

Uso de bases de datos / bases de datos

Lenguaje SQL I: creación y manipulación de BD

PRESENTACIÓN Y OBJETIVOS

El objetivo de este documento es facilitar el aprendizaje de la parte básica del lenguaje SQL, la cual se explica en el módulo didáctico "El lenguaje SQL I". Se proponen una serie de ejercicios y su solución. En el fichero DBVideoGamesI.sql tenéis todas las sentencias SQL que conforman este documento para que las podáis probar más cómodamente en PostgreSQL, ya sea usando el pdAdmin o l'SQuirreL.

1. CREACIÓN DE LA BD DE VIDEOJUEGOS

Una tienda dedicada al alquiler de videojuegos (todos para consola y de un mismo fabricante) ha decidido crear una pequeña BD que permita hacer la gestión de los alquileres de los videojuegos que posee. En concreto la tienda ha decidido crear las siguientes tablas (claves primarias subrayadas):

video_games(game_code, game_name, rental_fee, min_age, total_amount)

De cada videojuego guardamos su código (identificador, entero), el nombre del juego (cadena de caracteres, máximo de 30), cuánto cuesta alquilar el videojuego, la edad mínima recomendada y la cantidad total de copias que de cada videojuego dispone la tienda. Ningún atributo puede tomar valor nulo.

Algunas consideraciones a tener en cuenta serían:

- a) No hay dos juegos con el mismo nombre.
- b) El importe del alquiler será un valor positivo (hasta dos decimales) que no puede superar los 100 euros.
- c) No tendremos juegos con edad mínima por debajo de 4 años.
- d) La cantidad total expresa cuántas copias disponemos de un determinado videojuego. Esta cantidad irá entre 1 y 10 copias. Por defecto, existirá una copia.

customers(<u>customer_code</u>, customer_name, age, phone_number)

De cada cliente tenemos almacenado su código (identificador, entero), nombre (cadena de caracteres, máximo 50), edad y teléfono. Ningún atributo puede tomar valor nulo.

Algunas consideraciones a tener en cuenta serían:



- a) Pueden existir clientes con el mismo nombre (suponemos que el nombre incluye nombre y primer apellido).
- b) Sólo se admiten clientes entre 4 y 100 años.

employees(empl_code, empl_name, salary, age)

Datos de los empleados de la tienda, su código (identificador, entero), su nombre (cadena de caracteres, máximo 50), sueldo y edad. Ningún atributo puede tomar valor nulo.

Algunas consideraciones a tener en cuenta serían:

- a) No pueden existir dos empleados con el mismo nombre (el nombre también incluye el primer apellido).
- b) El sueldo es un valor positivo (hasta dos decimales). No se admiten sueldos por debajo de los 300 euros ni por encima de 800 euros.
- c) La edad de los empleados tiene que estar comprendida entre los 18 y los 65 años.

game_rental(game_code, customer_code, rental_date, ret_date, empl_code)

Esta tabla guardará toda la historia de alquileres realizados en la tienda. De cada alquiler guardaremos el código del juego, el código del cliente que efectúa el alquiler, la fecha en la que el cliente se lleva el juego, la fecha en que se devuelve el vídeo alquilado y el código del empleado que ha realizado el alquiler. El único atributo que puede tomar valor nulo es la fecha de devolución.

Una vez se hayan verificado las condiciones para hacer efectivo un alquiler (más adelante diremos cuáles son estas condiciones), introduciremos los datos del alquiler: código del juego, el código del cliente que alquila el juego, la fecha que se hace efectivo el alquiler y el código del empleado que hace el alquiler. La fecha de devolución inicialmente es nula. Esta fecha de devolución dejará de ser nula en la fecha que el cliente devuelva el juego alquilado.

Algunas consideraciones a tener en cuenta serían:

- a) Un cliente puede alquilar un mismo juego, siempre y cuando lo haga en fechas diferentes.
- b) La fecha de alquiler es anterior o igual a la fecha de devolución, siempre y cuando la fecha de devolución no sea nula. La fecha del alquiler podría ser por defecto la fecha actual.
- c) El cliente que alquila un juego y el empleado que efectúa un alquiler tienen que existir, respectivamente, en las tablas de clientes y de empleados.
- d) El juego a alquilar tiene que existir previamente en la tabla de videojuegos.



Se pide:

Proponer el conjunto de sentencias SQL necesarias con el fin de definir y crear una BD sobre PostgreSQL que se ajuste al diseño previo. Es importante destacar que el nombre de las tablas y de sus columnas tiene que coincidir exactamente con los nombres de las relaciones y de los atributos que se proporciona en este enunciado.

Además, a la hora de definir y crear la BD hay que considerar las **siguientes** restricciones:

- a) Todas aquellas restricciones que sean inherentes al modelo relacional (como, por ejemplo, claves primarias y claves foráneas) y todas aquéllas que se puedan deducir a partir del enunciado previo (restricciones de UNIQUE para un atributo o conjunto de atributos, admisión de valores nulos o no, etc.).
- b) Las restricciones específicas que han sido definidas en el enunciado.
- c) Adicionalmente, hay que añadir las siguientes condiciones para hacer efectivo un alquiler. Estas condiciones se tienen que implementar mediante las siguientes aserciones:
 - Aserción 1: Un cliente sólo puede alquilar juegos apropiados a su edad. Es decir, la edad del cliente es mayor o igual a la edad recomendada por el fabricante del videojuego.
 - Aserción 2: Sólo se puede efectuar un alquiler de un videojuego si tenemos copias suficientes, es decir, si de un videojuego existen 5 copias no podrán existir más de 5 alquileres activos para el videojuego en cuestión. Entendemos que los alquileres activos son aquéllos que tienen la fecha de devolución igual a nulo.

Nota 1: recordad que las aserciones no están soportadas por ningún SGBD comercial, por lo tanto tendréis que trabajar desde un punto de vista teórico (en definitiva, tenéis que proporcionar el texto asociado a la aserción).

Nota 2: adicionalmente, y por los motivos mencionados en la **nota 1** seremos nosotros, en última instancia, los responsables de que los datos de nuestra BD verifiquen las condiciones antes mencionadas. En un documento separado (DBVideoGamesII.pdf) implementaremos los mecanismos necesarios para que el SGBD (PostgreSQL) lo haga de manera automática.



Se proponen las siguientes sentencias de creación de tablas:

```
-- Create the SCHEMA to work on
CREATE SCHEMA videogames;
SET search path TO videogames, "$user", public;
-- Create tables
BEGIN WORK:
CREATE TABLE video_games(
       game code INTEGER,
       game name VARCHAR(30) NOT NULL,
       rental fee DECIMAL(5,2) NOT NULL,
       min age INTEGER NOT NULL,
       total amount INTEGER DEFAULT 1 NOT NULL,
       CONSTRAINT pk_video_games PRIMARY KEY (game_code),
       CONSTRAINT u_game_name UNIQUE(game_name),
       CONSTRAINT ck fee CHECK(rental fee > 0 AND rental fee <= 100),
       CONSTRAINT ck min age CHECK (min age >= 4),
       CONSTRAINT ck_total_amount CHECK(total amount >= 1)
);
CREATE TABLE customers (
       customer code INTEGER,
       customer name VARCHAR(50) NOT NULL,
       age INTEGER NOT NULL,
       phone_number CHAR(9) NOT NULL,
       CONSTRAINT pk customers PRIMARY KEY (customer code),
       CONSTRAINT ck_age CHECK(age BETWEEN 4 AND 100)
);
CREATE TABLE employees (
       empl code INTEGER,
       empl_name VARCHAR(50) NOT NULL,
       salary DECIMAL(5,2) NOT NULL,
       age INTEGER NOT NULL,
       CONSTRAINT pk employee PRIMARY KEY(empl code),
       CONSTRAINT u employees UNIQUE(empl name),
       CONSTRAINT ck_salary CHECK(salary BETWEEN 300 AND 800),
       CONSTRAINT ck empl age CHECK(age BETWEEN 18 AND 65)
);
CREATE TABLE game_rental(
       game code INTEGER,
       customer code INTEGER,
       rental date DATE DEFAULT CURRENT DATE,
       ret_date DATE DEFAULT NULL,
       empl code INTEGER NOT NULL,
       CONSTRAINT pk game rental PRIMARY KEY(game code, customer code, rental date),
       CONSTRAINT fk_video_games FOREIGN KEY(game_code) REFERENCES video_games(game_code),
       CONSTRAINT
                       fk_customers
                                          FOREIGN
                                                        KEY(customer code)
customers(customer_code),
       CONSTRAINT fk employees FOREIGN KEY(empl code) REFERENCES employees(empl code),
       CONSTRAINT ck dates CHECK(ret date IS NULL OR rental date <= ret date)
);
COMMIT WORK;
```

Las sentencias previas presentan la creación de las tablas teniendo en cuenta las restricciones expresadas en el enunciado. Aspectos importantes a destacar:

a) En el caso de los atributos que son o forman parte de claves primarias no hay que indicar NOT NULL, dado que es una condición que está implícita en la restricción PRIMARY KEY.



- b) Las restricciones se han impuesto como restricciones de tabla, aunque aquellas restricciones que afectan sólo a una columna se podrían haber definido como restricción de columna. Éste es, por ejemplo, el caso de claves primarias de sólo un atributo; queremos destacar que, conceptualmente, es más elegante definirlas como a restricción de tabla, dado que el concepto de clave primaria, de hecho, es una propiedad que afecta en la tabla en su conjunto y no únicamente a un atributo.
- c) Hemos impuesto nombre a las restricciones por elegancia; también, en caso de violaciones, obtendremos información extra de qué restricción en concreto se ha violado en cada tabla.
- d) La creación de tablas constituye una unidad atómica de ejecución, es decir una transacción.
- e) Como las cadenas de caracteres son de longitud variable, se ha escogido como tipo de datos el VARCHAR. La excepción son los teléfonos que siempre constan de 9 caracteres.

Finalmente, se proponen las siguientes aserciones:

```
- assertion 1: A customer can only rent games that are rated as suitable for their age.
-- That is, customer age is the same as or greater than the age recommended by the
manufacturer
-- of the video game.
CREATE ASSERTION ass1 AS (NOT EXISTS
       (SELECT
        FROM game_rental a, customers c, video_games v
        WHERE a.game_code = v.game_code AND
             a.customer_code = c.customer code AND
              c.age < v.min age));</pre>
-- assertion 2: A rental can only be made if there are enough copies in store. That is, if
we have 5 -- copies of a game, there cannot be 5 simultaneous active rentals of that game.
Active rentals are
-- those with a date of null date of return.
CREATE ASSERTION ass2 AS (NOT EXISTS
        FROM video games v
        WHERE v.total_amount < (SELECT COUNT(*)
                             FROM game rental a
                             WHERE a.game code = v.game code AND
                                 a.ret date IS NULL
                             GROUP BY a.game_code)));
```

En relación a las aserciones propuestas recordad:

- a) Que no las podéis probar, tal cual ni en PostgreSQL, ni en ningún otro producto comercial. Para implementar estos tipos de restricciones el mecanismo que finalmente se ha impuesto ha sido el disparador.
- b) Que la manera más sencilla de especificar aserciones es definir lo que no puede pasar, tal como se ha explicado en la GES del módulo 4 con ejemplos.



2. INSERCIÓN DE DATOS EN BD DE VIDEOJUEGOS

Proponer el conjunto de filas a insertar dentro de cada tabla. Se valorará la calidad (que no la cantidad) de datos introducidos. La calidad de los datos significa:

- a) Que los datos verifican las restricciones de integridad (inherentes al modelo relacional o específicas a la BD que estamos modelando) que se hayan especificado en el enunciado.
- b) Que los datos introducidos permitan probar adecuadamente la corrección de las soluciones propuestas a los siguientes ejercicios (consultas, modificaciones, borrados y creación de vistas).

Nota: Las fechas, en principio, se tendrán que introducir con el formato siguiente: mm-dd-aaaa (mes, día, año).

Se propone el siguiente conjunto de inserciones (la ejecución de todas las sentencias de inserción se considera una transacción):

```
-- Insert rows into tables
BEGIN WORK;
SET DATESTYLE = MDY:
INSERT INTO video_games VALUES(1, 'J001' ,80, 14, 5);
INSERT INTO video_games VALUES(2, 'J002' ,90, 18, 3);
INSERT INTO video_games VALUES(6, 'J006' ,90, 18, 2);
INSERT INTO video_games VALUES(7, 'J007' ,10, 4, 1);
INSERT INTO customers VALUES(1, 'Pablo Roig', 18, '934505151');
INSERT INTO customers VALUES(2, 'Maria Ba', 21, '916800000');
INSERT INTO CUSTOMERS VALUES(2, Maria Ba, 21, 910000000);
INSERT INTO customers VALUES(3, 'Pepe Puig', 14, '933500000');
INSERT INTO customers VALUES(4, 'Ana Ruiz', 18, '932660000');
INSERT INTO customers VALUES(5, 'Mario Caro', 21, '974600000');
INSERT INTO customers VALUES(6, 'Pepe Perez', 15, '913000000');
INSERT INTO customers VALUES(7, 'Clara Diaz', 18, '982428000');
INSERT INTO customers VALUES(8, 'Pepe Perez', 21, '938900000');
INSERT INTO customers VALUES(9, 'Raul Cano', 50, '981560000');
INSERT INTO employees VALUES(1, 'Ramon Pi', 350, 21);
INSERT INTO employees VALUES(2, 'Sara Ruso', 400, 40);
INSERT INTO employees VALUES(3, 'Juan Paz', 600, 25);
INSERT INTO employees VALUES(4, 'Angel Ros', 350.25, 18);
INSERT INTO employees VALUES(5, 'Marc Coimbra', 500, 40);
INSERT INTO game_rental VALUES(1, 1, '02-27-2006', NULL, 1);
INSERT INTO game_rental VALUES(2, 1, '02-20-2006', '03-01-2006', 1);
INSERT INTO game rental VALUES(3, 2, CURRENT_DATE, NULL, 2);
INSERT INTO game_rental VALUES(4, 1, '02-28-2006', NULL, 2); INSERT INTO game_rental VALUES(3, 5, '03-01-2006', NULL, 1);
INSERT INTO game_rental VALUES(4, 2, '03-01-2006', NULL, 2);
INSERT INTO game_rental VALUES(2, 2, '02-10-2006', '02-20-2006', 2);
INSERT INTO game_rental VALUES(2, 2, '02-10-2006', '02-20-2006', 2); INSERT INTO game_rental VALUES(5, 6, '02-10-2006', '02-20-2006', 2); INSERT INTO game_rental VALUES(5, 7, '02-10-2006', NULL, 1); INSERT INTO game_rental VALUES(5, 8, '02-10-2006', NULL, 3); INSERT INTO game_rental VALUES(5, 6, '03-01-2006', NULL, 4); INSERT INTO game_rental VALUES(5, 9, '02-18-2006', NULL, 1);
COMMIT WORK;
```



```
-- Check inserted data

BEGIN WORK;

SELECT * FROM video_games;
SELECT * FROM customers;
SELECT * FROM employees;
SELECT * FROM game_rental;

COMMIT WORK;
```

Podemos comprobar que el conjunto de filas insertado en las tablas verifica las restricciones de integridad especificadas en las aserciones, ejecutando las consultas asociadas a las aserciones y verificando que las consultas no devuelven ningún dato.

```
BEGIN WORK;
-- assertion 1: A customer can only rent games that are rated as suitable for their age.
-- That is, customer age is the same as or greater than the age recommended by the
manufacturer
-- of the video game.
FROM game_rental ll, customers c, video_games v WHERE ll.game_code = v.game_code AND
     11.customer code = c.customer code AND
      c.age < v.min age;
-- assertion: A rental can only be made if there are enough copies in store. That is, if
we have 5
-- copies of a game, there cannot be 5 simultaneous active rentals of that game. Active
rentals are
-- those with a null date of return.
SELECT *
FROM video games v
WHERE v.total_amount < (SELECT COUNT(*)
                      FROM game rental 11
                      WHERE 11.game code = v.game code AND
                           ll.ret date IS NULL
                      GROUP BY 11.game code);
COMMIT WORK;
```

Un buen ejercicio complementario, sería insertar filas que violen las aserciones, volver a ejecutar las consultas y ver que después sí devuelven datos.

3. CONSULTAS A LA BD DE VIDEOJUEGOS

Sobre la BD creada en el ejercicio 1 y con los datos insertados en el ejercicio 2, resolver las siguientes consultas:

 a) Nombre y edad de los clientes que nunca han alquilado (ni tienen alquilado actualmente) videojuegos con importe de alquiler superior a los 60 euros. El resultado se quiere ordenado por edad.



```
SELECT c.customer_name, c.age
FROM customers c
WHERE c.customer code NOT IN (SELECT 11.customer_code
                                        FROM game_rental ll, video_games v
                                       WHERE v.game_code = ll.game_code AND
                                     v.rental fee > 60)
ORDER BY c.age;
-- or alternatively (using INNER JOIN)
SELECT c.customer name, c.age
FROM customers c
WHERE c.customer_code NOT IN (SELECT 11.customer code
                                     FROM game_rental 11
                                     INNER JOIN video games v ON v.game code=11.game code
                                     WHERE v.rental fee>60)
ORDER BY c.age;
-- or, alternatively, using NOT EXISTS
SELECT c.customer_name, c.age
FROM customers c
WHERE NOT EXISTS (SELECT *
                         FROM game rental 11, video games v
                         WHERE v.game_code = ll.game_code AND
                        v.rental_fee > 60 AND
                        c.customer_code = ll.customer_code)
ORDER BY c.age;
-- and, alternatively, with INNER JOIN;
SELECT c.customer_name, c.age
FROM customers c
WHERE NOT EXISTS (SELECT *
                         FROM game rental 11
                         INNER JOIN video_games v ON v.game_code = ll.game_code
                         WHERE v.rental_fee > 60 AND
                        c.customer code = 11.customer code)
ORDER BY c.age;
```

El resultado asociado es:

```
Query Editor Query History
1
   SELECT c.customer_name, c.age
2
    FROM customers c
3
    WHERE NOT EXISTS (SELECT *
4
               FROM game_rental ll
5
               INNER JOIN video_games v ON v.game_code = ll.game_code
6
               WHERE v.rental_fee > 60 AND
                             c.customer_code = ll.customer_code)
7
8
    ORDER BY c.age;
```

Dat	ta Output Explain Me	ssages	Notifications
4	customer_name character varying (50)	age integer	
1	Pepe Puig	14	
2	Ana Ruiz	18	
3	Mario Caro	21	

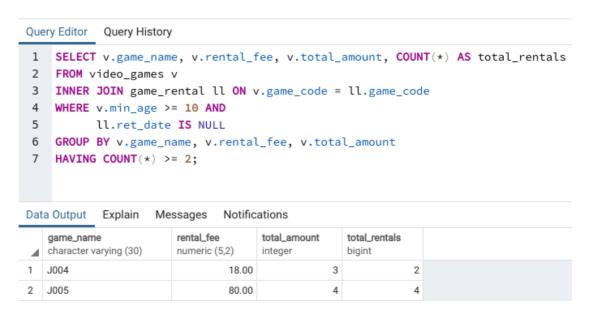
pág. 8



b) Nombre, importe y cantidad total de copias de aquellos videojuegos con edad mínima de 10 años de los cuales, como mínimo, existen 2 alquileres activos. También hay que proporcionar el número de alquileres activos de cada videojuego que verifique las condiciones mencionadas en la consulta.

Hemos puesto nombre a la columna que representa el cálculo del agregado, aunque estrictamente no es necesario, pero es más elegante, dado que aporta semántica adicional al usuario que ejecuta la consulta.

El resultado asociado es:



c) Código, nombre y edad de los clientes de Madrid que son mayores que algunos de los clientes de Barcelona. También, para cada cliente de Madrid que sea mayor, se quiere saber el número de clientes de Barcelona que son más jóvenes. El resultado se quiere sin repeticiones.



```
SELECT c.customer_code, c.customer_name, c.age, COUNT(*) AS total_customer_BCN
FROM customers c, customers c1
WHERE c.phone_number LIKE '91%' AND
c1.phone_number LIKE '93%' AND
c.age > c1.age
GROUP BY c.customer_code, c.customer_name, c.age;
```

No hay que usar la cláusula DISTINCT dado que la cláusula GROUP BY elimina los duplicados. Esta afirmación es cierta, porque todos los atributos del GROUP BY también forman parte del conjunto de atributos que se seleccionan. De lo contrario, sí que podrían existir duplicados.

Los teléfonos de Madrid empiezan por 91 y los de Barcelona por 93, por eso utilizamos el símbolo de % que asegura la busca de cadenas que contienen los prefijos mencionados. Obviamente, nos podemos despistar de clientes, en caso de tener teléfonos móviles, pero con el esquema de la BD este problema no se puede resolver.

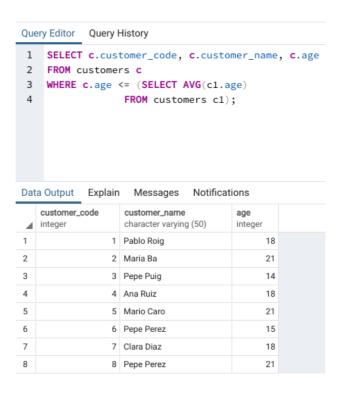
El resultado asociado es:

```
Query Editor Query History
    SELECT c.customer_code, c.customer_name, c.age, COUNT(*) AS total_customer_BCN
2
    FROM customers c, customers c1
3
   WHERE c.phone_number LIKE '91%' AND
4
           c1.phone_number LIKE '93%' AND
5
           c.age > c1.age
    GROUP BY c.customer_code, c.customer_name, c.age;
Data Output Explain Messages
                               Notifications
   customer_code
                   customer_name
                                         age
                                                 total_customer_bcn
   integer
                   character varying (50)
                                        integer
                                                 bigint
                 2 Maria Ba
                                              21
                                                                  3
1
2
                 6 Pepe Perez
                                              15
                                                                  1
```

d) Código y nombre de los clientes que tienen edad inferior o igual a la edad media de los clientes.

El resultado asociado es:





e) Código, nombre y sueldo de los empleados que cobran menos que el empleado (o empleados) con edad máxima.

```
SELECT e.empl_code, e.empl_name, e.salary
FROM employees e
WHERE e.salary < ALL (SELECT el.salary
FROM employees el
WHERE el.age = (SELECT MAX(e2.age)
FROM employees e2));
```

Ponemos la cláusula ALL dado que pueden existir más de un empleado con edad máxima y estos empleados pueden tener sueldos diferentes.

El resultado asociado es:

Que	ery E	ditor	Query History					
1	SE	ELECT	e.empl_code, e.empl_name, e.salary	/				
2	FR	FROM employees e						
3	WH	WHERE e.salary < ALL (SELECT el.salary						
4		FROM employees e1						
5		WHERE el.age = (SELECT MAX(e2.age)						
_		<pre>FROM employees e2));</pre>						
6			FROM employees e	2));				
Ū	ta Ou	itput l	FROM employees e Explain Messages Notifications	e2));				
J		ol_code		e2));				
J	emp	ol_code	Explain Messages Notifications empl_name salary	e2));				

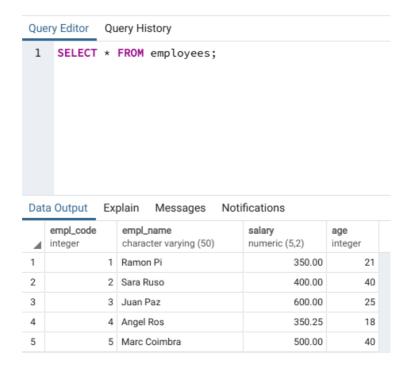


4. MODIFICACIÓN DE LA BD DE VIDEOJUEGOS

Incrementar un 10% el sueldo de aquellos empleados que tienen un mínimo de cuatro alquileres activos.

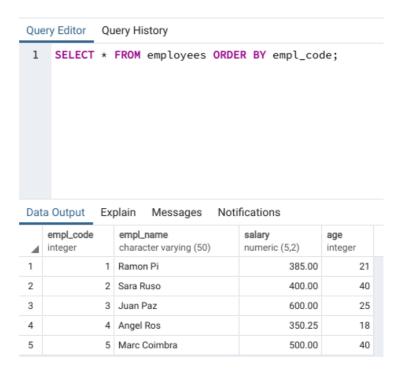
Con el conjunto de datos que hemos insertado, sólo verifica las condiciones el empleado con código de empleado igual a 1. Por lo tanto, sólo se modificará un sueldo. Proponemos la siguiente secuencia de sentencias SQL (son tratadas como una transacción):

Antes de la modificación el sueldo del empleado con código de empleado igual a 1 es:



Después de la modificación el sueldo del empleado con empleo code igual a 1 es:





Es importante destacar que, como consecuencia de este update, el check asociado (sueldo máximo permitido) se puede violar. Si fuera el caso, PostgreSQL nos avisaría de la situación con un mensaje de error, y se suspendería la ejecución del update. Teniendo en cuenta el concepto de transacción (que estudiaremos en profundidad en el módulo 6, se ejecuta todo o no se ejecuta nada), automáticamente se anularían todos los cambios que se hubieran realizado hasta el momento de producirse el error. No obstante, conceptualmente sería más correcto, de detectarse el error, finalizar con ROLLBACK, en lugar de con un COMMIT.

5. BORRADO DE LA BD DE VIDEOJUEGOS

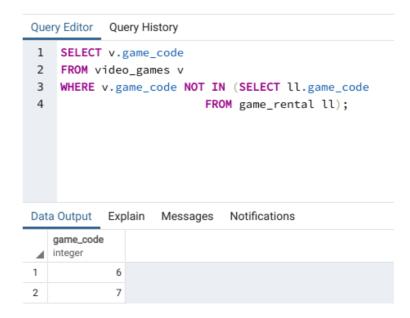
Eliminar de la base de datos los videojuegos que ni están ni nunca han sido alquilados por ningún cliente

Sólo verifican las condiciones mencionadas en el proceso de borrado los juegos con game_code 6 y 7. Proponemos la siguiente secuencia de sentencias SQL (son tratadas como una transacción):

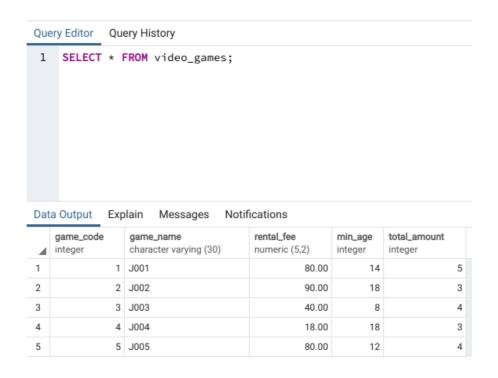


```
SELECT * FROM video_games;
COMMIT WORK;
```

Miramos los datos de los videojuegos con game code igual a 6 y 7:



Borramos y comprobamos que los videojuegos con $game_code$ igual a 6 y 7 han desaparecido:





6. CREACIÓN DE VISTAS

Crear las siguientes vistas:

a) Crear una vista que muestre todos los datos (código juego, código cliente, fecha alquiler, fecha de devolución y código de empleado) de los alquileres de vídeo juegos (independientemente que estos alquileres estén o no activos) realizados durante el mes de Febrero. ¿Cuál es el contenido de la vista? ¿Es actualizable esta vista? Es necesario que argumentéis brevemente vuestra respuesta.

Se propone el siguiente conjunto de sentencias SQL:

```
BEGIN WORK;
CREATE VIEW february rentals AS
FROM game rental
WHERE rental date BETWEEN '02-01-2006' AND '02-28-2006';
END;
BEGIN:
SELECT * FROM february rentals;
-- Trying to insert a new rental in February
INSERT INTO february rentals VALUES (4,1,'02-23-2006', NULL, 5);
SELECT * FROM february rentals;
-- Inserted rental is wrong (it does not correspond to February), but
-- the DBMS allows it and inserts the row into game rentals.
INSERT INTO february rentals VALUES (4,1,'03-23-2006', NULL, 5);
-- Modifying rentals other than those from February (no rows will be modified)
UPDATE february rentals SET empl code = 5 WHERE game code = 3 AND customer code = 2
      AND rental date = '2019-10-01';
ROLLBACK;
```

A pesar que la vista contiene todos los campos de la tabla, por lo que debería ser actualizable, PostgreSQL no permite actualizar las vistas sin utilizar las RULES (o para ser más exactos, sin utilizar el *query rewrite rule system*).

b) Crear una vista que muestre el nombre y el sueldo de los empleados que han alquilado videojuegos a clientes con edad por encima de los 20 años. Es independiente que los alquileres estén o no activos. El resultado se quiere sin repeticiones. ¿Cuál es el contenido de la vista? ¿Es actualizable esta vista? Es necesario que argumentéis **brevemente** vuestra respuesta.

A continuación mostramos la sentencia que crea la vista y la sentencia que consulta el contenido de la vista que justamente se acaba de crear.



```
BEGIN WORK;

CREATE VIEW rental_empl AS
SELECT DISTINCT e.empl_name, e.salary
FROM employees e, customers c, game_rental a
WHERE e.empl_code = a.empl_code AND c.customer_code = a.customer_code AND c.age > 20;

-- Check the content of the view

SELECT * FROM rental_empl;

COMMIT WORK;
```

La vista no es actualizable ni utilizando PostgreSQL ni utilizando otros gestores, puesto que incorpora, por ejemplo, operaciones de combinación (*join*), que causan (ver material didáctico) que ciertas operaciones de cambio (por ejemplo, borrados) no tengan una semántica perfectamente definida (no está claro si sólo se quieren borrar los alquileres de los videojuegos, o si también los clientes que los han alquilado y los empleados que han gestionado su alquiler). La vista tampoco incorpora todos los atributos de las tablas con restricción NOT NULL, por lo tanto, existirán operaciones de cambio (por ejemplo, inserciones) que serán imposibles de propagar desde la vista hacia las tablas implicadas en la definición de la vista.

7. DESTRUCCIÓN DE LA BD DE VIDEOJUEGOS

Finalmente, si se desea hacer limpieza de todo, procedemos a la destrucción de la BD, ejecutando para ello las siguientes sentencias:

```
-- Delete the data and the SCHEMA

DROP SCHEMA videogames CASCADE;

SET search_path TO "$user", public;
```