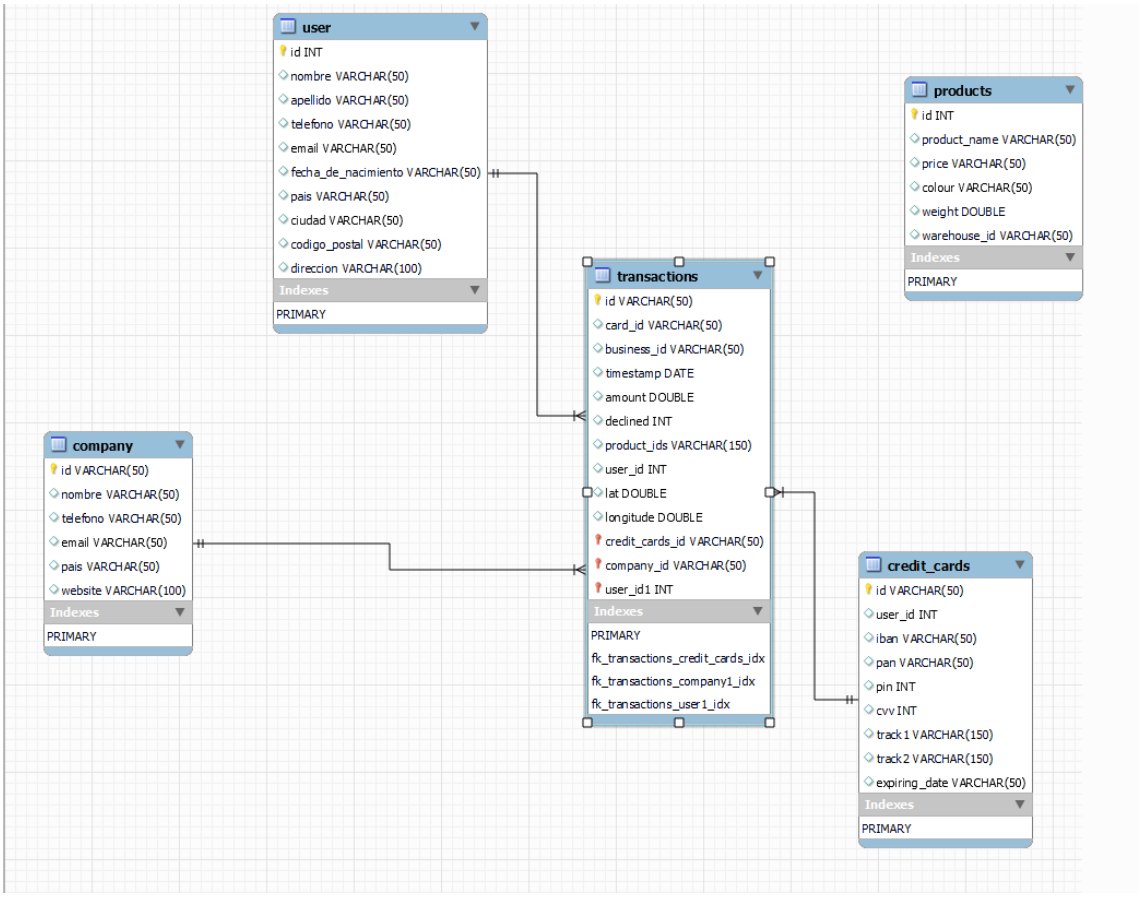
Tarea S4.01. Modelaje SQL

Nivel 1

Se crea el esquema de trabajo SP4, y luego se procede a crear la tabla user teniendo en cuenta la distribución de los datos en los archivos csv "users_ca", "users_uk" y "users_usa" manteniendo la característica del campo id que es la PRIMARY KEY del mismo modo se crean las tablas restantes dentro del modelo importando la data desde los archivos csv.



En este momento la tabla “Products” aún no ha sido integrada al esquema pues en la tabla “transactions” el identificador de productos relacionado a las transacciones agrupa en un mismo campo los productos vendidos impidiendo su relación directa con la tabla “Products”

Tarea S4.01. Modelaje SQL

Ejercicio 1

Realiza una subconsulta que muestre a todos los usuarios con más de 30 transacciones utilizando al menos 2 tablas. En la imagen se muestra el script se realiza una consulta (no Subconsulta) que muestra los resultados de manera efectiva.

```
143 •  SELECT u.id AS UserId, concat(u.nombre, ' ', u.apellido) AS User, count(t.id) AS No_operaciones
144    FROM user u
145    JOIN transactions t ON u.id = t.user_id
146    GROUP BY UserId
147    HAVING No_operaciones > 30;
148
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

UserId	User	No_operaciones
92	Lynn Riddle	39
267	Ocean Nelson	52
272	Hedwig Gilbert	76
275	Kenyon Hartman	48

Ejercicio 2

Muestra el promedio de la suma de transacciones por IBAN de las tarjetas de crédito en la compañía “Donec Ltd.” utilizando al menos 2 tablas.

```
148
149    # Ejercicio 2
150
151 •  SELECT format(AVG(t.amount), 2) AS Media_de_gasto, cc.iban AS Iban
152    FROM credit_cards cc
153    JOIN transactions t ON t.card_id = cc.id
154    JOIN company c ON t.business_id = c.id
155    WHERE c.nombre = "Donec Ltd"
156    GROUP BY Iban;
157
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Media_de_gasto	Iban
203.72	PT87806228135092429456346

Tarea S4.01. Modelaje SQL

Ejercicio

Nivel 2

Este script de MySQL crea una tabla llamada Status basada en una consulta que determina el estado de activación de las tarjetas de crédito según las últimas tres transacciones.

Subconsulta Interna:

La subconsulta interna está anidada dentro de otra consulta. Esta subconsulta se encarga de seleccionar las últimas tres transacciones para cada tarjeta de crédito:

La consulta selecciona las columnas “card_id”, “declined”, y “timestamp” de la tabla “transactions”, utilizando las variables de usuario “@rown” y “@target” para llevar un seguimiento del número de transacciones por tarjeta de crédito.

Ordena las filas por “card_id”, “timestamp” en orden descendente y “declined”, numerando las filas para cada tarjeta de crédito según la fecha de cada operación y muestra si la transacción fue rechazada, con un límite de 3 transacciones por tarjeta.

Consulta Externa:

La consulta externa se aplica a los resultados de la subconsulta interna.

Agrupa los resultados por “card_id”, y utilizando una función de agregación (SUM) calcula la suma de los valores de “declined” (0 aceptada, 1 rechazada) para cada tarjeta de crédito.

Luego, utiliza una expresión CASE para determinar el estado de la tarjeta de crédito:

Si la suma de “declined” para las últimas tres transacciones es igual a 3, entonces la tarjeta se considera "Inactiva".

De lo contrario, se considera "Activa".

El resultado de esta evaluación se almacena en una columna llamada Status.

Esta tabla se integra al esquema teniendo en cuenta que el campo “Card_id” contiene valores únicos que pueden relacionarse con la tabla “transactions” referenciado al campo del mismo nombre el cual es una FOREIGN KEY.

```sql
#Ejercicio Nivel 2

CREATE TABLE Status (
SELECT last3.Card_id, (
CASE
    WHEN sum(last3.declined) = 3 THEN 'Inactiva'
    ELSE 'Activa'
END) as Status
FROM (SELECT card_id, declined, timestamp
 FROM (SELECT t.declined, t.card_id, t.timestamp,
     @rown := IF(@target = t.card_id, @rown + 1, 1) AS rown, @target := t.card_id
        FROM transactions t JOIN (SELECT @target = NULL, @rown = 0)
AS Bucle ORDER BY t.card_id, t.timestamp DESC, t.declined ) AS T1 WHERE rown <= 3) AS last3
GROUP BY card_id);
```

Tarea S4.01. Modelaje SQL

```
1 • SELECT * FROM sp4.status;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Card_id	Status
CcU-2938	Activa
CcU-2945	Activa
CcU-2952	Activa
CcU-2959	Activa
CcU-2966	Activa
CcU-2973	Activa
CcU-2980	Activa
CcU-2987	Activa
CcU-2994	Activa
CcU-3001	Activa
CcU-3008	Activa
CcU-3015	Activa
CcU-3022	Activa
CcU-3029	Activa
CcU-3036	Activa
CcU-3043	Activa
CcU-3050	Activa
CcU-3057	Activa
CcU-3064	Activa
CcU-3071	Activa
CcU-3078	Activa
CcU-3085	Activa
CcU-3092	Activa
CcU-3099	Activa
CcU-3106	Activa
CcU-3113	Activa
CcU-3120	Activa
CcU-3127	Activa
CcU-3134	Activa

status 2 ×

Output

Action Output

#	Time	Action	Message
1	11:23:36	SELECT * FROM sp4.status LIMIT 0, 5000	275 row(s) returned

```
173 • SELECT count(status)
174   FROM status
175   WHERE status = "Activa";
176
177   #Ejercicio Nv3
```

Result Grid | Filter Rows: | Export: | Wrap Cell

count(status)
275

Tarea S4.01. Modelaje SQL

Ejercicio

Nivel 3

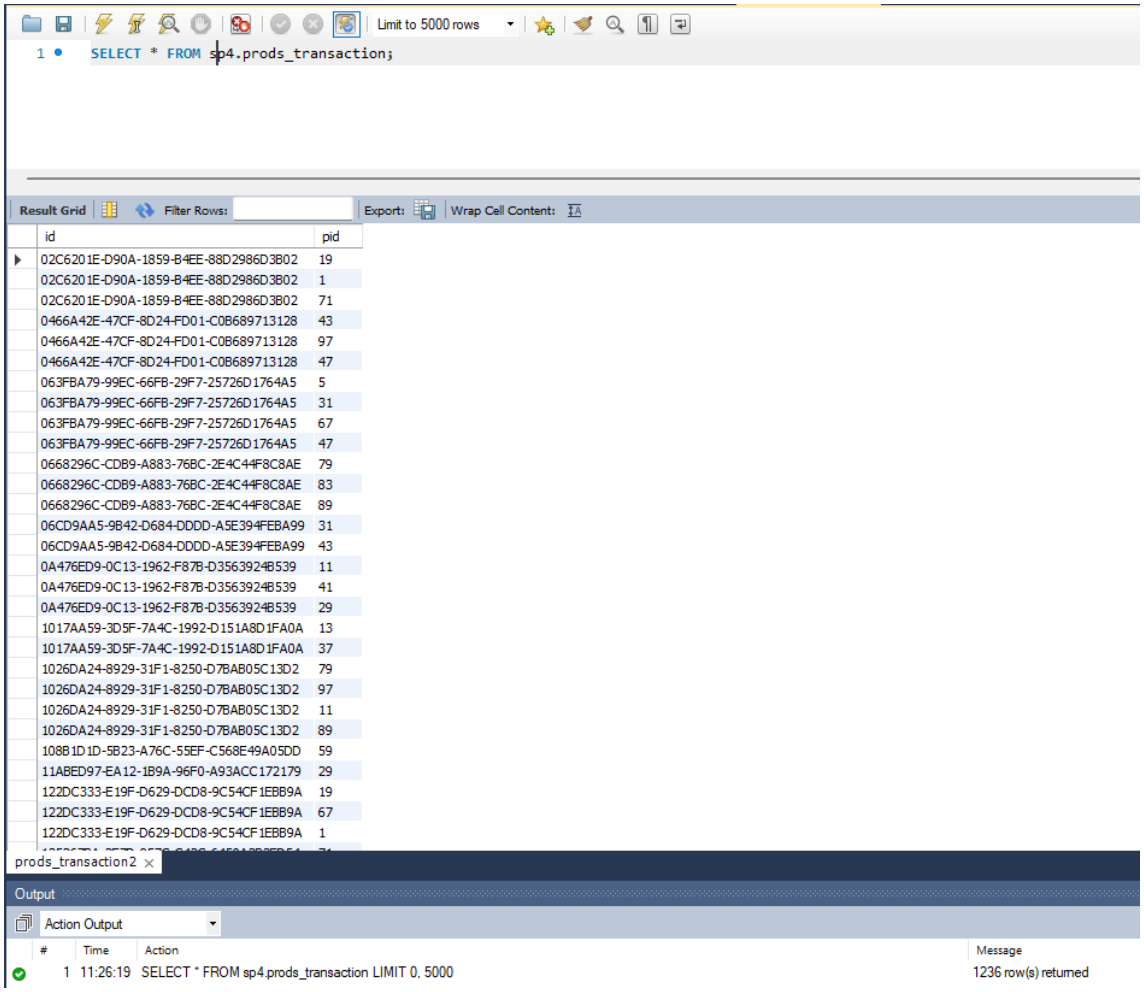
Se crea una nueva tabla denominada “Prods_Transaction”, en esta tabla se han separado los id de producto que estaban agrupados en un solo campo, de forma que en esta nueva tabla se muestra el id de la transacción y el producto vendido uno por uno. Valiéndome de la instrucción SUBSTRING_INDEX, y declarando una variable de usuario que determina el numero de productos vendidos en cada transacción, he logrado separar estos productos a fin de normalizar los datos y poder relacionar la nueva tabla creada con el esquema.

```
78
79  CREATE TABLE Prods_Transaction (
80  SELECT
81      t.id AS id,
82      CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_ids, ',', n.n), ',', -1) AS UNSIGNED) AS pid
83  FROM
84      transactions t
85  CROSS JOIN
86      (SELECT @prods := @prods + 1 AS n
87       FROM (SELECT @prods := 0) r
88       CROSS JOIN transactions
89       WHERE declined = 0) n
90  WHERE
91      n.n <= LENGTH(t.product_ids) - LENGTH(REPLACE(t.product_ids, ',', '')) + 1 AND t.declined = 0);
92
93  ALTER TABLE Prods_Transaction MODIFY pid int,
94  ADD FOREIGN KEY (id) REFERENCES transactions(id),
95  ADD FOREIGN KEY (pid) REFERENCES products(id);
96
```

Luego se han agregado las FOREIGN KEYS, a fin de relacionar la nueva tabla con el esquema.

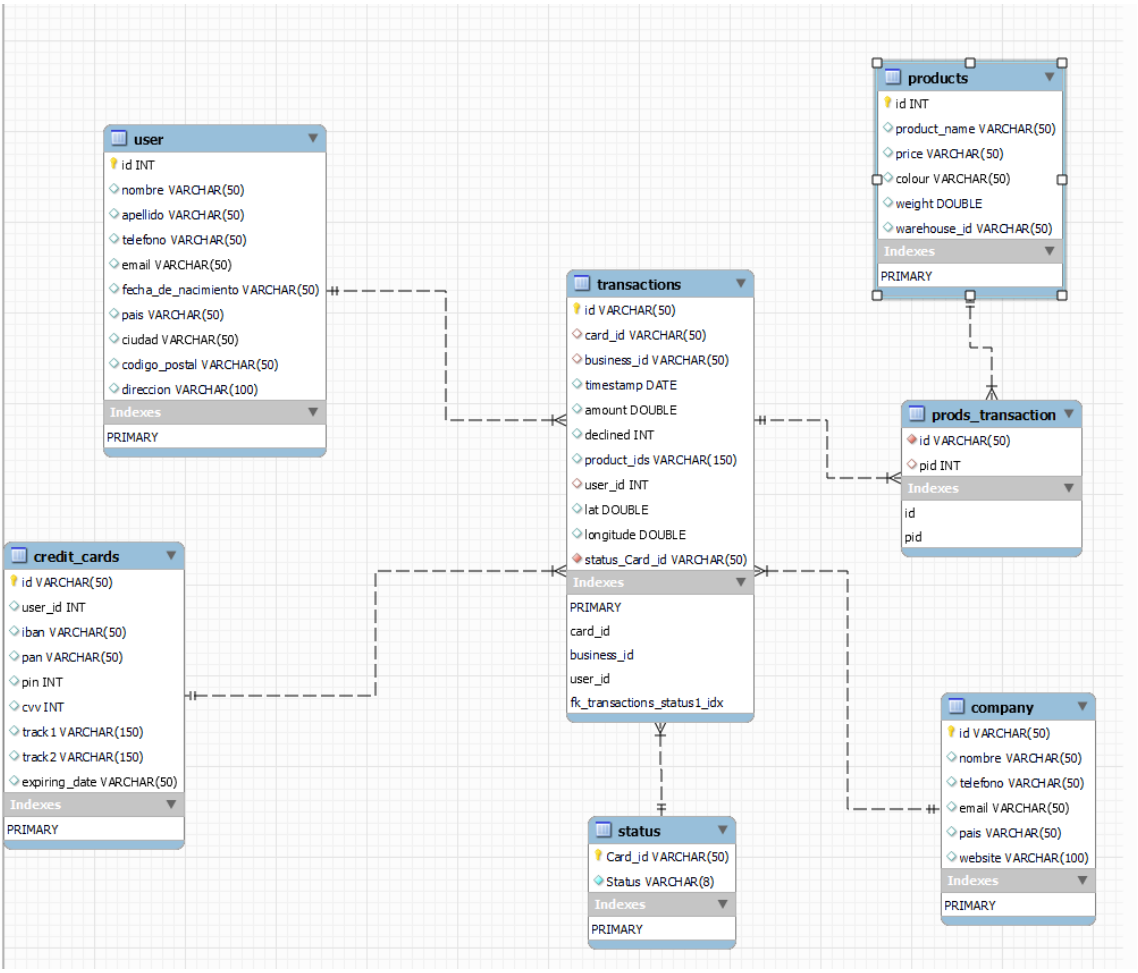
Vista de la nueva tabla “Prods_Transaction”. Se han tenido en cuenta solo las operaciones efectivamente realizadas.

Tarea S4.01. Modelaje SQL



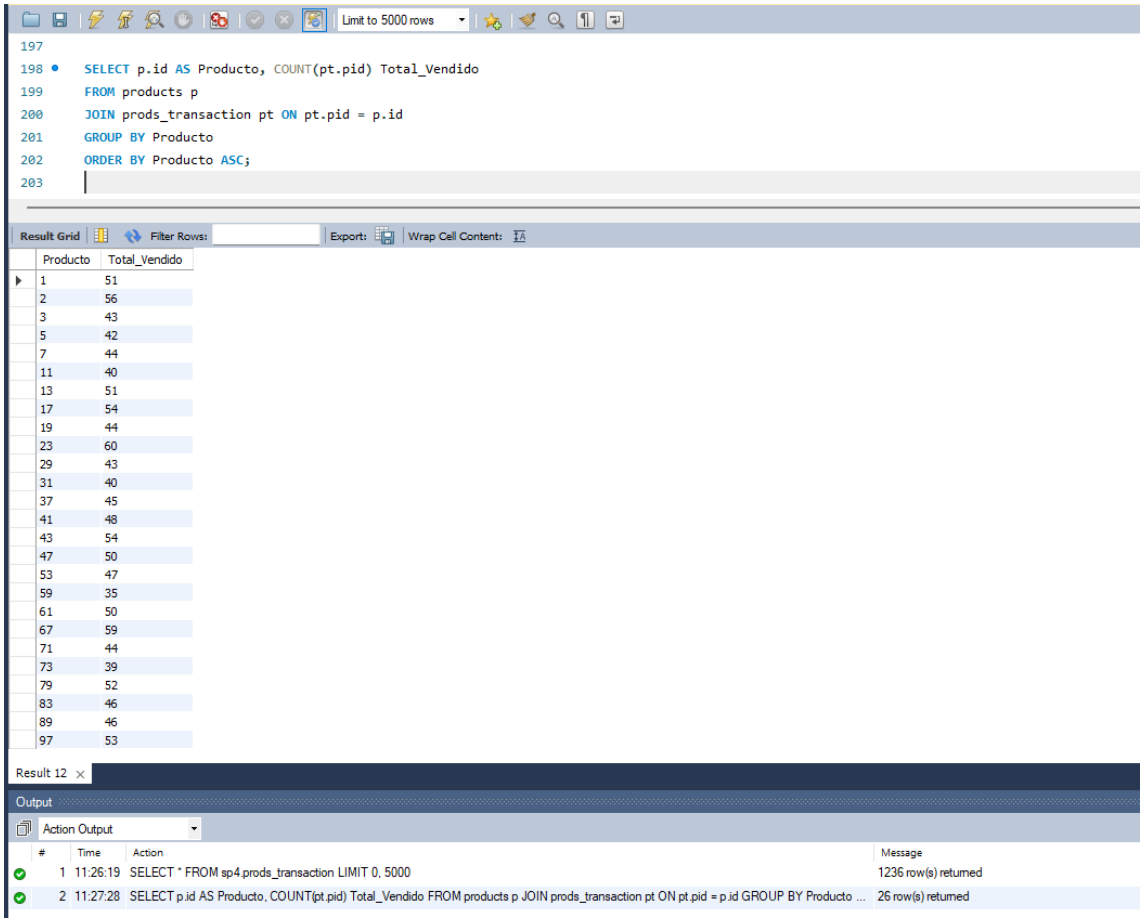
La nueva tabla contiene un campo “id” FOREIGN KEY referenciado al campo “id” de la Tabla “transactions” (PRIMARY KEY), contiene así mismo un campo “pid” FOREIGN KEY referenciado al campo “id” de la Tabla “products” (PRIMARY KEY), permitiendo de esta manera la integración de la tabla “products” al modelo:

Tarea S4.01. Modelaje SQL



Finalmente se realiza la consulta según se ha requerido en el ejercicio mostrándose el total de productos vendidos, según su id.

Tarea S4.01. Modelaje SQL



```
197
198 •  SELECT p.id AS Producto, COUNT(pt.pid) Total_Vendido
199    FROM products p
200    JOIN prods_transaction pt ON pt.pid = p.id
201    GROUP BY Producto
202    ORDER BY Producto ASC;
203
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Producto	Total_Vendido
1	51
2	56
3	43
5	42
7	44
11	40
13	51
17	54
19	44
23	60
29	43
31	40
37	45
41	48
43	54
47	50
53	47
59	35
61	50
67	59
71	44
73	39
79	52
83	46
89	46
97	53

Result 12

Output

Action Output

#	Time	Action	Message
1	11:26:19	SELECT * FROM sp4.prods_transaction LIMIT 0, 5000	1236 row(s) returned
2	11:27:28	SELECT p.id AS Producto, COUNT(pt.pid) Total_Vendido FROM products p JOIN prods_transaction pt ON pt.pid = p.id GROUP BY Producto ...	26 row(s) returned