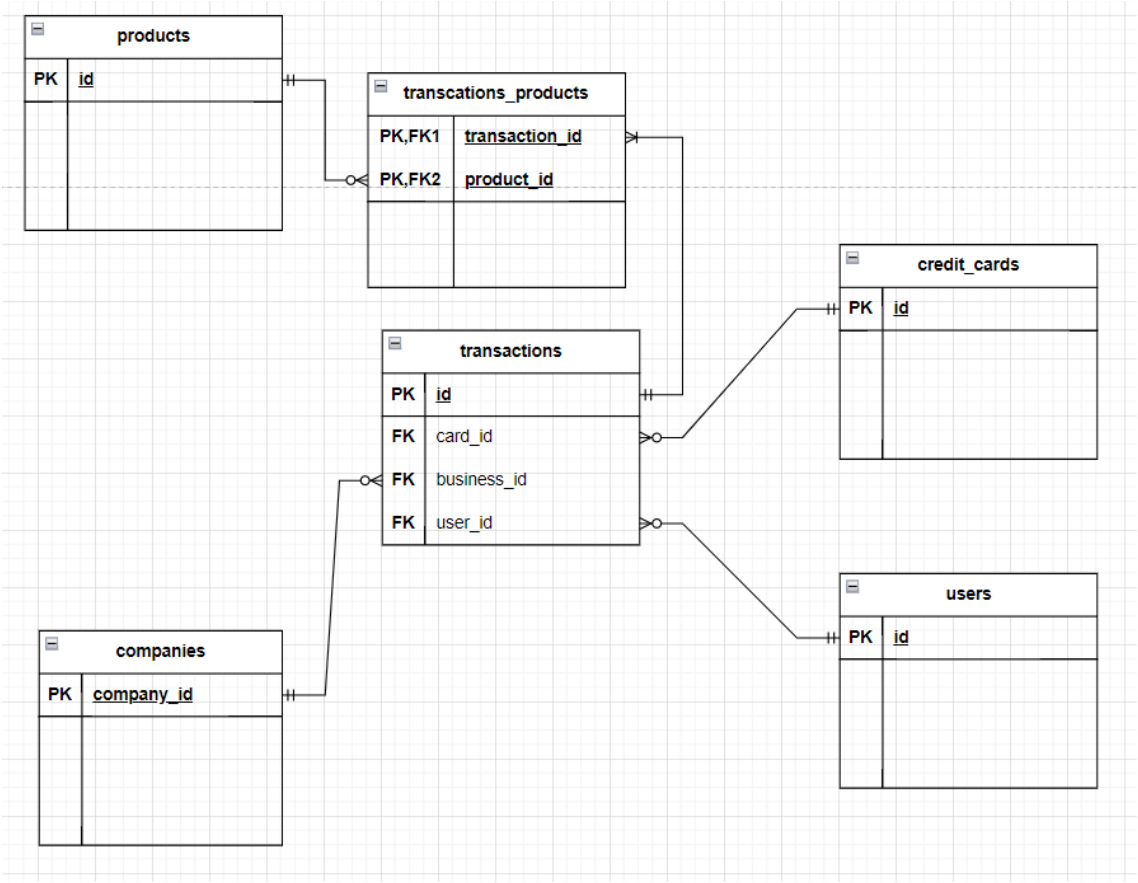
Nivell 1

Descàrrega els arxius CSV, estudia'ls i dissenya una base de dades amb un esquema d'estrella que contingui, almenys 4 taules de les quals puguis realitzar les següents consultes:

A continuación, se muestra el diagrama entidad-relación que realicé antes de importar los archivos a SQL:



Se trata de un modelo de estrella con la tabla de hechos **transactions** y cuatro tablas de dimensión: **companies**, **credit_cards, users** y **products**.

Debido a que products y transactions tienen una relación de muchos a muchos (en una transacción se pueden comprar muchos productos, y un producto puede comprarse en muchas transacciones), he creado la tabla de unión **transactions_products** para transformar la relación de muchos a muchos en dos relaciones de 1 a muchos.

Cabe notar también que las tres tablas de usuarios las unificaremos en una sola, **users**, ya que las tres tienen la misma estructura y simplemente varía el país de origen del usuario.

A continuación, creamos la database y la tabla **transactions**:

```
#Creamos la database

CREATE DATABASE sprint_4;

USE sprint_4;

CREATE TABLE transactions(
    id VARCHAR(255) PRIMARY KEY,
    card_id VARCHAR(15),
    business_id VARCHAR(15),
    timestamp TIMESTAMP,
    amount DECIMAL (10,2),
    declined TINYINT (1),
    product_ids VARCHAR(50),
    user_id INT,
    lat FLOAT,
    longitude FLOAT
    );
```

Creamos la tabla **products**:

```
CREATE TABLE products (
    id INT PRIMARY KEY,
    product_name VARCHAR(255),
    price DECIMAL (10,2),
    colour VARCHAR(15),
    weight FLOAT,
    warehouse_id VARCHAR(15)
    );
```

Creamos la tabla **users**:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    name VARCHAR (50),
    surname VARCHAR (50),
    phone VARCHAR (150),
    email VARCHAR (150),
    birth_date VARCHAR(50),
    country VARCHAR(50),
    city VARCHAR(50),
    postal_code VARCHAR(50),
    address VARCHAR(150)
    );
```

Creamos la tabla **credit_cards**:

```
CREATE TABLE credit_cards (
    id varchar(45) NOT NULL,
    user_id INT,
    iban varchar(45),
    pan varchar(45),
    pin char(4) DEFAULT NULL,
    cvv char(3) DEFAULT NULL,
    track1 varchar (150),
    track2 varchar(150),
    expiring_date date,
    PRIMARY KEY (id)
);
```

Creamos la tabla **companies**:

```
CREATE TABLE companies (
    company_id VARCHAR(45) PRIMARY KEY,
    company_name VARCHAR(255),
    phone VARCHAR(15),
    email VARCHAR(100),
    country VARCHAR(100),
    website VARCHAR(255)
    );
```

Creamos la tabla de unión **transactions_products**:

```
CREATE TABLE transactions_products (
    transaction_id VARCHAR(255),
    product_id INT,
    PRIMARY KEY (transaction_id, product_id),
    FOREIGN KEY (transaction_id) REFERENCES transactions(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
    );
```

Luego, insertamos nuestros datos en las diferentes tablas:

```
#Importamos a la tabla transactions
LOAD DATA
INFILE 'transactions.csv'
INTO TABLE transactions
FIELDS TERMINATED BY ';'
IGNORE 1 ROWS;
```

```
#Importamos a la tabla products
LOAD DATA
INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv'
INTO TABLE products
FIELDS TERMINATED BY ','
IGNORE 1 ROWS;
```

```
#Importamos a la tabla users
LOAD DATA
INFILE 'users_ca.csv'
INTO TABLE users
FIELDS TERMINATED BY ","
ENCLOSED BY '"'
LINES TERMINATED BY "\r\n"
IGNORE 1 ROWS;

LOAD DATA
INFILE 'users_uk.csv'
INTO TABLE users
FIELDS TERMINATED BY ","
ENCLOSED BY '"'
LINES TERMINATED BY "\r\n"
IGNORE 1 ROWS;

LOAD DATA
INFILE 'users_usa.csv'
INTO TABLE users
FIELDS TERMINATED BY ","
ENCLOSED BY '"'
LINES TERMINATED BY "\r\n"
IGNORE 1 ROWS;
```

```
#Importamos a la tabla credit_cards
LOAD DATA
INFILE 'credit_cards.csv'
INTO TABLE credit_cards
FIELDS TERMINATED BY ","
IGNORE 1 ROWS;
```

```
#Importamos a la tabla companies
LOAD DATA
INFILE 'companies.csv'
INTO TABLE companies
FIELDS TERMINATED BY ","
IGNORE 1 ROWS;
```

Por último, importamos los datos a la tabla de unión. Previamente hemos tenido que separar las filas ya que en las tablas originales una misma transacción tenía más de un product_id en la misma fila. Para poder importar los datos a esta tabla correctamente, hemos separado cada product_id y transaction_id de manera única, es decir, si una misma transacción tiene dos productos, esa transacción aparecerá en dos filas, una con cada producto.

```
135 •  LOAD DATA
136    INFILE 'transactions_products.csv'
137    INTO TABLE transactions_products
138    FIELDS TERMINATED BY ";"
139    IGNORE 1 ROWS;
140
141 •  SELECT * FROM transactions_products;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

transaction_id	product_id
02C6201E-D90A-1859-B4EE-88D2986D3B02	1
02C6201E-D90A-1859-B4EE-88D2986D3B02	19
02C6201E-D90A-1859-B4EE-88D2986D3B02	71
0466A42E-47CF-8D24-FD01-C0B689713128	43
0466A42E-47CF-8D24-FD01-C0B689713128	47
0466A42E-47CF-8D24-FD01-C0B689713128	97
063FBA79-99EC-66FB-29F7-25726D1764A5	5
063FBA79-99EC-66FB-29F7-25726D1764A5	31
063FBA79-99EC-66FB-29F7-25726D1764A5	47
063FBA79-99EC-66FB-29F7-25726D1764A5	67
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	79
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	83
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	89
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	31
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	43
07A46D48-31A3-7E87-65B9-0DA902AD109F	23
07A46D48-31A3-7E87-65B9-0DA902AD109F	47

transactions_products 6

Output

Action Output

#	Time	Action	Message
15	11:45:12	SELECT * FROM transactions_products LIMIT 0, 1000	1000 row(s) returned
16	11:45:23	SELECT * FROM transactions_products LIMIT 0, 2000	1457 row(s) returned

A continuación, creamos las foreing keys en la tabla transactions, para que pueda relacionarse correctamente con las demás tablas:

```sql
ALTER TABLE transactions
ADD CONSTRAINT
FOREIGN KEY (card_id) REFERENCES credit_cards(id);
```

```sql
ALTER TABLE transactions
ADD CONSTRAINT
FOREIGN KEY (business_id) REFERENCES companies(company_id);
```

```sql
ALTER TABLE transactions
ADD CONSTRAINT
FOREIGN KEY (user_id) REFERENCES users(id);
```

Y como ya no vamos a necesitar la columna “product_ids” en la tabla transactions, la eliminamos:

```sql
142  ALTER TABLE transactions
143  DROP COLUMN product_ids;
144
145  SHOW COLUMNS FROM transactions;
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PRI	NULL	
card_id	varchar(15)	YES	MUL	NULL	
business_id	varchar(15)	YES	MUL	NULL	
timestamp	timestamp	YES		NULL	
amount	decimal(10,2)	YES		NULL	
declined	tinyint(1)	YES		NULL	
user_id	int	YES	MUL	NULL	
lat	float	YES		NULL	
longitude	float	YES		NULL	

Exercici 1: Realitza una subconsulta que mostri tots els usuaris amb més de 30 transaccions utilitzant almenys 2 taules.

```sql
SELECT CONCAT(name, " ", surname) AS full_name
FROM users
WHERE id IN (
    SELECT user_id
    FROM transactions
    GROUP BY user_id
    HAVING COUNT(*) > 30);
```

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

full_name
Lynn Riddle
Ocean Nelson
Hedwig Gilbert
Kenyon Hartman

Los 4 usuarios que realizaron más de 30 transacciones se detallan aquí arriba.

Exercici 2: Mostra la mitjana de la suma de transaccions per IBAN de les targetes de crèdit en la companyia Donec Ltd utilitzant almenys 2 taules.

En primer lugar, cabe destacar que la empresa Donec Ltd solo tiene un IBAN, como se puede apreciar aquí:

```sql
SELECT DISTINCT cc.iban
FROM credit_cards cc
JOIN transactions t
    ON cc.id = t.card_id
JOIN companies co
    ON co.company_id = t.business_id
WHERE company_name = "Donec Ltd";
```

Result Grid | Filter Rows: | Export:

iban
PT87806228135092429456346

Por ende, no sería necesario agrupar por IBAN para saber la media de gastos de esta empresa:

```sql
SELECT
    ROUND(AVG(t.amount),2) AS avg_amount
FROM transactions t
JOIN companies co
    ON co.company_id = t.business_id
WHERE company_name = "Donec Ltd";
```

avg_amount
203.72

La media de las transacciones de la empresa Donec Ltd es de 203,72 €.

Nivell 2

Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les últimes tres transaccions van ser declinades.

En primer lugar, creamos la tabla **active_credit_cards**, donde tendremos dos columnas: el id y el estado de la tarjeta.

```sql
CREATE TABLE active_credit_cards (
    id VARCHAR(45),
    status VARCHAR(45),
    PRIMARY KEY (id),
    FOREIGN KEY (id) references credit_cards(id)
    );
```

A continuación, creamos una consulta para seleccionar cada card_id y su status.
El card_id se considerará inactive si:

- Las últimas tres transacciones fueron rechazadas
- Solo tiene dos transacciones, y ambas fueron rechazadas
- Solo tiene una transacción y fue rechazada

```sql
-- creamos una primera CTE table para saber cuál es el número de transacciones de cada card_id
WITH count_transactions AS (
    SELECT card_id, COUNT(*) AS numTransactions
    FROM transactions
    GROUP BY card_id),
-- creamos otra CTE table donde generamos un ranking para cada card_id, ordenando por timestamp para saber cuáles fueron las últimas transacciones;
-- además, hacemos un conteo acumulado (running total) de la columna "declined" que utilizaremos en el CASE statement
general_ranking AS (
    SELECT *,
        ROW_NUMBER() OVER(PARTITION BY card_id ORDER BY TIMESTAMP DESC) AS ranking,
        SUM(declined) OVER(PARTITION BY card_id ORDER BY TIMESTAMP DESC) AS running_total
    FROM transactions)


SELECT c.card_id,
    CASE WHEN
    -- si el id tiene 3 o más transacciones, y en el ranking 3 tenemos un total acumulado de 3, entonces las últimas 3 transacciones fueron declined
    (c.numTransactions >= 3 AND g.ranking = 3 AND g.running_total = 3) THEN "inactive"
    WHEN
    -- si el id tiene 2 transacciones, y en el ranking 2 tenemos un total acumulado de 2, entonces ambas transacciones fueron rechazadas
    (c.numTransactions = 2 AND g.ranking = 2 AND g.running_total = 2) THEN "inactive"
    WHEN
    -- si el id tiene 1 transacción, y el total acumulado es 1, la transacción fue rechazada
    (c.numTransactions = 1 AND g.running_total = 1) THEN "inactive"
    ELSE "active"
    END AS status
FROM count_transactions c
JOIN general_ranking g
    ON c.card_id = g.card_id
WHERE g.ranking = 3 -- de los IDs con 3 o más transacciones, solo queremos el ranking "3" de cada id
OR (g.ranking = 2 AND c.numTransactions = 2) -- de los IDs con 2 transacciones, solo queremos el ranking "2" de cada id
OR (c.numTransactions = 1) -- IDs con 1 transacción
;
```

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

card_id	status
CcU-2938	active
CcU-2945	active
CcU-2952	active
CcU-2959	active
CcU-2966	active

Result 30

Output

Action Output

#	Time	Action	Message
39	10:49:40	WITH count_transactions AS (SELECT card_id, COUNT(*) AS numTransactions FROM transactions GRO...	275 row(s) returned

Como podemos ver, la consulta arroja 275 resultados, que corresponde con el número de tarjetas registradas en la base de datos, y a su lado el estado en el que están.

A continuación, queremos ingresar los resultados de esta consulta a la tabla active_credit_cards. Para esto, simplemente incluimos la consulta anterior dentro de un **INSERT INTO** active_credit_cards:

```
INSERT INTO active_credit_cards
-- creamos una primera CTE table para saber cuál es el número de transacciones de cada card_id
WITH count_transactions AS (
    SELECT card_id, COUNT(*) AS numTransactions
    FROM transactions
    GROUP BY card_id),
-- creamos otra CTE table donde generamos un ranking para cada card_id, ordenando por timestamp para saber cuáles fueron las úl
-- además, hacemos un conteo acumulado (running total) de la columna "declined" que utilizaremos en el CASE statement
general_ranking AS (
    SELECT *,
        ROW_NUMBER() OVER(PARTITION BY card_id ORDER BY TIMESTAMP DESC) AS ranking,
```

Al seleccionar los datos de la tabla active_credit_cards, vemos que se han ingresado correctamente:

```
238 •  SELECT * FROM active_credit_cards;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	status
CcU-2938	active
CcU-2945	active
CcU-2952	active
CcU-2959	active
CcU-2966	active

active_credit_cards 32

Output

Action Output

#	Time	Action	Message
41	10:56:52	SELECT * FROM active_credit_cards LIMIT 0, 1000	275 row(s) returned

Exercici 1: Quantes targetes estan actives?

```sql
SELECT COUNT(*) as active_cards
FROM active_credit_cards
WHERE status = "active";
```

Result Grid | Filter Rows: | Expor

active_cards
275

La totalidad de las tarjetas registradas, que son 275, se encuentran activas.

Nivell 3

Crea una taula amb la qual puguem unir les dades del nou arxiu products.csv amb la base de dades creada, tenint en compte que des de transaction tens product_ids. Genera la següent consulta:

Exercici 1: Necessitem conèixer el nombre de vegades que s'ha venut cada producte.

```sql
SELECT
    p.id AS product_id,
    p.product_name,
    p.price,
    p.colour,
    COUNT(tp.transaction_id) AS sales
FROM products p
LEFT JOIN transactions_products tp -- usamos LEFT JOIN para que aquellos productos que no se vendieron aparezcan en los resultados con un "0"
    ON p.id = tp.product_id
GROUP BY p.id, p.product_name, p.price, p.colour
ORDER BY sales DESC, p.price DESC;
```

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

product_id	product_name	price	colour	sales
67	Winterfell	195.94	#1c1c1c	68
23	riverlands north	169.96	#545454	68
79	Direwolf riverlands the	132.86	#b2b2b2	66
43	duel	59.80	#5b5b5b	65
2	Tarly Stark	9.24	#919191	65

Result 3

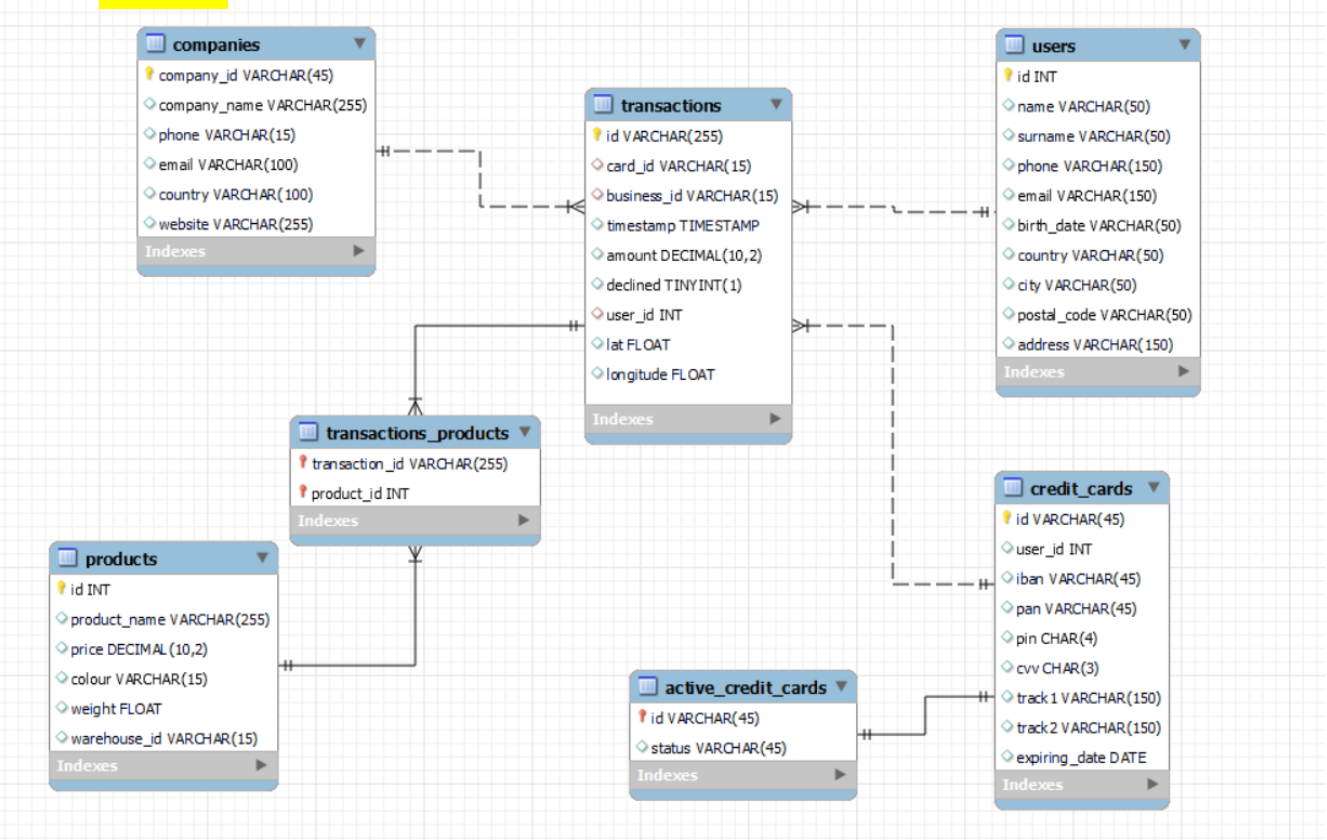
Output

Action Output

#	Time	Action	Message
9	11:32:03	SELECT p.id AS product_id, p.product_name, p.price, p.colour, COUNT(tp.transaction_id) AS sales...	100 row(s) returned
10	11:32:07	SELECT p.id AS product_id, p.product_name, p.price, p.colour, COUNT(tp.transaction_id) AS sales...	100 row(s) returned

Aquí arriba se muestra la lista de productos y la cantidad de veces que se ha vendido cada uno. Están ordenados de manera descendente primero por cantidad de ventas, y segundo por precio.

A continuación, mostramos como ha quedado el esquema final de nuestra base de datos:



Hemos de notar que en las tablas users y credit_cards existe el mismo campo: **user_id.** Tras corroborar si ambos campos contienen los mismos valores, hemos descubierto que no:

```sql
SELECT u.id AS users_userID, t.user_id AS transactions_userID, c.user_id AS creditcards_userID
FROM users u
JOIN transactions t
    ON u.id = t.user_id
JOIN credit_cards c
    ON c.id = t.card_id;
```

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

users_userID	transactions_userID	creditcards_userID
92	92	275
275	275	275
92	92	275
92	92	275
87	87	275
92	92	275
82	82	275
89	89	275
92	92	275
92	92	275
85	85	275
92	92	275
80	80	275
92	92	275
83	83	275
92	92	275
91	91	275

Result 7

Output

Action Output

#	Time	Action	Message
11	11:37:51	SELECT u.id AS users_userID, t.user_id AS transactions_userID, c.user_id AS creditcard_userID FROM users...	587 row(s) returned
12	11:37:57	SELECT u.id AS users_userID, t.user_id AS transactions_userID, c.user_id AS creditcards_userID FROM user...	587 row(s) returned

user_id de la tabla **users** coincide siempre con el user_id de la tabla **transactions**, pero el user_id de la tabla **credit_cards** es diferente.

Por este motivo, en vez de eliminar el campo de la tabla credit_cards, le cambiaremos el nombre a pedido del cliente:

```
273 •   ALTER TABLE credit_cards
274     RENAME COLUMN user_id TO userID_NeedsReview;
275
276 •   SHOW COLUMNS FROM credit_cards;
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(45)	NO	PRI	NULL	
userID_NeedsReview	int	YES		NULL	
iban	varchar(45)	YES		NULL	
pan	varchar(45)	YES		NULL	
pin	char(4)	YES		NULL	
cvv	char(3)	YES		NULL	
track1	varchar(150)	YES		NULL	
track2	varchar(150)	YES		NULL	
expiring_date	date	YES		NULL	