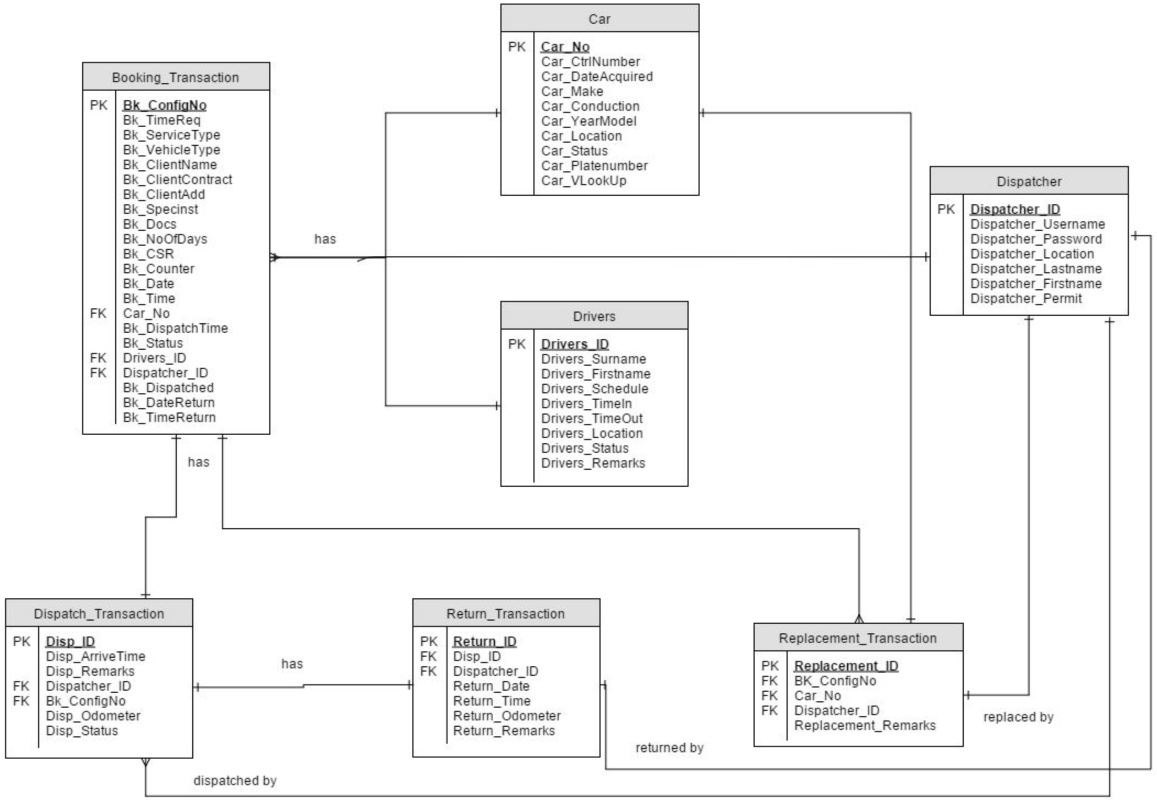
# Database Design

# Introduction

Understanding how data is organized within a database is important when interacting with that database, either as the one creating the database or as a person extracting information from it.

Questions to ask:

- What data is stored?
- What are the relationships between tables?
- What level of normalization should be used?



| | Booking_Transaction |
|---|---|
| PK | **Bk_ConfigNo** |
| | Bk_TimeReq |
| | Bk_ServiceType |
| | Bk_VehicleType |
| | Bk_ClientName |
| | Bk_ClientContract |
| | Bk_ClientAdd |
| | Bk_Specinst |
| | Bk_Docs |
| | Bk_NoOfDays |
| | Bk_CSR |
| | Bk_Counter |
| | Bk_Date |
| | Bk_Time |
| FK | Car_No |
| | Bk_DispatchTime |
| | Bk_Status |
| FK | Drivers_ID |
| FK | Dispatcher_ID |
| | Bk_Dispatched |
| | Bk_DateReturn |
| | Bk_TimeReturn |

| | Car |
|---|---|
| PK | **Car_No** |
| | Car_CtrlNumber |
| | Car_DateAcquired |
| | Car_Make |
| | Car_Conduction |
| | Car_YearModel |
| | Car_Location |
| | Car_Status |
| | Car_Platenumber |
| | Car_VLookUp |

| | Dispatcher |
|---|---|
| PK | **Dispatcher_ID** |
| | Dispatcher_Username |
| | Dispatcher_Password |
| | Dispatcher_Location |
| | Dispatcher_Lastname |
| | Dispatcher_Firstname |
| | Dispatcher_Permit |

| | Drivers |
|---|---|
| PK | **Drivers_ID** |
| | Drivers_Surname |
| | Drivers_Firstname |
| | Drivers_Schedule |
| | Drivers_TimeIn |
| | Drivers_TimeOut |
| | Drivers_Location |
| | Drivers_Status |
| | Drivers_Remarks |

| | Dispatch_Transaction |
|---|---|
| PK | **Disp_ID** |
| | Disp_ArriveTime |
| | Disp_Remarks |
| FK | Dispatcher_ID |
| FK | Bk_ConfigNo |
| | Disp_Odometer |
| | Disp_Status |

| | Return_Transaction |
|---|---|
| PK | **Return_ID** |
| FK | Disp_ID |
| FK | Dispatcher_ID |
| | Return_Date |
| | Return_Time |
| | Return_Odometer |
| | Return_Remarks |

| | Replacement_Transaction |
|---|---|
| PK | **Replacement_ID** |
| FK | BK_ConfigNo |
| FK | Car_No |
| FK | Dispatcher_ID |
| | Replacement_Remarks |

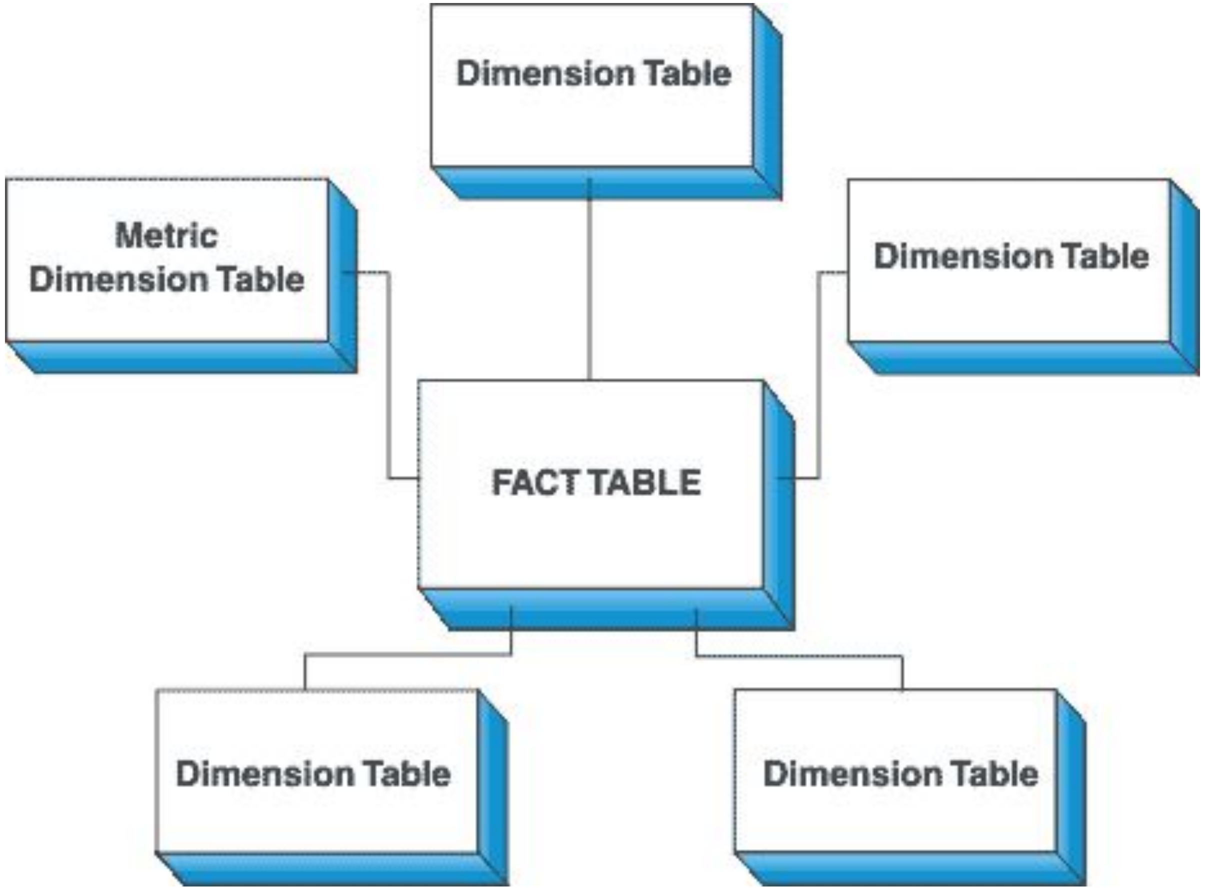
# Star Schema

Aka dimensional model

A database model optimized for queries and data warehouse tools.

Goal: query performance and schema simplicity

Design: “fact table” surrounded by some number of “dimension tables”



# Star Schema

**Facts:**

- Usually numeric measurements of a business
- Examples: Sales dollars, sales units, costs
- Often one fact per business transaction
- Change regularly

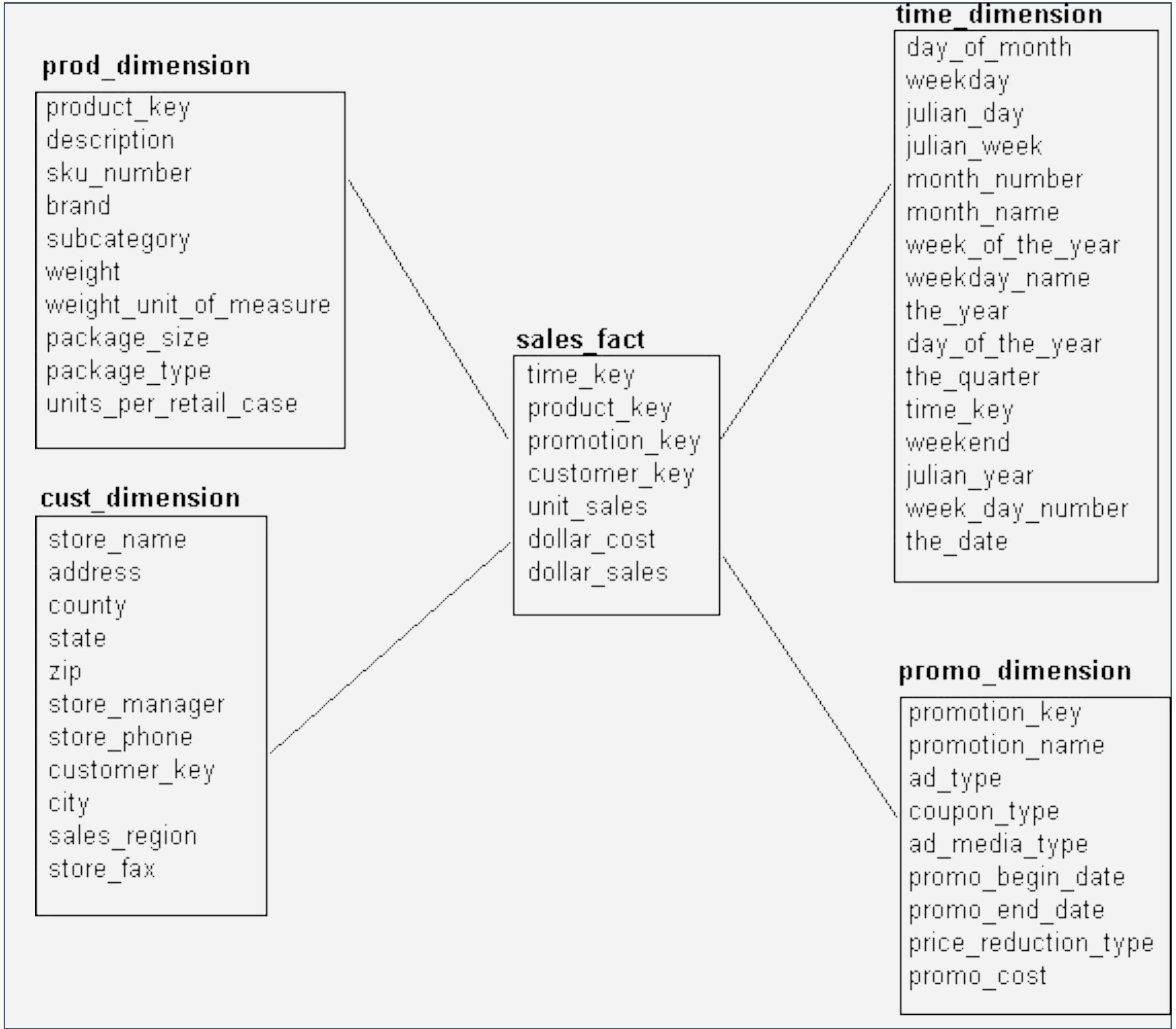
**Dimensions:**

- Descriptive data providing context for facts
- Examples: Product description, customer contact information, time dimensions
- Rarely if ever change

# Star Schema

Fact table connected to dimensional tables through “keys”

The “primary key” of each dimension table is linked to a “foreign key” of fact table



prod_dimension

product_key
description
sku_number
brand
subcategory
weight
weight_unit_of_measure
package_size
package_type
units_per_retail_case

cust_dimension

store_name
address
county
state
zip
store_manager
store_phone
customer_key
city
sales_region
store_fax

sales_fact

time_key
product_key
promotion_key
customer_key
unit_sales
dollar_cost
dollar_sales

time_dimension

day_of_month
weekday
julian_day
julian_week
month_number
month_name
week_of_the_year
weekday_name
the_year
day_of_the_year
the_quarter
time_key
weekend
julian_year
week_day_number
the_date

promo_dimension

promotion_key
promotion_name
ad_type
coupon_type
ad_media_type
promo_begin_date
promo_end_date
price_reduction_type
promo_cost

# Normalization

**Goals:** reduce data redundancy and improve data integrity

If a customer address changes, you should only have to update it in one table.

Allows for extending the database structure with minimal impacts to the existing structure.

The properties are encoded in the 6 database “normal forms”.

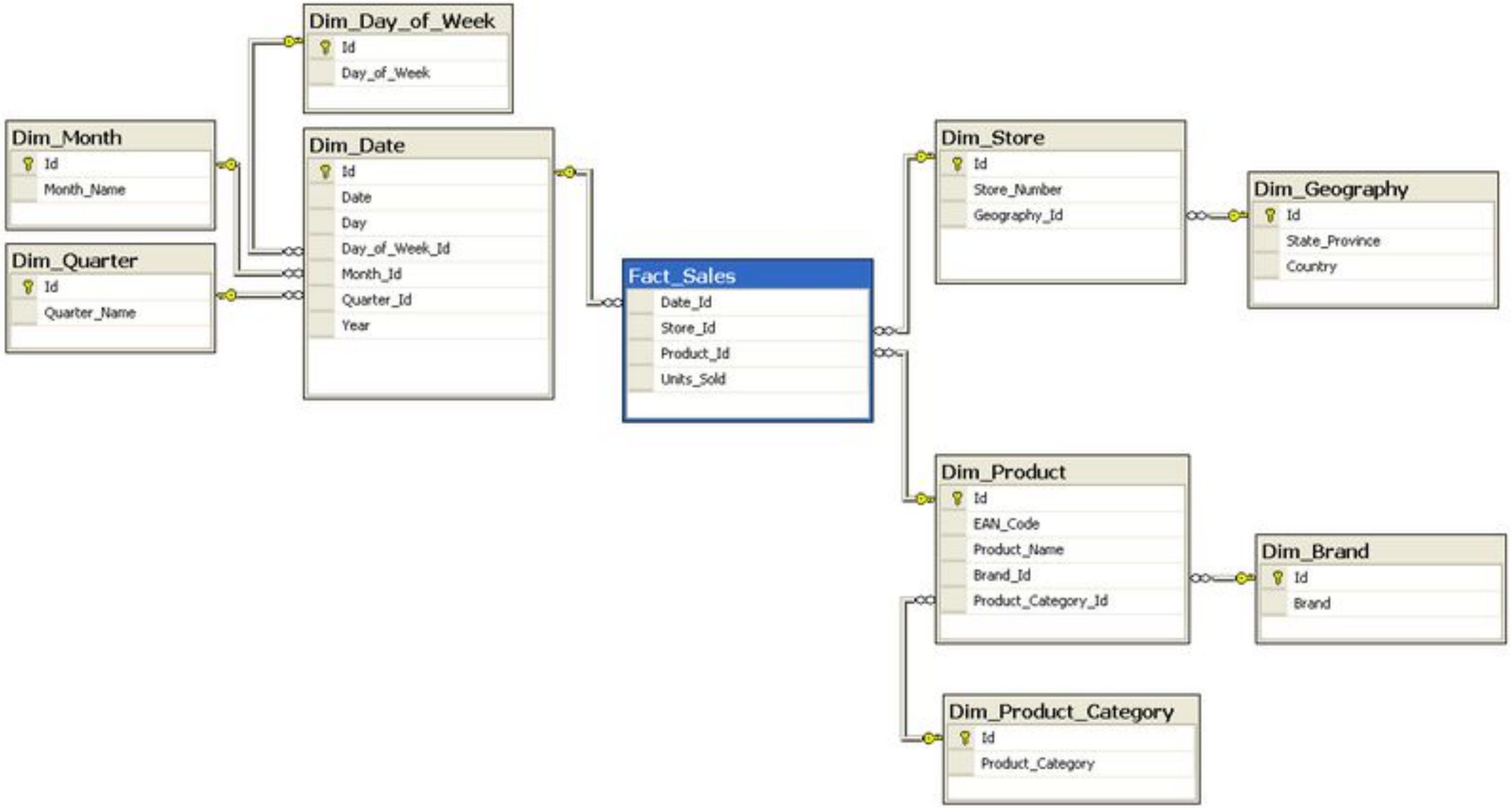
There are tradeoffs - more normalization leads to less redundancy, but more tables and more complicated queries.

To learn more about datbase normalization and the normal forms, see https://en.wikipedia.org/wiki/Database_normalization

# Snowflake Schema

"Normalized" version of a star schema

Central fact table surrounded by multiple layers of dimension tables



# Snowflake Schema

**Advantages:**

- Reduced redundancy and increased data integrity
- Less storage space required
- Fewer updates needed when data changes

**Disadvantages:**

- More complicated
- Higher query complexity - more joins needed

# About the Database

The ecd database contains information related to the The Tennessee Department of Economic and Community Development (TNECD) FastTrack grants. These grants are divided into three types:

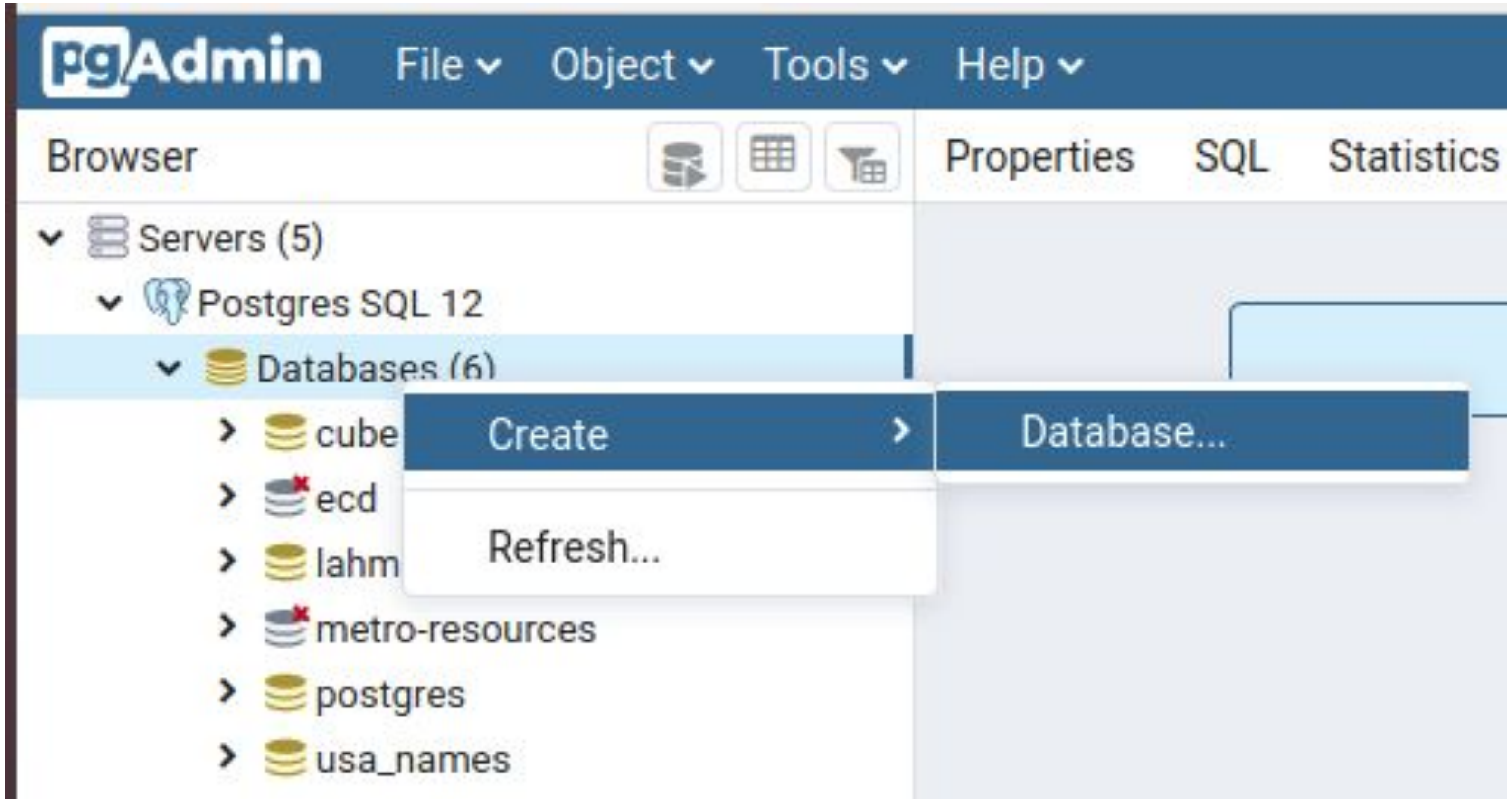
- FastTrack Job Training Assistance Program (FJTAP)
- FastTrack Infrastructure Development Program (FIDP)
- FastTrack Economic Development (ED) -- to offset expenses related to expansion and relocation

The data contained in the *ecd* table was scraped from https://www.tn.gov/transparenttn/open-ecd/openecd/fasttrack-project-database.html

The database also contains two other tables *population* and *unemployment* which contain county-level information on those two metrics.

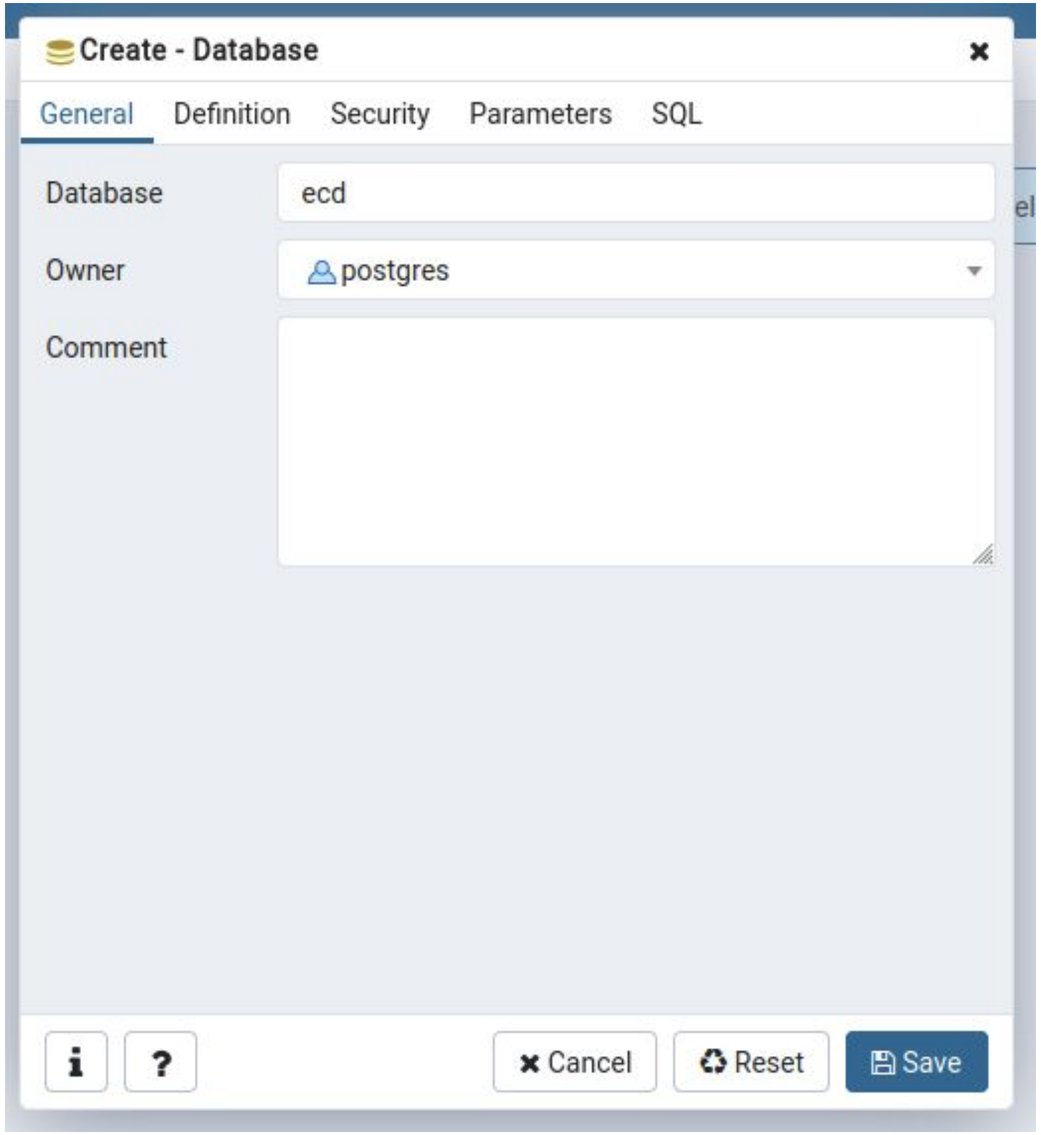
# Restoring from a Backup

You have received this database as a backup (.tar) file. We will need to restore it in order to work with it. We will do this through pgAdmin.



Start by right-clicking on "Databases". Then choose Create > Database...

# Restoring from a Backup



Create - Database

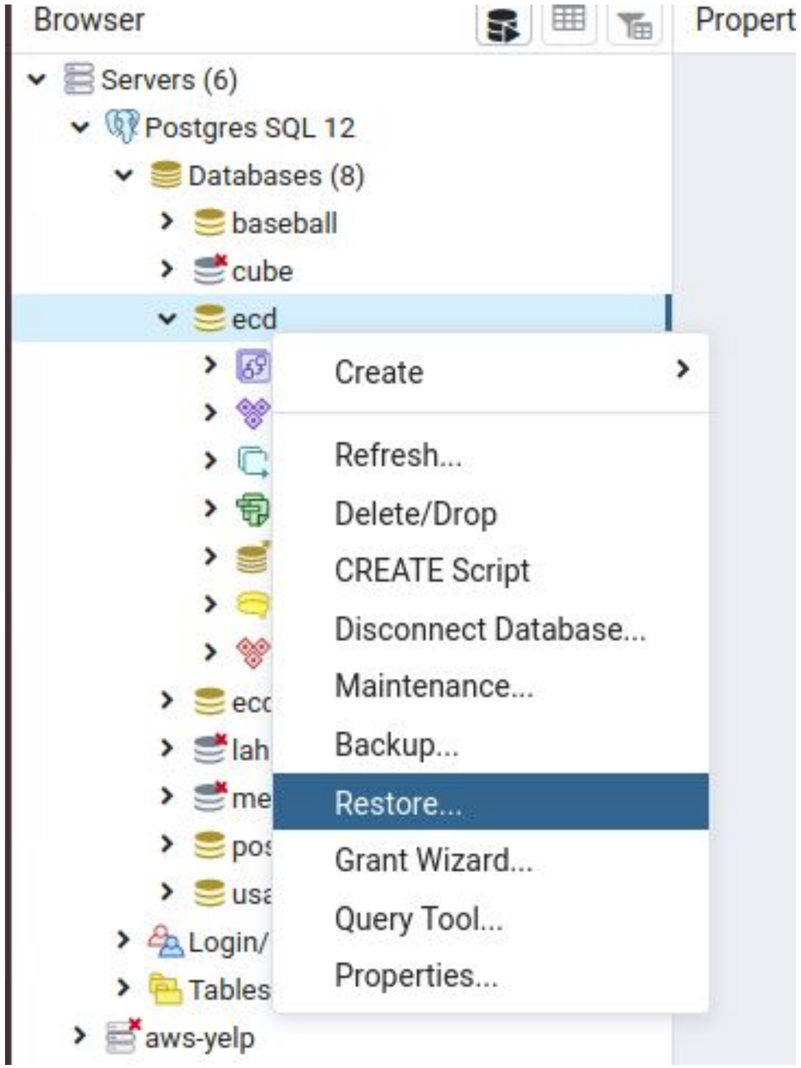
Database: ecd

Owner: postgres

Comment:

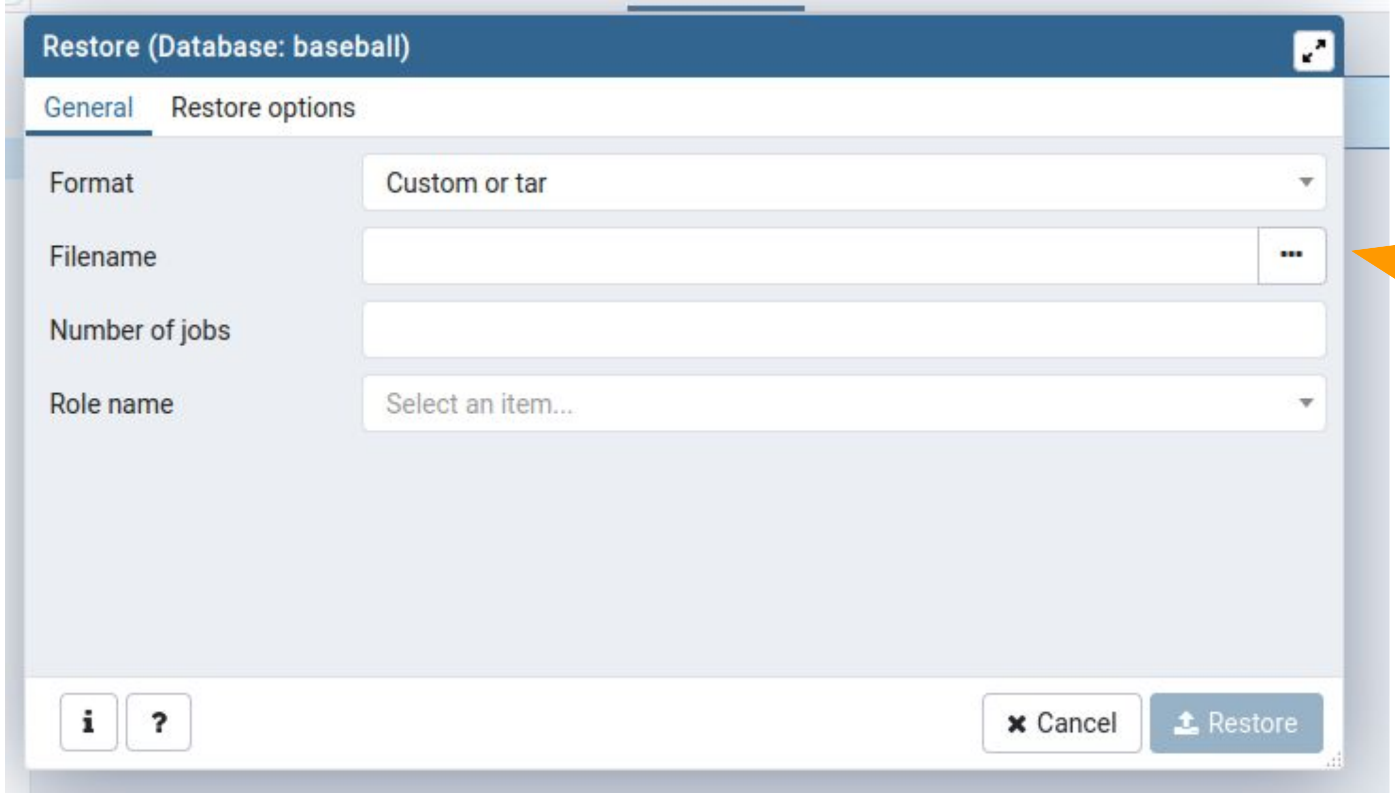
Name the database “ecd” and then click Save

# Restoring from a Backup



Right-click on the newly-created database and choose Restore...

# Restoring from a Backup



Restore (Database: baseball)

General | Restore options

Format: Custom or tar

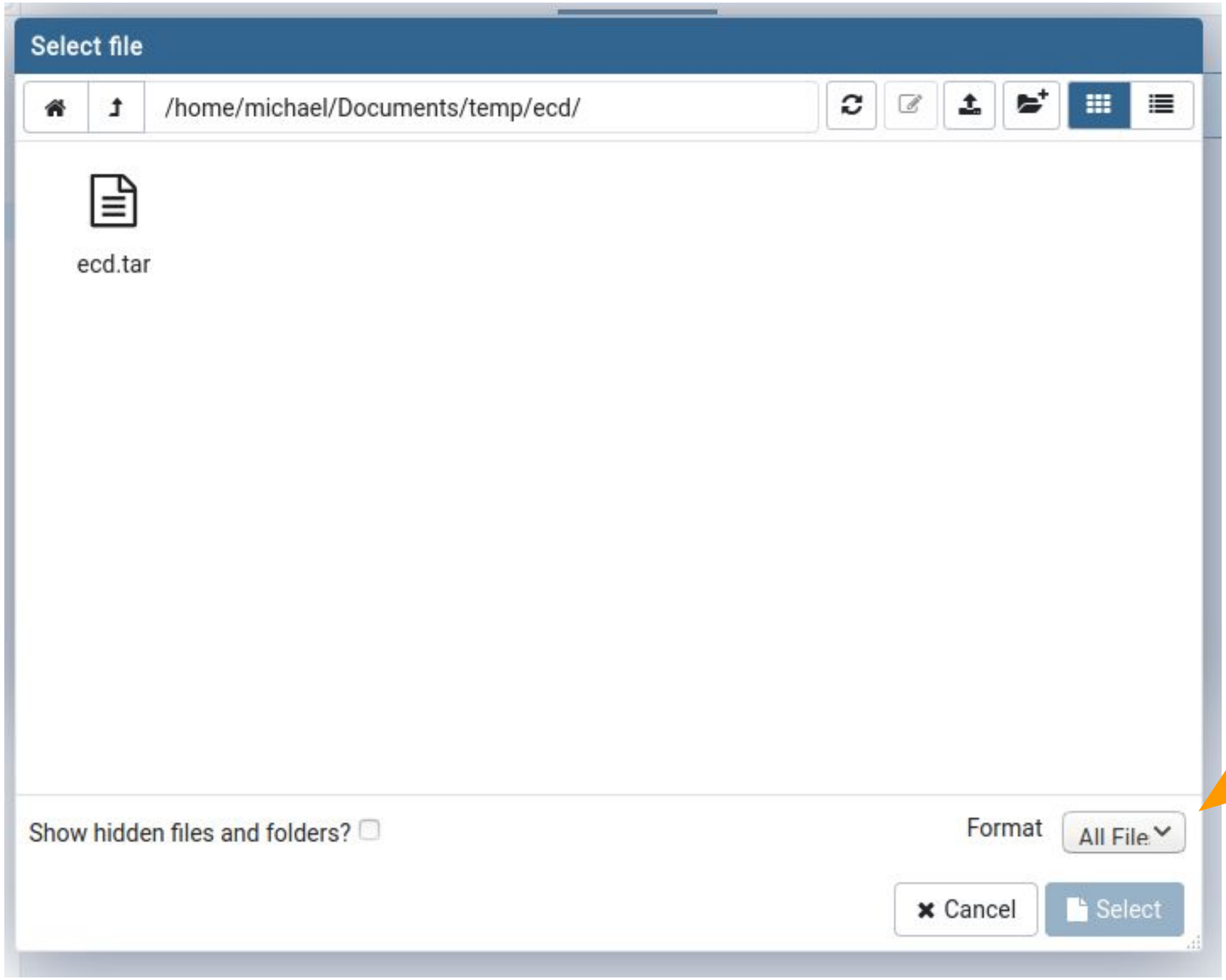
Filename:

Number of jobs:

Role name: Select an item...

In the next window, click the three dots next to Filename

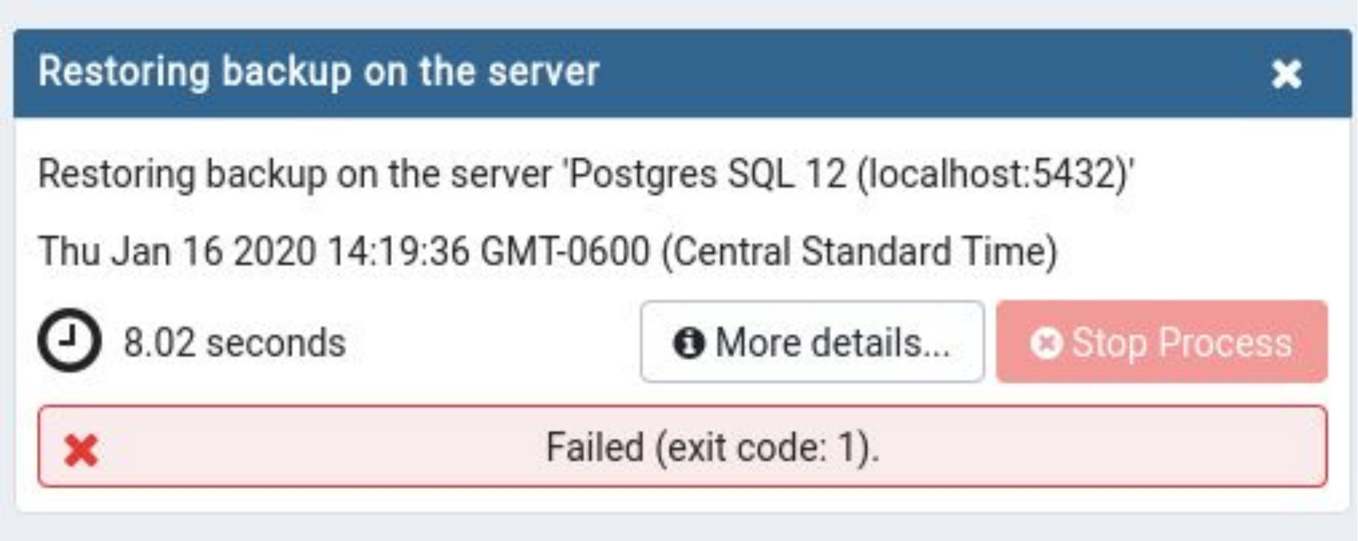
# Restoring from a Backup



Select file
/home/michael/Documents/temp/ecd/
ecd.tar
Show hidden files and folders? □
Format
All File
✖ Cancel
Select

Navigate to the folder containing the lahman.tar file. **You will have to change Format to "All Files"**

# Restoring from a Backup



Restoring backup on the server 'Postgres SQL 12 (localhost:5432)'

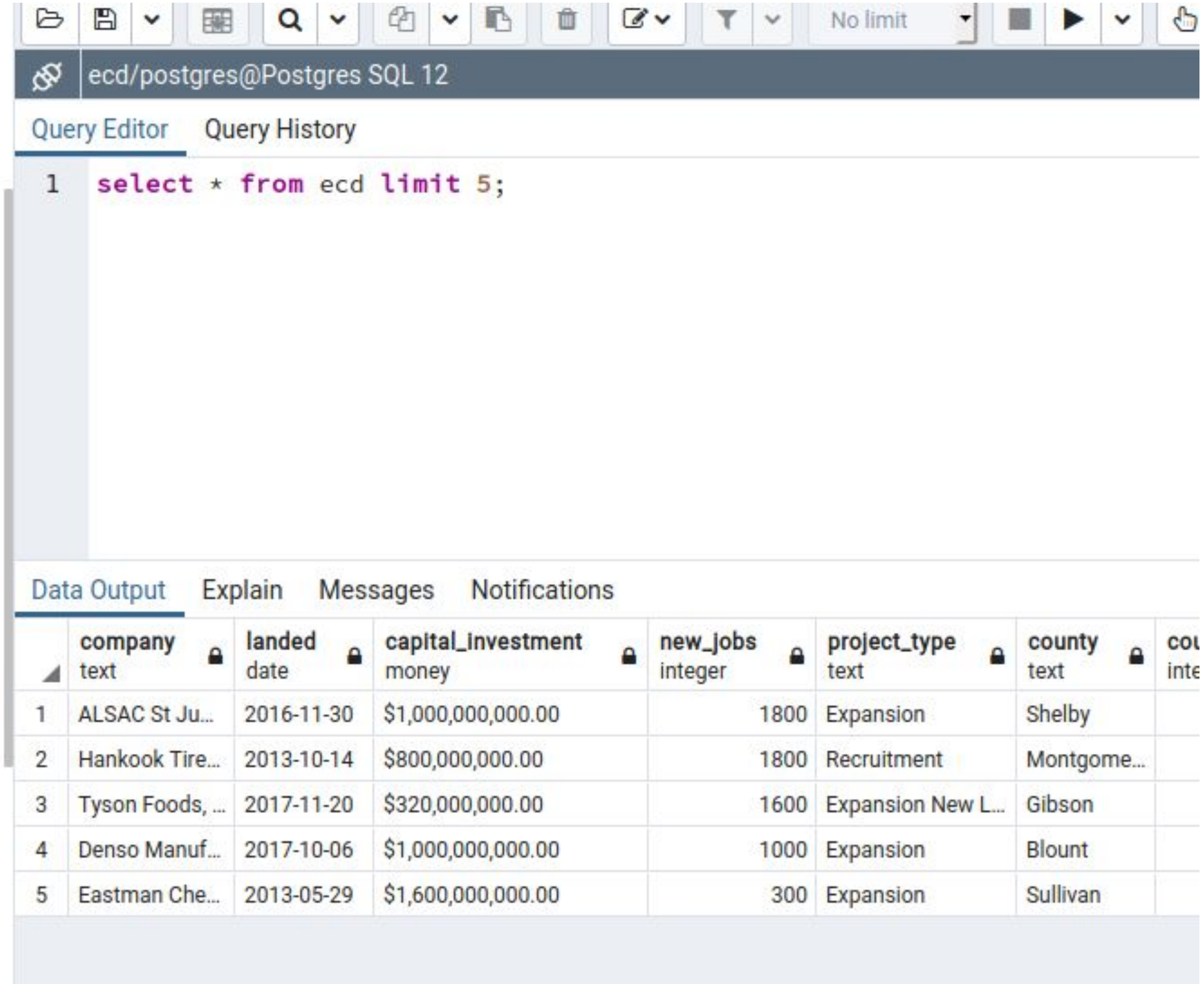
Thu Jan 16 2020 14:19:36 GMT-0600 (Central Standard Time)

8.02 seconds

✖ Failed (exit code: 1).

Select the file and then click Restore. You will likely get an error code that says the restore failed. Ignore this.

# Restoring from a Backup



```
select * from ecd limit 5;
```

|   | company | landed | capital_investment | new_jobs | project_type | county | cou |
|---|---|---|---|---|---|---|---|
|   | text | date | money | integer | text | text | inte |
| 1 | ALSAC St Ju... | 2016-11-30 | $1,000,000,000.00 | 1800 | Expansion | Shelby |   |
| 2 | Hankook Tire... | 2013-10-14 | $800,000,000.00 | 1800 | Recruitment | Montgome... |   |
| 3 | Tyson Foods, ... | 2017-11-20 | $320,000,000.00 | 1600 | Expansion New L... | Gibson |   |
| 4 | Denso Manuf... | 2017-10-06 | $1,000,000,000.00 | 1000 | Expansion | Blount |   |
| 5 | Eastman Che... | 2013-05-29 | $1,600,000,000.00 | 300 | Expansion | Sullivan |   |

To check that the restore was successful, click on the Public Schema for the ecd database, launch the query tool run the query “SELECT * FROM ecd LIMIT 5;” Your result set should be the same at the one shown.