying tables, columns and data

- ALTER TABLE
- Review additional ALTER TABLE Options in PostgreSQL
- UPDATE
- ADD COLUMN
- ALTER COLUMN
- DROP COLUMN

Inspecting and modifying data

Modifying tables with ALTER TABLE

- Adding a column
    - ALTER TABLE table ADD COLUMN column data_type;
- Delete a column
    - ALTER TABLE table DROP COLUMN column;
- To change the data type of a column, we would use this code:
    - ALTER TABLE table ALTER COLUMN column SET DATA TYPE data_type;

Inspecting and modifying data

Modifying tables with ALTER TABLE

- Adding a NOT NULL constraint to a column will look like the following:
    - ALTER TABLE table ALTER COLUMN column SET NOT NULL;

Note that in PostgreSQL and some other systems, adding a constraint to the table causes all rows to be checked to see whether they comply with the constraint. If the table has millions of rows, this could take a while.

- Removing the NOT NULL constraint looks like this:
    - ALTER TABLE table ALTER COLUMN column DROP NOT NULL;

Inspecting and modifying data

Modifying values with UPDATE

- The UPDATE statement modifies the data in a column in all rows or in a subset of rows that meet a condition.

```
UPDATE table
SET column = value
```

- The new value to place in the column can be a string, number, the name of another column, or even a query or expression that generates a value.

- We can update values in multiple columns at a time by adding additional columns and source values, and separating each column and value statement with a comma:

```
UPDATE table
SET column_a = value,
SET column_b = value;
```

Inspecting and modifying data

Modifying values with UPDATE

- Restrict update to certain rows with WHERE

```
UPDATE table
SET column = value
WHERE criteria;
```

- Update one table with values from another table.
- Standard ANSI SQL requires that we use a **subquery** (we cover this in a seperate presentation), a query inside a query, to specify which values and rows to update:

```
UPDATE table
SET column = (SELECT column
            FROM table_b
            WHERE table.column = table_b.column)
WHERE EXISTS (SELECT column
            FROM table_b
            WHERE table.column = table_b.column);
```

Inspecting and modifying data

Modifying values with UPDATE

- Some database managers offer additional syntax for updating across tables.
- PostgreSQL supports the ANSI standard but also a simpler syntax using a FROM clause for updating values across tables:

```
UPDATE table
SET column = table_b.column
FROM table_b
WHERE table.column = table_b.column;
```

- When you execute an UPDATE statement, PostgreSQL returns a message stating UPDATE along with the number of rows affected.

Inspecting and modifying data

Creating backup tables

- Backing up a table (create an identical table):

```
CREATE TABLE meat_poultry_egg_inspect_backup
AS (SELECT *
   FROM meat_poultry_egg_inspect);
```

- Check number of records:

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

original	backup
6287	6287

Inspecting and modifying data

Creating backup tables

- Indexes are not copied when creating a table backup using the CREATE TABLE statement.
- If you decide to run queries on the backup, be sure to create a separate index on that table.

Inspecting and modifying data

Creating a column copy

- Creating and filling the st_copy column with ALTER TABLE and UPDATE

```
-- add a new column st_copy
ALTER TABLE meat_poultry_egg_inspect ADD COLUMN st_copy varchar(2);
```

```
-- fill the new column with st
UPDATE meat_poultry_egg_inspect
SET st_copy = st;
```

- Checking values in the st and st_copy columns

```
SELECT st,
       st_copy
FROM meat_poultry_egg_inspect
ORDER BY st;
```

st	st_copy
AK	AK
AK	AK
AK	AK
AK	AK
AK	AK
AK	AK

Inspecting and modifying data

Updating rows where values are missing

- Atlas Inspection is located in Minnesota; Hall-Namie Packing is in Alabama; and Jones Dairy is in Wisconsin:

```
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';
```

Inspecting and modifying data

Updating rows where values are missing

- If something goes wrong, we could restore the original st column values:

A) Restoring from the column backup

```
UPDATE meat_poultry_egg_inspect
SET st = st_copy;
```

B) Restoring from the table backup

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

Inspecting and modifying data

Updating values for consistency

- In our data, we have the following spelling variations:

```
Armour - Eckrich Meats, LLC
Armour-Eckrich Meats LLC
Armour-Eckrich Meats, Inc.
Armour-Eckrich Meats, LLC
```

- We use UPDATE to standardize the spelling
- However, we do not alter the original column but first create a new one, which we name `company_standard`

Inspecting and modifying data

Updating values for consistency

- Creating and filling the company_standard column:

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

```
UPDATE meat_poultry_egg_inspect
SET company_standard = company;
```

Inspecting and modifying data

Updating values for consistency

- Let's standardize any name with "Armour" to "Armour-Eckrich Meats"
- Use UPDATE to modify field values that match a string

```
UPDATE meat_poultry_egg_inspect
SET company_standard = 'Armour-Eckrich Meats'
WHERE company LIKE 'Armour%';
```

Inspecting and modifying data

Concatenation

- Now we come back to the issue with the column ZIP (missing zeros at the beginning)
- Creating and filling the zip_copy column:

```sql
ALTER TABLE meat_poultry_egg_inspect ADD COLUMN zip_copy varchar(5);
```

```sql
UPDATE meat_poultry_egg_inspect
SET zip_copy = zip;
```

Inspecting and modifying data

Concatenation

- Modify codes in the zip column missing two leading zeros for Puerto Rico (PR) and the Virgin Islands (VI):

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

- The double-pipe string operator (||) performs concatenation.

Inspecting and modifying data

Concatenation

- Modify codes in the zip column missing one leading zero

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

Inspecting and modifying data

Concatenation

- Using length() and count() to test the zip column

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

Inspecting and modifying data

Concatenation

- Before concatenation

length	length_count
3	86
4	496
5	5705

- After concatenation

length	length_count
5	6287

Inspecting and modifying data

Updating values across tables

- Let’s say we’re setting an inspection date for each of the companies in our table.
- We want to do this by U.S. regions, such as Northeast, Pacific, and so on, but those regional designations don’t exist in our table.
- However, they do exist in a data set we can add to our database that also contains matching st state codes.
- This means we can use that other data as part of our UPDATE statement to provide the necessary information.

Inspecting and modifying data

Updating values across tables

*Let's begin with the New England region to see how this works.

Creating and filling a state_regions table:

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

Inspecting and modifying data

Updating values across tables

- Add a column for inspection dates, and then fill in that column with the New England states.

```
COPY state_regions
FROM '/Users/jankirenz/Documents/HdM/Vorlesungen/DataScience/ProgrammingLanguages/SQL/sq
WITH (FORMAT CSV, HEADER, DELIMITER ',');
```

Inspecting and modifying data

Updating values across tables

- Adding and updating an inspection_date column

```
ALTER TABLE meat_poultry_egg_inspect ADD COLUMN inspection_date date;
```

```
UPDATE meat_poultry_egg_inspect AS 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');
```

Inspecting and modifying data

Updating values across tables

- Viewing updated inspection_date values

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

Show 8 entries Search:

	st	inspection_date
1	AK	
2	AL	
3	AR	
4	AS	
5	AZ	
6	CA	
7	CO	
8	CT	2019-12-01

Showing 1 to 8 of 20 entries Previous 1 2 3 Next

Inspecting and modifying data

Deleting data

- DELETE FROM: Deleting all rows from a table

```
DELETE FROM table_name;
```

- Alternatively, you can drop the entire table from the databse

```
DROP TABLE table_name;
```

- Delete matching cases:

```
DELETE FROM table_name
WHERE expression;
```

Inspecting and modifying data

Deleting data

- Delete rows matching an expression

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

Inspecting and modifying data

Deleting data

- DROP COLUMN: Delete columns
- Remove a column from a table using DROP

```
ALTER TABLE meat_poultry_egg_inspect DROP COLUMN zip_copy;
```

- Remove a table from a database using DROP

```
DROP TABLE meat_poultry_egg_inspect_backup;
```

Inspecting and modifying data

Transaction blocks

- The essential point of a transaction is that it bundles multiple steps into a single, all-or-nothing operation.
- The intermediate states between the steps are not visible to other concurrent transactions.
- If some failure occurs that prevents the transaction from completing, then none of the steps affect the database at all.

Source: PostgreSQL

Inspecting and modifying data

Transaction blocks

- START TRANSACTION signals the start of the transaction block.
- In PostgreSQL, you can also use the non-ANSI SQL BEGIN keyword.
- COMMIT signals the end of the block and saves all changes.
- ROLLBACK signals the end of the block and reverts all changes.

  > When you start a transaction, any changes you make to the data aren't visible to other database users until you execute COMMIT

Inspecting and modifying data

Transaction blocks

- We can apply this transaction block technique to review changes a query makes and then decide whether to keep or discard them.
- let's say we're cleaning dirty data related to the company AGRO Merchants Oakland LLC.

```
AGRO Merchants Oakland LLC
AGRO Merchants Oakland LLC
AGRO Merchants Oakland, LLC
```

- We want the name to be consistent, so we'll remove the comma from the third row using an UPDATE query, as we did earlier.
- But this time we'll check the result of our update before we make it final (and we'll purposely make a mistake we want to discard).

Transaction block demo

- Demonstrating a transaction block
- START TRANSACTION

```
START TRANSACTION;
```

- UPDATE TABLE (with error in spelling)

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

Transaction block demo

- Show result

```
-- view changes
SELECT company
FROM meat_poultry_egg_inspect
WHERE company LIKE 'AGRO%'
ORDER BY company;
```

Show 8 entries Search:

	company
1	AGRO Merchants Oakland LLC
2	AGRO Merchants Oakland LLC
3	AGRO Merchantss Oakland LLC

Showing 1 to 3 of 3 entries Previous 1 Next

Transaction block demo

- Revert changes with ROLLBACK

```
ROLLBACK;
```

- Show result

```
-- view changes
SELECT company
FROM meat_poultry_egg_inspect
WHERE company LIKE 'AGRO%'
ORDER BY company;
```

Show 8 entries Search:

	company
1	AGRO Merchants Oakland LLC
2	AGRO Merchants Oakland LLC
3	AGRO Merchants Oakland, LLC

Showing 1 to 3 of 3 entries Previous 1 Next

Thank you!

Prof. Dr. Jan Kirenz

HdM Stuttgart
Nobelstraße 10
70569 Stuttgart

HOCHSCHULE
DER MEDIEN