



Bases relacionales usando PostgreSQL

Pilar Fernandez

Paul G. Allen School for Global Health
Washington State University

Email | pilar.fernandez@wsu.edu

Ferdinando Urbano · Francesca Cagnacci
Editors

Spatial Database for GPS Wildlife Tracking Data

A Practical Guide to Creating a Data
Management System with PostgreSQL/
PostGIS and R

Foreword by Ran Nathan



Dia 1

- Construir y visualizar una base de datos relacional

Capítulos 2/3/4

Dia 2

- Convertir base de datos a una base especial

Capítulo 5/6

- De el manejo de datos al análisis de datos en R

Capítulo 10

Algunos recursos para aprender SQL

- <https://www.coursera.org/learn/sql-for-data-science#syllabus>
- <https://www.datacamp.com/courses/introduction-to-sql>



Instalar

- PostgreSQL:
<https://www.postgresql.org/download/>
- Dbeaver:
<https://dbeaver.io/download/>
- QGIS:
<https://www.qgis.org/en/site/>



Setup

Al configurar PostgreSQL en este ejemplo, solo un host local.

Esto se puede cambiar y hacer que la base de datos acceda de forma remota, pero debe configurar un puerto que esté abierto para conexiones externas.

Esto es un poco más avanzado, pero algo a tener en cuenta.

Siguiendo el código primero creamos un superusuario. Luego creamos a otro usuario básico.

Trabajando en PostgreSQL

Se puede trabajar en la interfaz gráfica SQL (primero asegurarse de hacer clic en la base de datos en la que se quiere trabajar)



o mediante la línea de comandos PSQL



Para hacer esto, primero debe decirle a pgadmin dónde se encuentra postgres en sus archivos.

¿Cómo están organizadas las bases de datos en PostgreSQL?

1. Databases

a. Schemas: Son como carpetas que organizan la información.

i. Tables

1. Constraints

2. Indexes

3. Rules

4. Triggers

5. etc

ii. Operators

iii. Functions

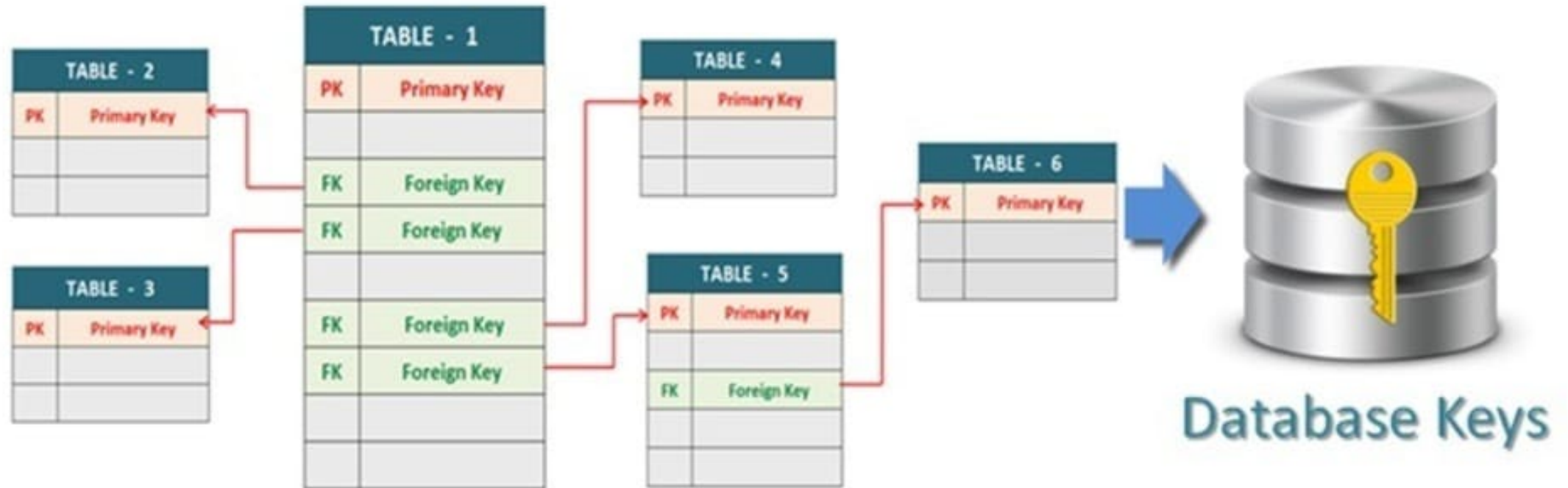
iv. etc

Diferentes **databases** pueden compartir muchos usuarios y grupos de usuarios, con diferentes niveles de permisos, pero no pueden compartir datos.

Los **schemas** dentro de las bases de datos pueden compartir datos y relaciones entre esquemas. Los schemas se utilizan para organizar objetos en grupos lógicos o, por ejemplo, para dar cierto tipo de acceso a algunos usuarios a un esquema determinado pero no a otros.

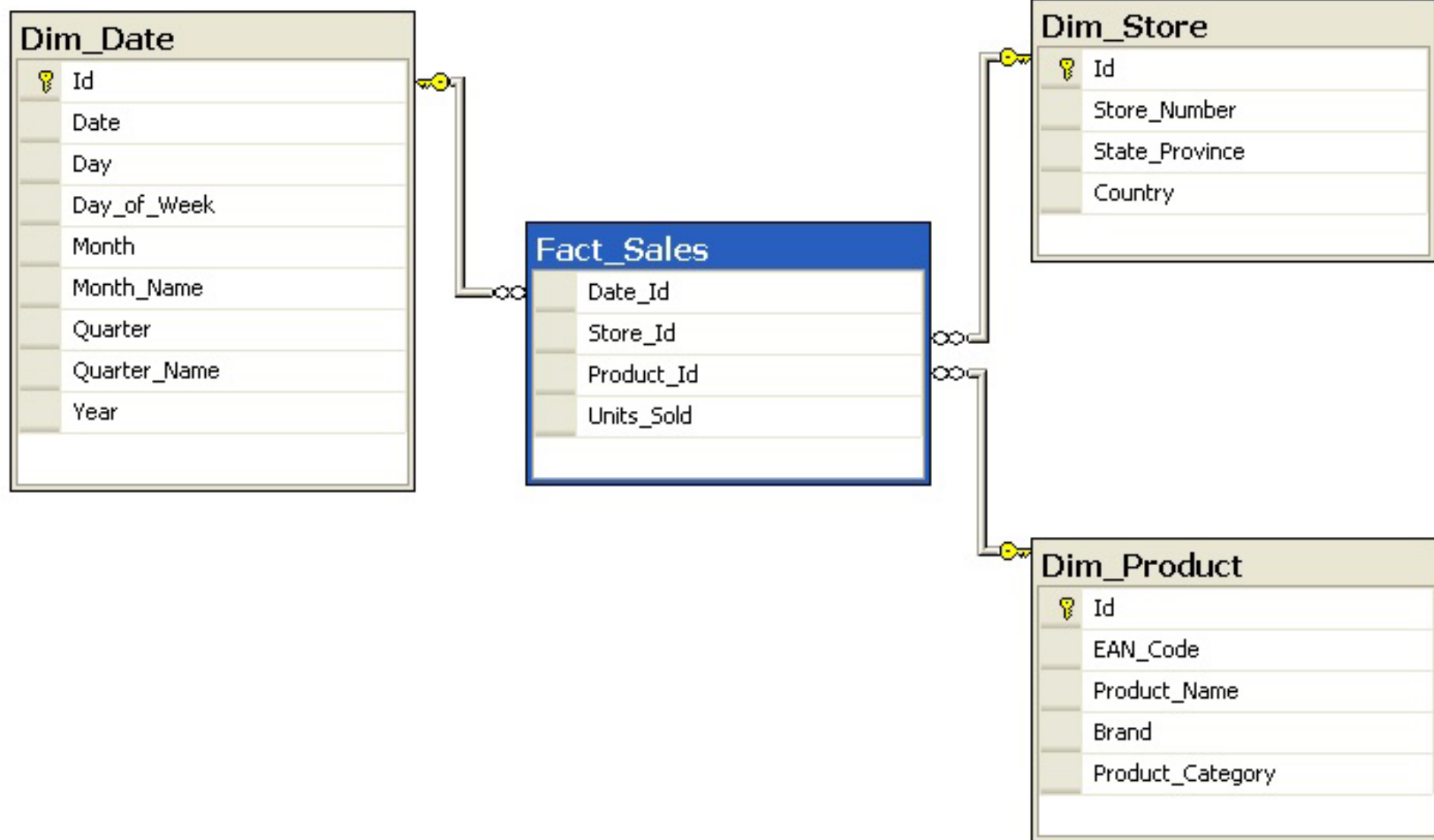
Las **tables** contienen los datos, las restricciones y las reglas de la base de datos

Keys: primary and foreign



Primary key: Es la clave unica para separar un registro de la tabla de otros registros.

Foreign key: Esta es la clave que corresponde al campo de primary key en la tabla upstream.



Arranquemos...

```
CREATE DATABASE new_db  
ENCODING = 'UTF8'  
TEMPLATE = template0  
LC_COLLATE = 'C'  
LC_CTYPE = 'C';
```

```
CREATE SCHEMA main;
```

```
CREATE TABLE main.new_table( variable type of variable...);
```

- Al hacer referencia a esta tabla, debe usar schema.table_name que en este caso es main.new_table
- Si tenes tus datos como .dbf se puede importarlos directamente, pero siempre es una buena idea crear la tabla antes de importar los datos y especificar las características.
- Acá podemos agregar una nueva variable que se utilizará para configurar la primary key y podría estar ausente en la base de datos original (a veces, dependiendo del origen de los datos, podría incluirse). La nueva variable agregada es gps_data_id y se configura como un tipo "serial" (será administrada por la base de datos y será única para cada fila).

Modificando las características de la tabla

Siempre se empieza con ALTER TABLE main.new_table

- Agregar primary key

```
ALTER TABLE main.new_table  
ADD CONSTRAINT new_table_pkey  
PRIMARY KEY(id_variable);
```

En el ejemplo: id_variable is gps_data_id

- Agregar constraint (restricción) para valores únicos:

```
ALTER TABLE main.new_table  
ADD CONSTRAINT unique_new_table_record  
UNIQUE(variable_1, variable_2,...);
```

Add new column(variable):

```
ALTER TABLE main.new_table  
ADD COLUMN column_name type  
DEFAULT default_value/function;
```

En el ejemplo, agregamos una nueva columna para realizar un seguimiento de los nuevos registros

```
ALTER TABLE main.gps_data  
ADD COLUMN insert_timestamp timestamp with time zone  
DEFAULT now();
```

La función now() agrega una marca de tiempo que indica la fecha y hora en que se agregó ese registro.

Ahora si, a importar los datos

Hay muchas maneras de importar una base de datos a Postgre:

La principal manera es usando COPY FROM

Para importar datos es importante dónde se encuentran:

<https://www.neilwithdata.com/copy-permission-denied>

La forma más fácil es poner todos mis archivos en la carpeta pública

```
COPIAR main.new_table(variable 1, variable 2,...)
FROM 'C:\Public\data\data.csv'
CON (FORMATO csv, ENCABEZADO, DELIMITADOR ';');
```

También (mejor) se puede usar la línea de comandos psql o la función de importación/exportación:
PSQL: \COPY <nombre de tabla> DESDE 'UBICACIÓN + file_name' DELIMITADOR ',' ENCABEZADO CSV;

El delimitador depende de su archivo, ya que los formatos utilizados en los EE. UU. es ',' pero, por ejemplo, en el libro es ';' porque se basa en la codificación utilizada para los idiomas basados en latín (solo para tener en cuenta si está utilizando bases de datos de otros países)

Creando índices

Son como tabla de contenido.

Los índices para los que cree índices tienen que ver con el tipo de consulta que realizará con más frecuencia. Uso de btree: el tipo de índice B-tree, implementado como método de acceso "btree", es adecuado para los datos que se pueden ordenar.

En el libro, crean dos índices en "main.gps_data" para mejorar el rendimiento en los quearies, utilizando acquisition_time y gps_sensor_code

#index 1

```
CREATE INDEX acquisition_time_index  
ON main.gps_data  
USING btree (acquisition_time );
```

#index 2

```
CREATE INDEX gps_sensors_code_index  
ON main.gps_data  
USING btree (gps_sensors_code);
```

Asignar permisos

Por lo general, tiene un solo administrador que puede cambiar la base de datos y un conjunto de usuarios que solo pueden leer los datos.

Acá creamos un nuevo usuario que tiene acceso para leer los elementos actuales y futuros del esquema principal.

```
CREAR ROL basic_user INICIAR SESIÓN  
CONTRASEÑA 'basic_user'  
NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;  
CONCEDER SELECCIÓN EN TODAS LAS TABLAS  
EN ESQUEMA principal  
A basic_user;  
MODIFICAR PRIVILEGIOS PREDETERMINADOS  
EN ESQUEMA principal  
CONCEDER SELECCIÓN EN TABLAS  
A basic_user;
```

Más información sobre usuarios y permisos: <https://www.postgresql.org/docs/9.2/user-manag.html>

Exportar tablas

Igual que COPIAR DESDE, pero ahora cambia a COPIAR A (dependiendo de dónde desee guardarlo, puede usar psql o la GUI de SQL

```
COPY (SELECT variables FROM main.gps_data)  
TO 'file location+name of file' WITH (FORMAT csv, HEADER, DELIMITER ',');
```

Quearies (Consultas)

¿Cómo ejecutar una consulta simple?

Ejemplo: Visualizar los primeros diez registros relacionados con el sensor "GSM015112" (ordenados por tiempo de adquisición)

```
SELECT gps_data_id AS id, gps_sensors_code AS sensor_id, latitude, longitude, acquisition_time  
FROM main.gps_data  
WHERE gps_sensors_code = 'GSM01512' and EXTRACT(MONTH FROM acquisition_time) = 5 ORDER BY  
acquisition_time  
LIMIT 10;
```

```
SELECT variables  
FROM table  
WHERE condition  
LIMIT 10
```

Si queremos cambiar el nombre de una variable en la consulta usamos AS:
Original_variable_name AS new_variable name

Construcción de una base de datos relacional

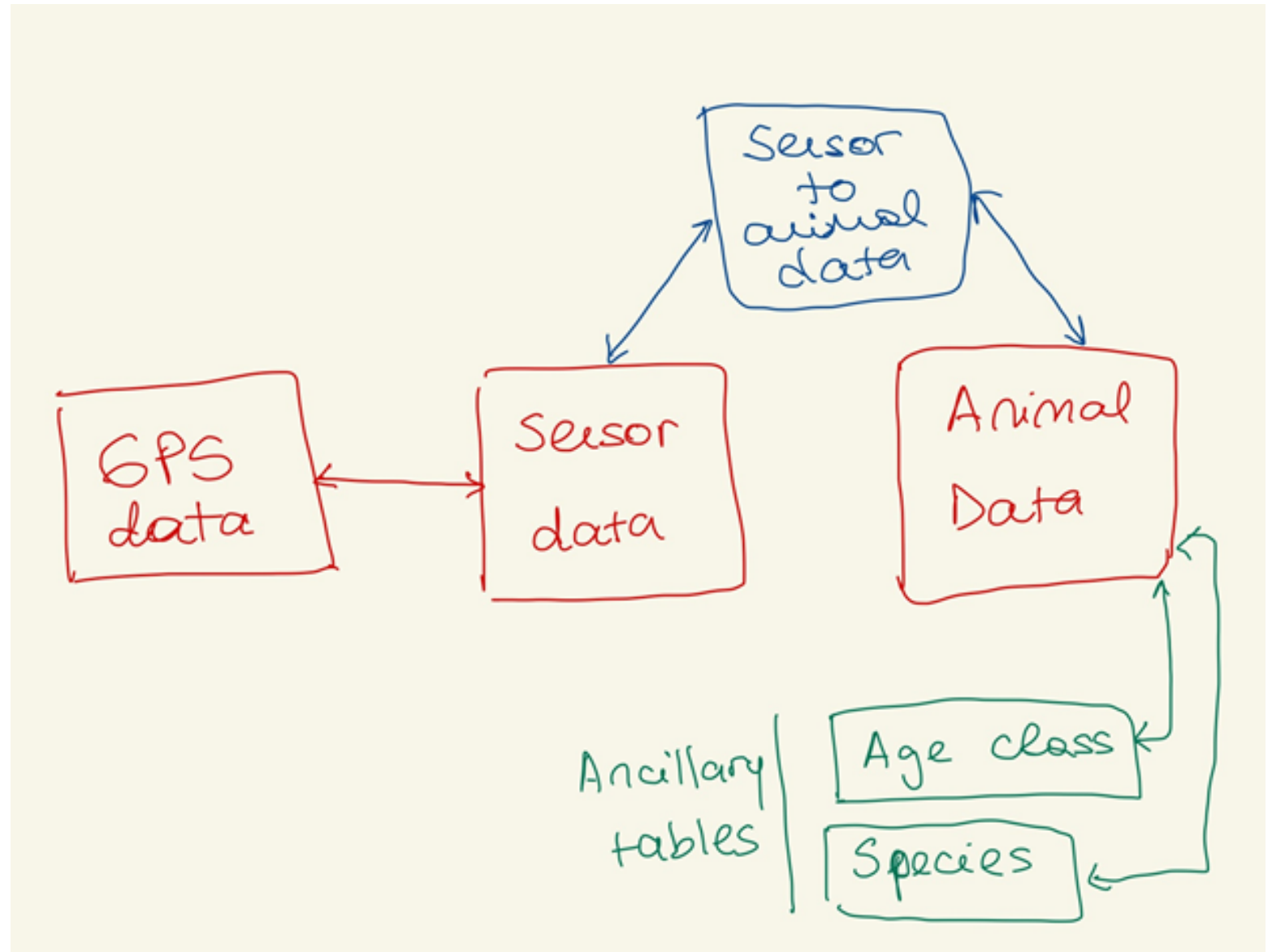
Capítulo 3

Acá agregaremos dos tablas de datos:

- Sensor
- Animal

Y cree dos tablas auxiliares que proporcionen soporte a las tablas principales:

- Age classes
- Species



En este ejemplo

- Los individuos son las unidades de muestreo: cada individuo tendrá rasgos demográficos y datos GPS vinculados a ellos.
- La tabla gps_data contiene los datos GPS recopilados de todos los sensores (coordenadas geog, marca de tiempo, etc.)
- La tabla de sensor data contiene los datos relacionados con cada sensor (nombre del dispositivo, etc.)
- La tabla animals contiene las características registradas para cada animal (clase de edad, especie, etc.) cuando se colocó / retiró el sensor
- Hay otra tabla que se incorporará más adelante (sensor a animales) que vinculará el dispositivo sensor a cada animal y, en última instancia, vinculará los datos GPS a cada animal.

Arranquemos...

- *Agregamos la tabla de sensores y la tabla de animales como hicimos antes*
- *Vinculamos la tabla GPS y la tabla Sensor usando una clave externa:*

Por ejemplo, creamos una clave externa que vincula gps_data con gps_sensors:

```
ALTER TABLE main.gps_data
  ADD CONSTRAINT gps_data_gps_sensors_fkey
  FOREIGN KEY (gps_sensors_code)
  REFERENCES main.gps_sensors (gps_sensors_code)
  MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION;
```

Foreign keys

Puede pensar en esto como una jerarquía, donde sensor_gps datos (table.upstream) están upstream de gps_data (table_downstream)

FOREIGN KEY (variable in the table_downstream)

- REFERENCES schema.table_upstream (variable in the table_upstream)
- MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION: Esta última parte indica que para eliminar un sensor de los datos del sensor, primero debe eliminar todos los registros asociados a ese sensor en el gps_data; Y para agregar cualquier registro a la gps_data asociada a un nuevo sensor, ese sensor debe agregarse primero a los datos del sensor.

Una clave externa crea una dependencia. Por ejemplo:

- No podemos agregar datos GPS de un sensor que no esté en la tabla de sensores
- No podemos agregar un animal de una especie que no esté en la tabla de especies

Tablas auxiliares

Crear tablas auxiliares (tablas de búsqueda) para reforzar la coherencia en la base de datos.

Estas tablas de búsqueda (o lookup tables en inglés) almacenarán la lista y las descripciones de todos los valores posibles.

En este caso, para especies y clases de edad. Se recomienda mantenerlos en un esquema separado para dar a la base de datos una estructura de datos más clara (pero no se aplican restricciones si desea agregarlos al mismo esquema que las tablas que contienen los datos).

Cree las tablas de búsqueda para los valores de los campos species_id y rellénelos con los valores admitidos:

```
CREATE TABLE lu_tables.lu_species(  
  species_code integer,  
  species_description character varying,  
  CONSTRAINT lu_species_pkey  
  PRIMARY KEY (species_code));
```

```
COMMENT ON TABLE lu_tables.lu_species IS 'Look up table for species.';
```

Agregue valores:

```
INSERT INTO lu_tables.lu_species VALUES (1, 'roe deer');  
INSERT INTO lu_tables.lu_species VALUES (2, 'rein deer');  
INSERT INTO lu_tables.lu_species VALUES (3, 'moose');
```

Otro ejemplo de query pero entre tablas

Ejecute una consulta para visualizar los animales de la tabla con la descripción de los campos codificados:

```
SELECT
    animals.animals_id AS id,
    animals.animals_code AS code,
    animals.name,
    animals.sex,
    lu_age_class.age_class_description AS age_class,
    lu_species.species_description AS species
FROM
    lu_tables.lu_age_class,
    lu_tables.lu_species,
    main.animals
WHERE
    lu_age_class.age_class_code = animals.age_class_code
AND
    lu_species.species_code = animals.species_code;
```

Resultado

<i>id</i>	<i>code</i>	<i>name</i>	<i>sex</i>	<i>age_class</i>
1	F09	Daniela	f	adult
2	M03	Agostino	m	adult
3	M06	Sandro	m	adult
4	F10	Alessandra	f	adult
5	M10	Decimo	m	adult

Main.animals

animals Enter a SQL expression to filter results (use Ctrl+Space)

	animals_id	animals_code	name	sex	age_class_code	species_code	note	insert_timestamp
1	1	F09	Daniela	f	3	1	[NULL]	2022-02-07 15:57:34.011 -0500
2	2	M03	Agostino	m	3	1	[NULL]	2022-02-07 15:57:34.011 -0500
3	3	M06	Sandro	m	3	1	[NULL]	2022-02-07 15:57:34.011 -0500
4	4	F10	Alessandra	f	3	1	[NULL]	2022-02-07 15:57:34.011 -0500
5	5	M10	Decimo	m	3	1	[NULL]	2022-02-07 15:57:34.011 -0500

Lu_tables.lu_age_class

lu_age_class Enter a SQL expression to filter results (use C

	age_class_code	age_class_description
1	1	fawn
2	2	yearling
3	3	adult

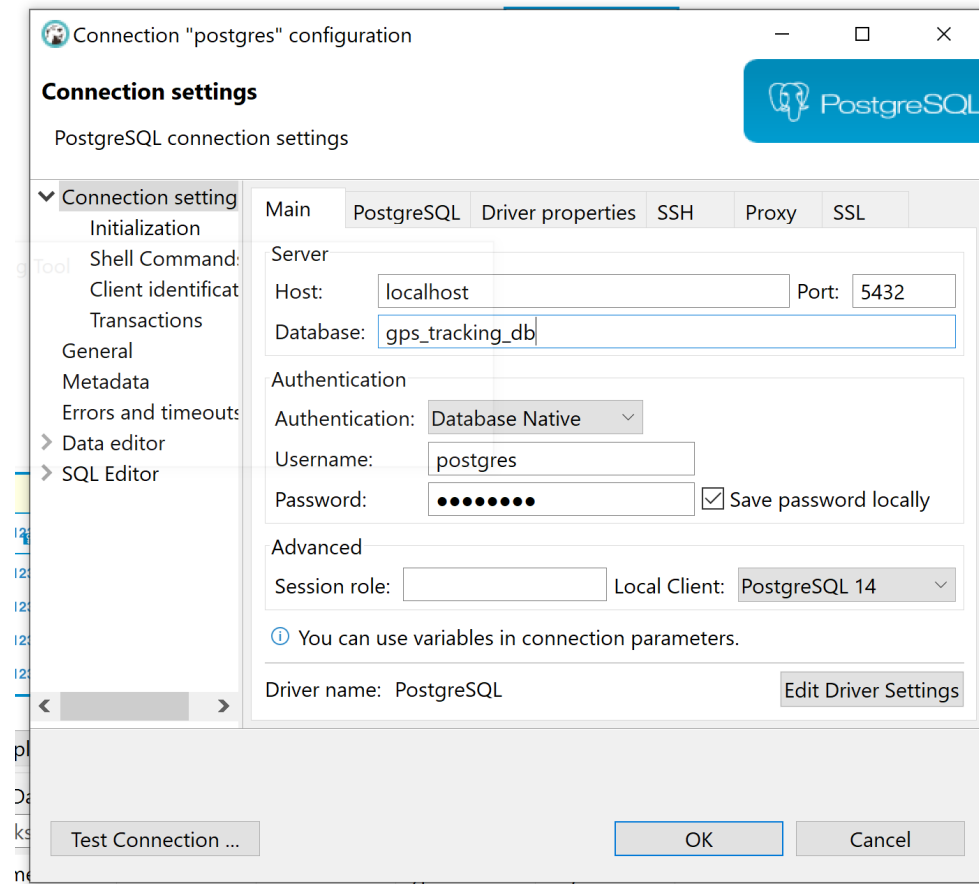
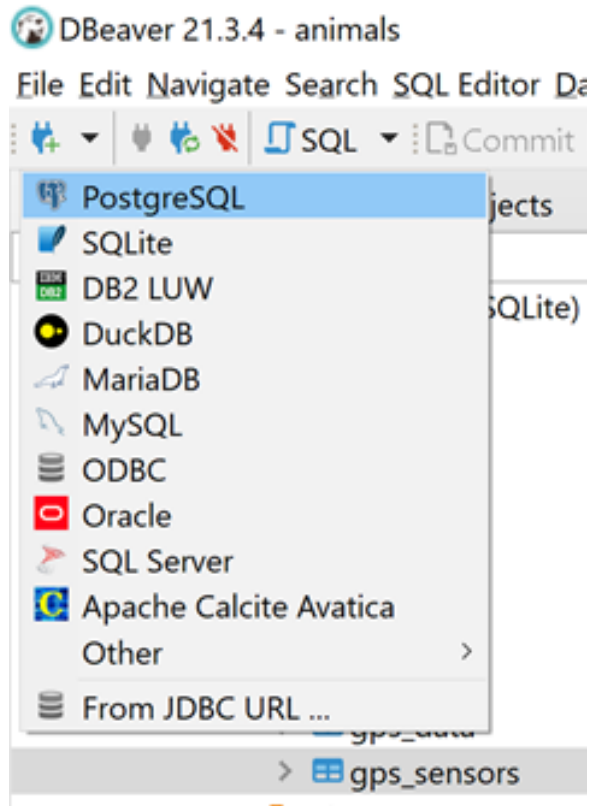
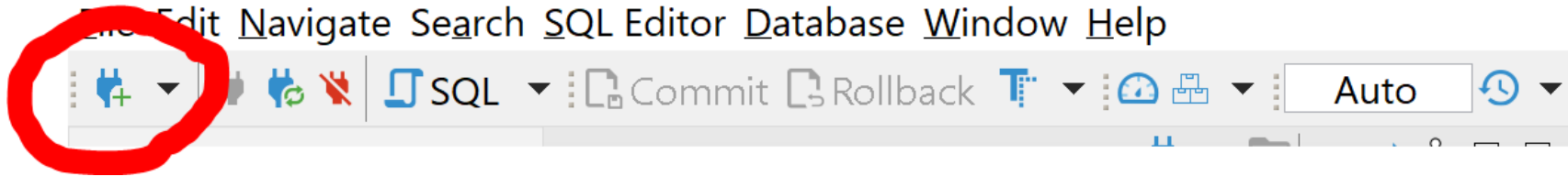
Lu.tables.lu_species

lu_species Enter a SQL expression to filter results (use C

	species_code	species_description
1	1	roe deer
2	2	rein deer
3	3	moose


¿Cómo visualizar las tablas y la estructura de la base de datos?


DBeaver 21.3.4 - animals






▼ postgres - localhost:5432

✓  **gps_tracking_db**

>  lu_tables

▼  main

> animals

32K

6.9M

48K

➤  Material

➤ **Indexes**


Sequences


- **Data types**

➤ **Aggregate**

 public

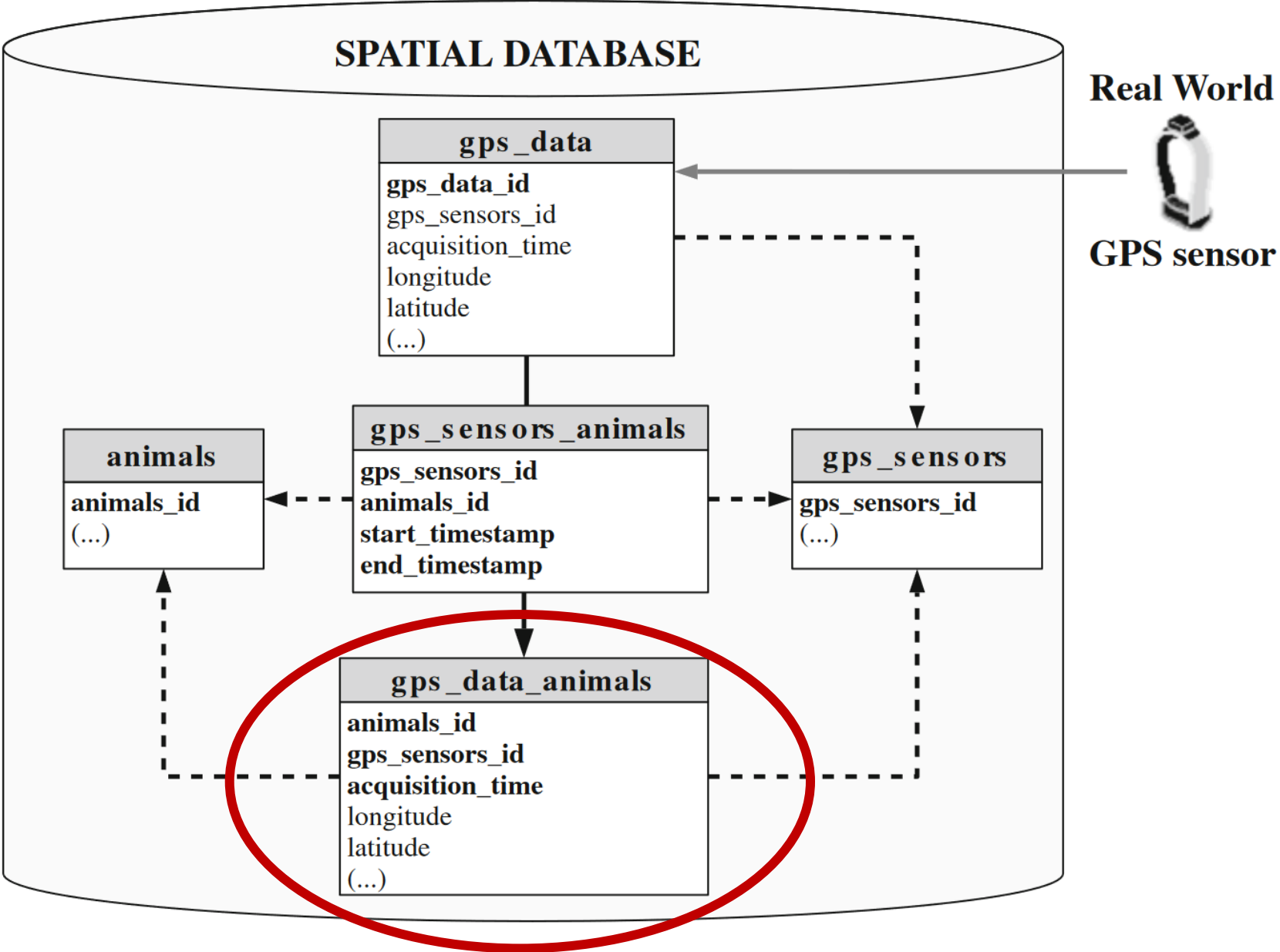
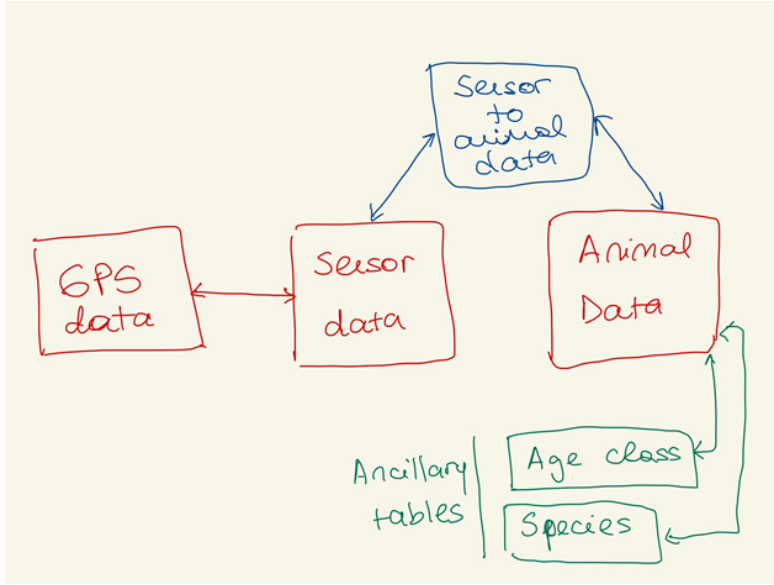
>  public

 plpgsql

>  System Info

➤ Administer

> System Info



Cambios en el timestamp en la base usando triggers

Un **trigger** es una especificación de que la base de datos debe ejecutar automáticamente una función determinada cada vez que se realiza un determinado tipo de operación en una tabla determinada de la base de datos.

El **trigger** dispara una función específica para realizar algunas acciones ANTES o DESPUÉS de que los registros se ELIMINEN, ACTUALICEN o INSERTEN en una tabla.

La función del **trigger** debe definirse antes de crear el propio **trigger**

Por ejemplo, al insertar un nuevo registro en una tabla, puede actualizar otra tabla que debería verse afectada por esta nueva carga.