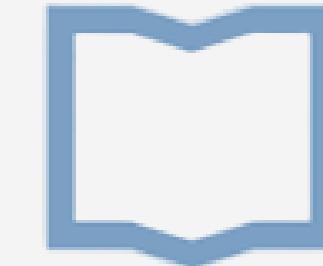
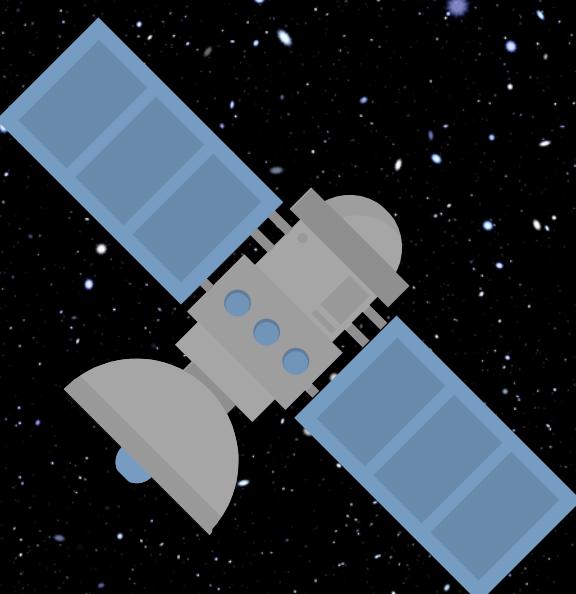
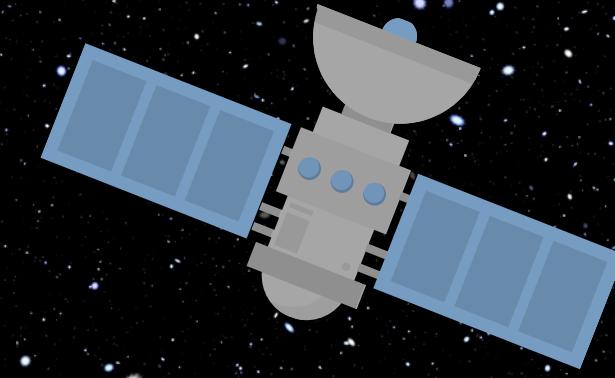


# Argos

## Organización de investigación espacial

Lenguaje SQL

UNSAM - 2°C 2024



Licenciatura en  
Ciencia de Datos  
ECYT\_UNSAM

**Integrantes:**

**Gerardo Toboso**  
**Santino Semec**  
**Alejo Senra**  
**Nicolas Pontiroli**

# **Indice de contenidos**

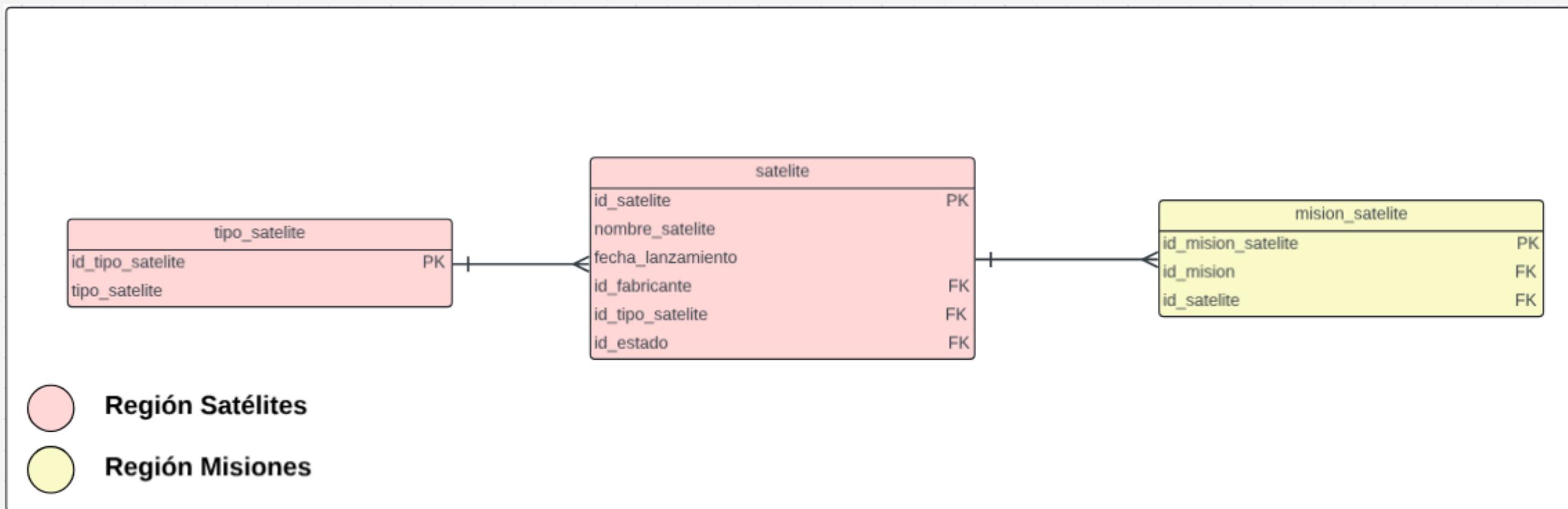
**1- Operaciones sobre tres tablas**

**2- Funcionalidades definidas en la base  
de datos (Functions + Trigger)**

# Operaciones sobre tres tablas

A continuación pasamos a presentar diferentes operaciones de prueba sobre **tres tablas** en nuestra base de datos.

Estas serán **operaciones de consulta** que se realizarán sobre **tres tablas relacionadas**. La relación y las tablas elegidas para llevar a cabo la prueba son las siguientes:



# Operaciones sobre tres tablas

La primera operación consistirá en obtener datos de la tabla “mision\_satelite” a partir de la tabla “tipo\_satelite”.

La idea de esta operación es contar la cantidad de misiones realizadas por cada tipo de satélite disponible en la base de datos.

```
1  SELECT
2      tipo_satelite.tipo_satelite,
3          COUNT(mision_satelite.id_mision) AS cantidad_misiones
4  FROM tipo_satelite
5      JOIN satelite ON satelite.id_tipo_satelite = tipo_satelite.id_tipo_satelite
6      JOIN mision_satelite ON mision_satelite.id_satelite = satelite.id_satelite
7  GROUP BY tipo_satelite.tipo_satelite
8  ORDER BY cantidad_misiones DESC;
```

# Operaciones sobre tres tablas

La segunda operación consistirá en obtener datos de la tabla “satelite” a partir de la tabla “mision\_satelite”.

La idea de esta operación es obtener los identificadores de las misiones realizadas por el satélite “**Erebos**”, esto con el propósito de ver en que misiones se ha visto implicado.

```
1  SELECT
2      satelite.nombre_satelite,
3      mision_satelite.id_mision
4  FROM mision_satelite
5      JOIN satelite ON satelite.id_satelite = mision_satelite.id_satelite
6  WHERE satelite.nombre_satelite = 'Erebos';
```

# Operaciones sobre tres tablas

La última operación consistirá en obtener datos de la tabla “**tipo\_satelite**” a partir de la tabla “**mision\_satelite**”.

La idea de esta operación es obtener los identificadores de las misiones realizadas por los satélites de tipo **“Comunicaciones”**, esto con el propósito de ver en que misiones se han visto implicados este tipo de satélites.

```
1  SELECT
2      mision_satelite.id_mision,
3      tipo_satelite.tipo_satelite
4  FROM mision_satelite
5      JOIN satelite ON satelite.id_satelite = mision_satelite.id_satelite
6      JOIN tipo_satelite ON satelite.id_tipo_satelite = tipo_satelite.id_tipo_satelite
7  WHERE tipo_satelite.tipo_satelite = 'Comunicaciones';
```

# Funcionalidades definidas en la base de datos

Ahora vamos a presentar funcionalidades que definimos dentro de la base de datos de **Argos**.

La definición de estas tiene como objetivo **optimizar ciertas operaciones repetitivas** en la base de datos y reducir considerablemente el **riesgo de error humano** al realizar diversas acciones sobre ella.

# Funcionalidades definidas en la base de datos

**Function:** “actualizar\_estado\_satelite”, como su nombre indica su propósito radica en cambiar el estado de un **satélite** perteneciente a la tabla “satelite”.

Recibe **2** argumentos: el **id** del **satélite** al cual se le va a cambiar el estado y el nuevo estado a colocar en formato de cadena. Entre los **4** estados posibles para un **satélite** se encuentran:

- Activo
- Inactivo
- En mantenimiento
- Retirado

```
1  CREATE OR REPLACE FUNCTION actualizar_estado_satelite(
2      p_id_satelite INT,
3      p_estado_satelite VARCHAR(50)
4  )
5  RETURNS VOID AS $$$
6  DECLARE
7      v_id_estado INT;
8      v_id_satelite INT;
9  BEGIN
10     -- Verifico si el satélite indicado existe
11     SELECT
12         id_satelite
13     INTO v_id_satelite
14     FROM satelite
15     WHERE id_satelite = p_id_satelite;
16
17     -- De no existir lanzo una excepción
18     IF v_id_satelite IS NULL THEN
19         RAISE EXCEPTION 'El satélite de id % no existe', p_id_satelite;
20     END IF;
21
22     -- Busco el id del estado indicado
23     SELECT
24         id_estado_satelite
25     INTO v_id_estado
26     FROM estado_satelite
27     WHERE estado_satelite = p_estado_satelite;
28
29     -- Si no existe, lanzo una excepción
30     IF v_id_estado IS NULL THEN
31         RAISE EXCEPTION 'El estado del satélite % no existe', p_estado_satelite;
32     END IF;
33
34     -- Finalmente, actualizo el estado del satélite
35     UPDATE satelite
36     SET id_estado = v_id_estado
37     WHERE id_satelite = p_id_satelite;
38
39 END;
40 $$ LANGUAGE plpgsql;
```

# Funcionalidades definidas en la base de datos

**Function:** “`reasignar_estacion_satelite`”, esta función tiene como propósito cambiar la **estación** a la cual un **satélite** manda información, la misma opera sobre la tabla “`estacion_satelite`”.

Recibe **3** argumentos: el **id** del **satélite** que se reasignará, el **id** de la **estación** en donde operaba y el **id** de la nueva estación donde va a empezar a operar.

Más allá de la reasignación de un **satélite**, esta función tiene como propósito mantener la **consistencia** en la tabla “`estacion_satelite`”, evitando que aparezca más de una vez la misma relación entre **satélite** y **estación** o tratar de asignar un **satélite** a una **estación** que no existe.

```
1  CREATE OR REPLACE FUNCTION reasignar_estacion_satelite(
2      p_id_satelite INT,
3      p_id_estacion_anterior INT,
4      p_id_nueva_estacion INT
5  )
6  RETURNS VOID AS $$ 
7  DECLARE
8      v_id_estacion_anterior INT;
9      v_id_estacion_nueva INT;
10     v_id_relacion INT;
11     v_id_nueva_relacion INT;
12     v_id_satelite INT;
13 BEGIN
14     -- Verifico si el satélite indicado existe
15     SELECT
16         id_satelite
17     INTO v_id_satelite
18     FROM satelite
19     WHERE id_satelite = p_id_satelite;
20
21     -- De no existir lanza una excepción
22     IF v_id_satelite IS NULL THEN
23         RAISE EXCEPTION 'El satélite de id % no existe', p_id_satelite;
24     END IF;
25
26     -- Verifico si la estación terrestre anterior existe
27     SELECT
28         id_estacion
29     INTO v_id_estacion_anterior
30     FROM estacion_terrestre
31     WHERE id_estacion = p_id_estacion_anterior;
32
33     -- De no existir lanza una excepción
34     IF v_id_estacion_anterior IS NULL THEN
35         RAISE EXCEPTION 'La estación terrestre anterior con id % no existe', p_id_estacion_anterior;
36     END IF;
37
```

```
38      -- Verifico que el satélite indicado y la estación anterior hayan estado relacionados
39      SELECT
40          id_estacion_satelite
41      INTO v_id_relacion
42      FROM estacion_satelite
43      WHERE id_estacion_terrestre = p_id_estacion_anterior AND id_satelite = p_id_satelite;
44
45      -- De no existir la relación lanza una excepción
46      IF v_id_relacion IS NULL THEN
47          RAISE EXCEPTION 'La estación terrestre con id % no tiene asociado al satélite con id %', p_id_estacion_anterior, p_id_satelite;
48      END IF;
49
50      -- Verifico si la nueva estación terrestre existe
51      SELECT
52          id_estacion
53      INTO v_id_estacion_nueva
54      FROM estacion_terrestre
55      WHERE id_estacion = p_id_nueva_estacion;
56
57      -- De no existir lanza una excepción
58      IF v_id_estacion_nueva IS NULL THEN
59          RAISE EXCEPTION 'La estación terrestre con id % no existe', p_nueva_estacion;
60      END IF;
61
62      -- Verifico que el satélite indicado y la nueva estación no estén relacionados
63      SELECT
64          id_estacion_satelite
65      INTO v_id_nueva_relacion
66      FROM estacion_satelite
67      WHERE id_estacion_terrestre = p_id_nueva_estacion AND id_satelite = p_id_satelite;
68
69      -- De existir la relación lanza una excepción
70      IF v_id_nueva_relacion IS NOT NULL THEN
71          RAISE EXCEPTION 'La estación terrestre con id % ya tiene asociado al satélite con id %', p_id_nueva_estacion, p_id_satelite;
72      END IF;
```

```
74      -- Finalmente reasigno el satélite a la nueva estación
75      UPDATE estacion_satelite
76      SET id_estacion_terrestre = p_id_nueva_estacion
77      WHERE id_satelite = p_id_satelite AND id_estacion_terrestre = p_id_estacion_anterior;
78
79      END;
80      $$ LANGUAGE plpgsql;
```

# Funcionalidades definidas en la base de datos

**Trigger:** "verificar\_estado\_satelite", este trigger tiene como propósito mantener la **consistencia** en la tabla "**observacion**" ante la inserción de un nuevo registro en la misma.

Este se encarga de verificar que antes de registrar una nueva observación, el **satélite** que la realizó figuraba como **Activo** en el sistema, de no ser así este **Trigger** se encarga de lanzar **una excepción** alertando al programador de esta posible **inconsistencia**.

De este modo se logra **evitar** registrar observaciones de **satélites** que no se encuentran activos o algún otro posible **error humano** a la hora de la inserción.

```
1 CREATE OR REPLACE FUNCTION verificar_estado_satelite()
2 RETURNS TRIGGER AS $$ 
3 DECLARE
4     estado_satelite INT;
5 BEGIN
6     -- Primero selecciono el estado del satélite que realizó la observación
7     SELECT
8         id_estado
9         INTO estado_satelite
10        FROM satelite
11       WHERE id_satelite = NEW.id_satelite;
12
13     -- Verifico que el satélite del cual se quiere insertar observaciones esté activo
14     IF estado_satelite <> 1 THEN
15         RAISE EXCEPTION 'Se está tratando de insertar observaciones pertenecientes a un satélite que no está activo';
16     END IF;
17
18     RETURN NEW;
19 END;
20 $$ LANGUAGE plpgsql;
21
22 CREATE TRIGGER trigger_verificar_estado_satelite
23 BEFORE INSERT ON observacion
24 FOR EACH ROW
25 EXECUTE FUNCTION verificar_estado_satelite();
```

**Gracias por ver!**