Exploratory Data Analysis with R

Complex queries

Xuemao Zhang
East Stroudsburg University

December 5, 2022

Outline

- Complex queries
  - Column aliases
  - Table aliases
  - Subquery
- Database import
- Northwind database
- ETL with R

Column aliases

- https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-column-alias/
- A **column alias** allows you to assign a column or an expression in the select list of a SELECT statement a **temporary name**. The column alias exists temporarily during the execution of the query.
- Syntax:
  - The `AS` keyword is optional so we can omit it.

```
SELECT column_name AS alias_name
FROM table_name;
```

or

```
SELECT column_name alias_name
FROM table_name;
```

- Set an alias for an expression in the `SELECT` clause:

```
SELECT expression AS alias_name
FROM table_name;
```

- Again, we consider the database `Database1` created in Lecture 27.

Column aliases

- Assigning a column alias to a column
- Rename `hindfoot_length` to `length`

```
SELECT
    weight,
    hindfoot_length AS length
FROM surveys;
```

Column aliases

- Assigning a column alias to an expression
- Concatenate the 3 columns `month`, `day` and `year` as date
  - To concatenate two or more strings into one, you use the string concatenation operator `||` or the `CONCAT` function

```
SELECT plot_id, species_id, sex, hindfoot_length, weight,
    month || '-' || day || '-' || year AS date
FROM
    surveys;
```

```
SELECT plot_id, species_id, sex, hindfoot_length, weight,
    CONCAT (month,'-', day, '-', year) AS date
FROM
    surveys;
```

Column aliases

- Column aliases that contain spaces
  - If a column alias contains one or more spaces, you need to surround it with **double quotes**.

```
SELECT plot_id, species_id, sex, hindfoot_length, weight,
    CONCAT (month,'-', year) AS "month_year"
FROM
    surveys;
```

Table aliases

- https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-alias/
- Table aliases temporarily assign tables new names during the execution of a query.
- Syntax
  - Again, `AS` keyword is optional.

```
table_name AS alias_name
```

- It can make queries more readable when a long table name becomes short which is useful when joining tables.

```
SELECT *
FROM species sp
INNER JOIN surveys su
ON sp.species_id = su.species_id
WHERE su.plot_id=2;
```

Subquery

- https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-subquery/
- A subquery is a query within another PostgreSQL query and embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Example: select all records with weight greater than the average weight without subquery.

```
SELECT  AVG(weight)
FROM    surveys;
-- It returns 42.67

SELECT *
FROM  surveys
WHERE weight > 42.67;
```

Subquery

- Example: select all records with weight greater than the average weight with a subquery.

```
SELECT *
FROM  surveys
WHERE weight > (
    SELECT  AVG(weight)
    FROM    surveys
    );
```

Subquery

- Subqueries with the SELECT Statement

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
      (SELECT column_name [, column_name ]
      FROM table1 [, table2 ]
      [WHERE])
```

Subquery

- PostgreSQL subquery with IN operator: A subquery can return zero or more rows. To use this subquery, you use the `IN` operator in the `WHERE` clause.

```
SELECT year, plot_id
FROM  surveys
WHERE year in (
    SELECT  year
    FROM    surveys
    where weight > 42.67
    );
```

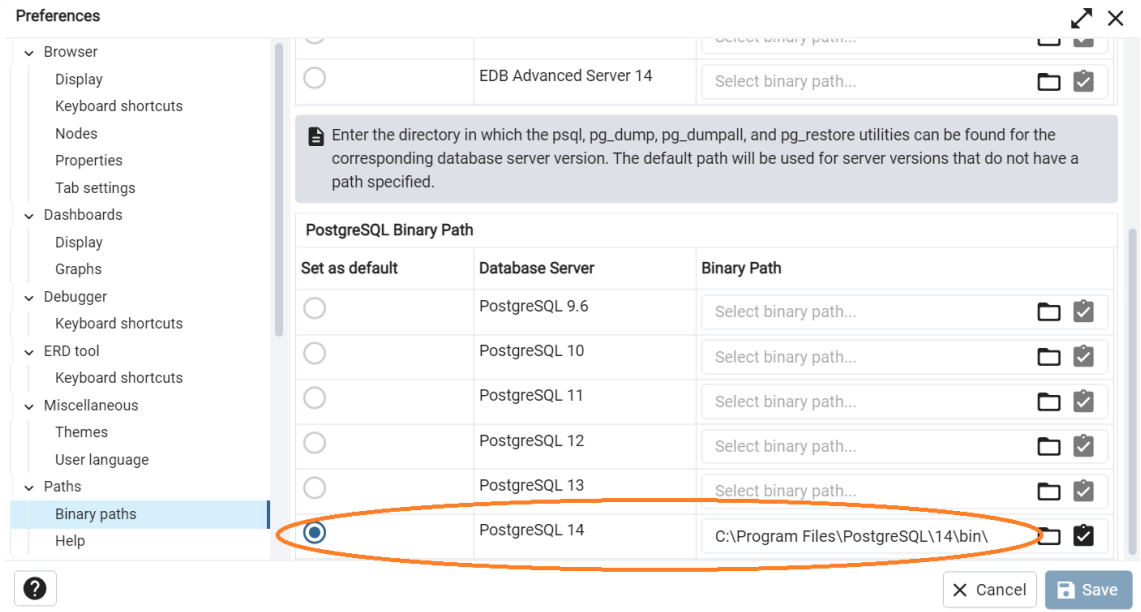
Database import

- If you have a database file `*.sql` or `*.tar`, you can import it to postgreSQL directly.

1. Creat a database in PgAdmin4
2. Open/select the database created above
3. In the top menu, Click Tools, Query Tool
4. Click the left most icon `Open File` in Query Tool and drill down to find the .sql file you want.
5. Click `Select`
6. Run all the code in the Query Editor.

Database import

- Common Error that you might face while importing/restoring any database for the first time is: **Utility file not found. Please correct the Binary Path in the Preferences dialog:**
- Quick Fix: go to file -> Preferences -> Path -> Binary Path and set Postgres Binary path as your desktop/laptop Postgres bin folder path

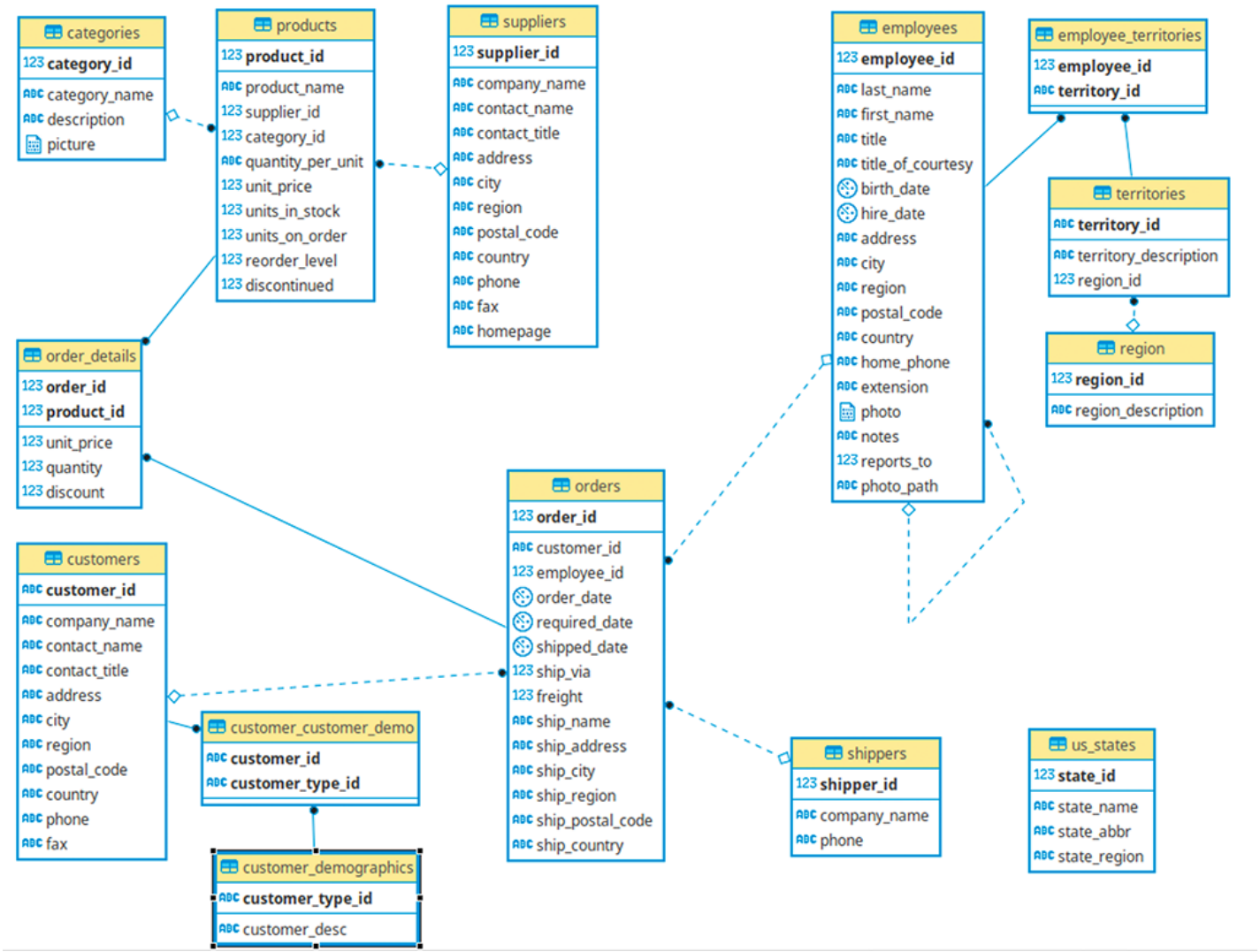


Northwind database

- Northwind database
  - Download https://github.com/pthom/northwind_psql
  - The Northwind database was first released in Microsoft Access Database as a sample database with data already pre-populated. The database contains sales data for a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.
- The Northwind dataset includes sample data for the following.
  - **Suppliers**: Suppliers and vendors of Northwind
  - **Customers**: Customers who buy products from Northwind
  - **Employees**: Employee details of Northwind traders
  - **Products**: Product information
  - **Shippers**: The details of the shippers who ship the products from the traders to the end-customers
  - **Orders** and **Order_Details**: Sales Order transactions taking place between the customers & the company

Northwind database

- Data structure
  https://github.com/pthom/northwind_psql/blob/master/ER.png



Northwind database

Let's practice SQL queries following http://www.geeksengine.com/

- Part I https://www.geeksengine.com/database/problem-solving/northwind-queries-part-1.php

Northwind database

- Part II https://www.geeksengine.com/database/problem-solving/northwind-queries-part-2.php

Northwind database

- Part III https://www.geeksengine.com/database/problem-solving/northwind-queries-part-3.php

ETL with R

- ETL (Extract, Transform and Load) is a process that extracts, transforms, and loads data from multiple sources to a data warehouse or other unified data repository (https://www.ibm.com/cloud/learn/etl).
- Extract: During data extraction, raw data is copied or exported from source locations to a staging area. Data management teams can extract data from a variety of data sources, which can be structured or unstructured. Those sources include but are not limited to:
  - SQL or NoSQL servers
  - CRM and ERP systems
  - Flat files
  - Email
  - Web pages

ETL with R

- Transform: In the staging area, the raw data undergoes data processing. Here, the data is transformed and consolidated for its intended analytical use case. This phase can involve the following tasks:
  - Filtering, cleansing, de-duplicating, validating, and authenticating the data.
  - Performing calculations, translations, or summarizations based on the raw data. This can include changing row and column headers for consistency, converting currencies or other units of measurement, editing text strings, and more.
  - Conducting audits to ensure data quality and compliance
  - Removing, encrypting, or protecting data governed by industry or governmental regulators
  - Formatting the data into tables or joined tables to match the schema of the target data warehouse.

ETL with R

- Load
  - In this last step, the transformed data is moved from the staging area into a target data warehouse. Typically, this involves an initial loading of all data, followed by periodic loading of incremental data changes and, less often, full refreshes to erase and replace data in the warehouse. For most organizations that use ETL, the process is automated, well-defined, continuous and batch-driven. Typically, ETL takes place during off-hours when traffic on the source systems and the data warehouse is at its lowest.
- R is not designed for ETL, but we do see R packages like `tidyverse`, `lubridate` and `stringr` powerful in data wrangling.
- There are a lot of other good ETL tools for large volumes of data. For example, Apache Spark provides the framework to up the **ETL** game. It supports multi-languages Scala, Java, SQL, Python and R, and can create simple and but robust ETL pipelines.
  - Apache Spark is a lightning-fast unified analytics engine for **big data** and **machine learning**.
  - A free e-book: Mastering Spark with R
- In this section, we just check how to connect a database with R.

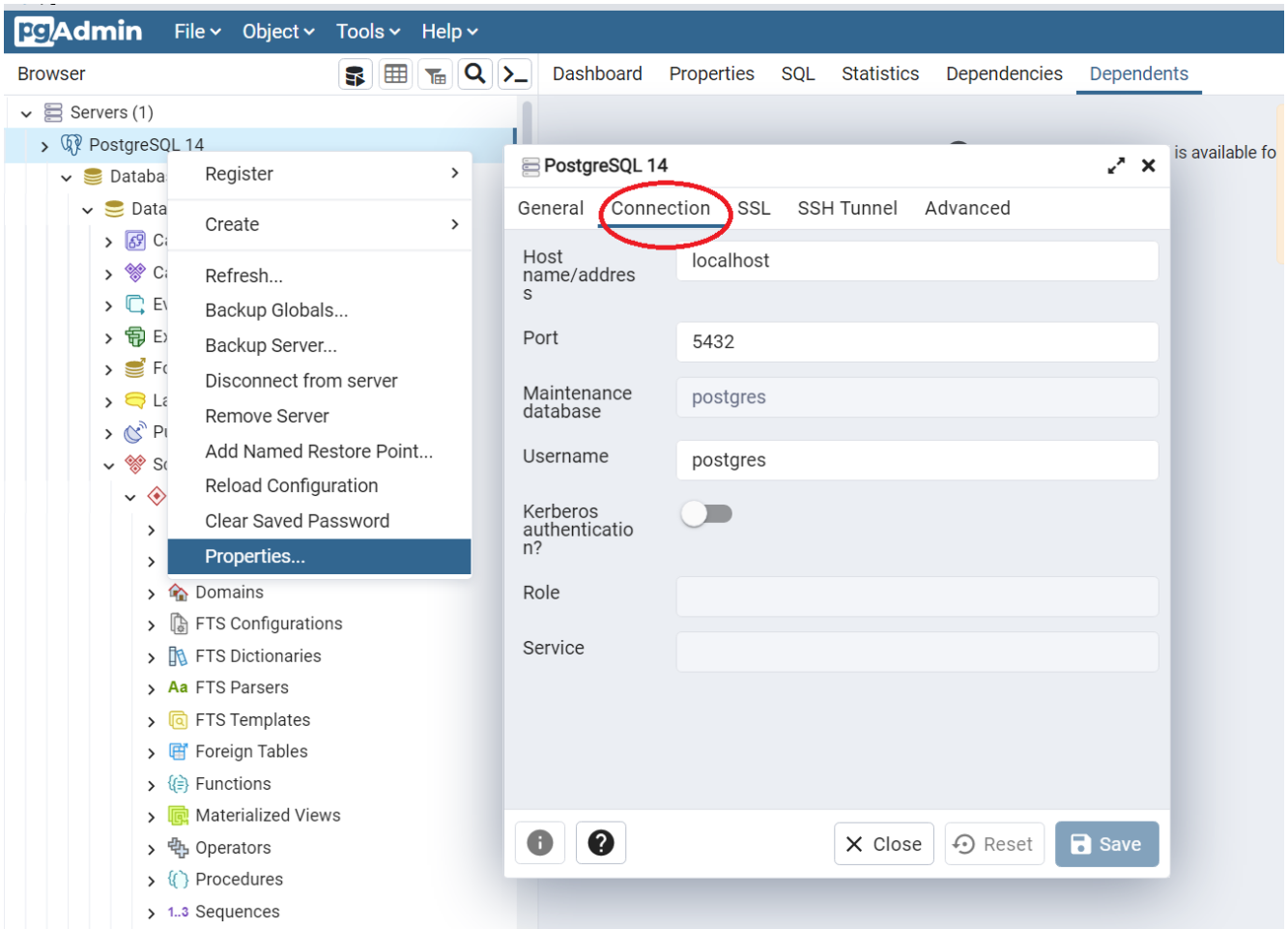
ETL with R

- Step one: install and load required R packages.

```
packs=c('RPostgres','DBI','RODBC','odbc','dbplyr')
if(!require(packs)) install.packages(packs)
library(RPostgres)
library(DBI)
library(RODBC)
library(odbc)
library(dbplyr)
```

ETL with R

- Step two: Define a database connection
  - Right-click on PostgreSQL14 and select **Properties** (a dialog box will open)
  - Select the **Connection** tab, and you will see all information needed to create the connection.



ETL with R

- Step 3: use the `dbCanConnect()` function from the `DBI` package to check whether a connection exists

```
con<-DBI::dbCanConnect(RPostgres::Postgres(),
                       dbname="Database1",
                       port=5432,
                       user="postgres",
                       password="1234")
con
```

- If the result of printing the connection object con is `TRUE`, we can connect to the database

ETL with R

- Step 4: Create the connection object

```
con_1=DBI::dbConnect(RPostgres::Postgres(),
                     dbname="Database1",
                     port=5432,
                     user="postgres",
                     password="1234")
con_1
```

- If the previous step is good, now you will see

```
<PqConnection> Database1@localhost:5432
```

- Now list all views and tables in the database

```
DBI::dbListTables(con_1)
```

ETL with R

- Step 5: Finally we can query the data base. For example

```
surveys_head=dbGetQuery(con_1, 'select * from surveys limit 50')
class(surveys_head)
surveys_head
```

ETL with R

- After you are done with data wrangling, you can send your clean data to the data base using function `DBI::dbWriteTable()`.

```
surveys=dbGetQuery(con_1, 'select * from surveys')
surveys_clean=surveys %>%
filter(!is.na(hindfoot_length)) %>%
filter(!is.na(weigth))
DBI::dbWriteTable(con_1,name='surveys1',value=surveys_clean)
dbReadTable(con_1, "surveys1")
dbDisconnect(con_1) ##close the connection
```

License