SPRINT 4

Creació de base de dades

Juanjo MONTERO

NIVELL 1 – Exercici 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:

La base de dades es dissenya en **estrella**, amb la taulta *transactions* al centre, contenint les **claus foràanies** *card_id*, *business_id* i *user_id* que referencien a les claus primàries de la taula *user*, *companies* i *credit_card* respectivament.

1	N
users.id	*transactions.card_id*
companies.company_id	*transactions.business_id*
credit_card.id	*transactions.card_id*

Per tots els camps del tipus VARCHAR s'especifica un número de caràcters ajustat a les longituds dels valors proporcionats als CSVs i tenint en compte possibles entrades de dades futures.

```sql
CREATE TABLE IF NOT EXISTS users (
    id INT,
    name VARCHAR(50),
    surname VARCHAR(50),
    phone VARCHAR(25),
    email VARCHAR(70),
    birth_date VARCHAR(30),
    country VARCHAR(50),
    city VARCHAR(50),
    postal_code VARCHAR(50),
    address VARCHAR(100),
    PRIMARY KEY (id)
);
```

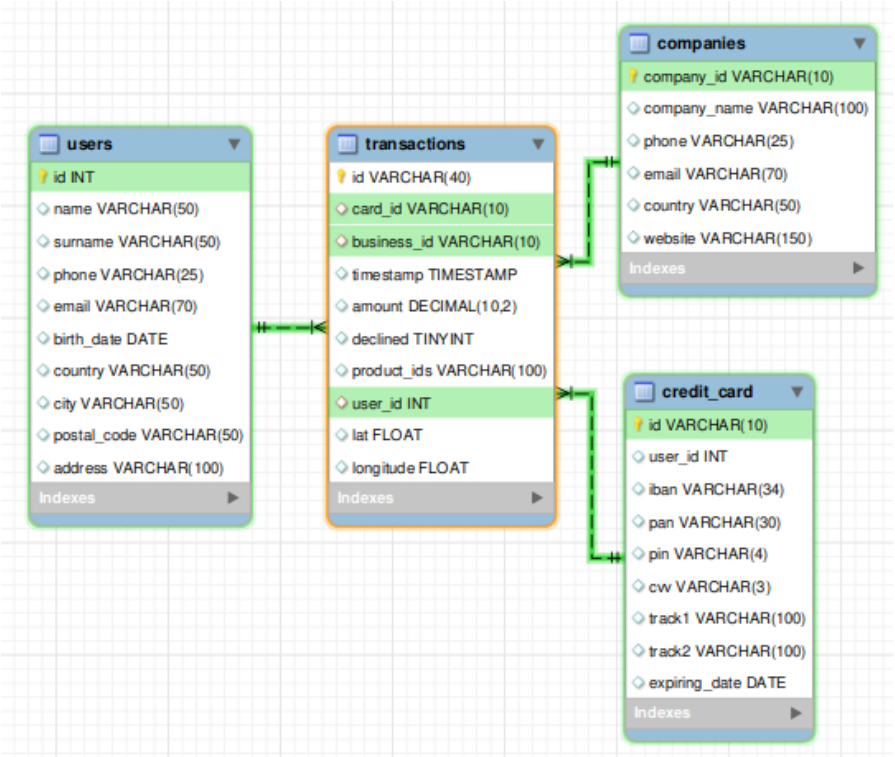
```sql
CREATE TABLE IF NOT EXISTS companies (
    company_id VARCHAR(10),
    company_name VARCHAR(100),
    phone VARCHAR(25),
    email VARCHAR(70),
    country VARCHAR(50),
    website VARCHAR(150),
    PRIMARY KEY (company_id)
);
```

```sql
CREATE TABLE IF NOT EXISTS credit_card (
    id VARCHAR(10),
    user_id INT,
    iban VARCHAR(34),
    pan VARCHAR(30),
    pin VARCHAR(4),
    cvv VARCHAR(3),
    track1 VARCHAR(100),
    track2 VARCHAR(100),
    expiring_date VARCHAR(20),
    PRIMARY KEY (id)
);
```

```sql
CREATE TABLE IF NOT EXISTS transactions (
    id VARCHAR(40),
    card_id VARCHAR(10),
    business_id VARCHAR(10),
    timestamp TIMESTAMP,
    amount DECIMAL(10,2),
    declined TINYINT,
    product_ids VARCHAR(100),
    user_id INT,
    lat FLOAT,
    longitude FLOAT,
    PRIMARY KEY (id),
    FOREIGN KEY (card_id) REFERENCES credit_card(id),
    FOREIGN KEY (business_id) REFERENCES companies(company_id),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

A més a més, mitjançant la funció STR_TO_DATE es modifiquen *users.birth_date* i *credit_card.expiring_date* per tal que compleixin amb el **format DATE** (p.e. 2025-12-31) i es canvien les columnes a aquest tipus de dada.

D'aquesta manera es poden comparar/ordenar correctament les dades si fos necessari en el futur. El diagrama ER de la base de dades queda de la següent manera:



Per carregar les dades des dels arxius CSV descarregats, es fa servir l'ordre **LOAD DATA LOCAL INFILE**. Per permetre aquesta entrada, primer s'ha d'autoritzar canviant el valor de la variable global *local_infile* introduïnt **SET GLOBAL local_infile = 1**.

Les ordres LOAD DATA tenen la següent estructura:

```
LOAD DATA LOCAL INFILE "/home/itacademy/csv/file.csv" -- fitxer origen de dades
INTO TABLE my_table -- taula destí
FIELDS TERMINATED BY ',' -- separador entre valors
ENCLOSED BY '"' -- delimitador de cadenes de caracters
LINES TERMINATED BY '\r\n' -- tipus de salt de línia
IGNORE 1 LINES; -- ignora la línia amb els títols de columna
```

S'han d'especificar la **ruta absoluta de l'arxiu** en el sistema local, així com la taula en la que es carreguen les dades. D'igual manera, cal especificar **separadors**, **delimitadors** de cadena i caràcters de **salt de línia** (aquests útlims poden variar en funció del sistema operatiu en el que es va crear el fitxer CSV).

S'ignora la primera línia degut a que en els fitxers donats aquesta correspon al títol de cada una de les columnes.

Aquestes són les ordres per la càrrega de cada un dels csv a l'adjunt *sprint4_script.sql:*

```
-- carreguem les dades des del respectius CSVs
LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/users_ca.csv"
INTO TABLE users -- usuaris de Canada
FIELDS TERMINATED BY ',' -- indica el separador de camp
ENCLOSED BY '"' -- com s'acoten les cadenes de text
LINES TERMINATED BY '\r\n' -- Carriage Return (CR) i Line Feed (LF)
IGNORE 1 LINES; -- ignora la primera línia que conté el nom de columna

LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/users_uk.csv"
INTO TABLE users -- usuaris Regne Unit
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;

LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/users_usa.csv"
INTO TABLE users -- usuaris Estats Units
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;


LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/companies.csv"
INTO TABLE companies
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;

LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/credit_cards.csv"
INTO TABLE credit_card
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n' -- no conté el \r (CARRIAGE RETURN)
IGNORE 1 LINES;

LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/transactions.csv"
INTO TABLE transactions
FIELDS TERMINATED BY ';' -- utilitza punt i coma com a separador
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

Output de la creació de taules i càrrega de dades des del CSVs:

✓	3	10:24:20	SET GLOBAL local_infile = 1	0 row(s) affected
✓	4	10:24:20	CREATE TABLE IF NOT EXISTS users (id INT, name VARCHAR(50), surname VARCHAR(50), ...	0 row(s) affected
✓	5	10:24:20	CREATE TABLE IF NOT EXISTS companies (company_id VARCHAR(10), company_name VARCHAR(1...	0 row(s) affected
✓	6	10:24:20	CREATE TABLE IF NOT EXISTS credit_card (id VARCHAR(10), user_id INT, iban VARCHAR(...	0 row(s) affected
✓	7	10:24:20	CREATE TABLE IF NOT EXISTS transactions (id VARCHAR(40), card_id VARCHAR(10), busi...	0 row(s) affected
✓	8	10:24:20	LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/users_ca.csv" ...	75 row(s) affected Records: 75 Deletec
✓	9	10:24:20	LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/users_uk.csv" ...	50 row(s) affected Records: 50 Deletec
✓	10	10:24:20	LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/users_usa.csv" ...	150 row(s) affected Records: 150 Delete
✓	11	10:24:20	LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/companies.csv"...	100 row(s) affected Records: 100 Delete
✓	12	10:24:20	LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/credit_cards.cs...	275 row(s) affected Records: 275 Delete
✓	13	10:24:20	LOAD DATA LOCAL INFILE "/home/soliton/Desktop/ITAcademy/SPRINTS/03-10 Sprint 4/csv/transactions.cs...	587 row(s) affected Records: 587 Delete

Output de la modificació del tipus de dades de VARCHAR a DATE:

✓	18	10:24:20	ALTER TABLE users MODIFY birth_date DATE	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0
✓	19	10:24:20	ALTER TABLE credit_card MODIFY expiring_date DATE	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0

NIVELL 1 – Exercici 1.b

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

```sql
-- Exercici 1.b Es mostren els usuaris que tenen més de 30 transaccions
SELECT
    users.id,
    users.name,
    users.surname
FROM users
WHERE users.id IN (
                SELECT
                    user_id
                FROM transactions
                WHERE transactions.declined = 0
                GROUP BY transactions.user_id
                HAVING COUNT(transactions.id) > 30
                );
```

#	id	name	surname
1	92	Lynn	Riddle
2	267	Ocean	Nelson
3	272	Hedwig	Gilbert

1 14:10:30 SELECT users.id, users.name, users.surn... 3 row(s) returned

Fent servir la taula *transactions* s'identifiquen els *user_id* amb més de 30 transaccions. Aquests es passen com a filtre a la selecció de la taula *users* i es mostren l'id, nom i cognom dels ususaris que compleixen amb el requeriment.

A efectes d'aquest exercici, no s'han tingut en compte les transaccions rebutjades (és a dir, aquelles amb estat *declined=1*).

NIVELL 1 – Exercici 2

Mostra la mitjana d'amount per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 taules.

```sql
-- NIVELL 1 - Exercici 2 ------------------------------------------------------
-- Preu mitjà de transacció per IBAN de l'empresa Donec Ltd

    SELECT
        companies.company_name AS nom_companyia,
        credit_card.iban AS IBAN,
        ROUND(AVG(transactions.amount),2) AS import_mitja_transaccio_en_euros
    FROM transactions
    JOIN credit_card ON transactions.card_id = credit_card.id
    JOIN companies ON transactions.business_id = companies.company_id
    WHERE
        companies.company_name = 'Donec Ltd'
        AND transactions.declined = 0
    GROUP BY nom_companyia, IBAN;
```

#	nom_companyia	IBAN	import_mitja_transaccio_en_euros
1	Donec Ltd	PT87806228135092429456346	42.82

✓	1	14:24:22	SELECT	companies.company_name AS nom_companyia…	1 row(s) returned

A efectes d'aquest exercici, s'han filtrat totes les transaccions rebutjades (és a dir, aquelles amb estat *declined=1*).

S'arrodoneix l'import mitjà per transacció a dos deciamals.

La companyia Donec Ltd només té un IBAN amb el que hagi fet transaccions, amb un valor mitjà de 42.82 euros.

NIVELL 2 – Exercici 1

Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les últimes tres transaccions van ser declinades i genera la següent consulta:

En lloc de crear una taula amb aquesta informació, es pot fer servir una vista.

Les vistes es generen amb el contingut actualitzat de les taules en cada crida, cosa que pot ser interessant si es vol consultar l'estat de les targetes recurrentment.

Per crear la vista farem servir l'ordre **CREATE VIEW**. Si es preferís fer servir una taula, es crearia a través de l'ordre CREATE TABLE.

```
-- Opció fent servir vistes. Aquesta opció és la recomanada ja que les vistes es creen en base a
-- les dades actualitzades cada vegada que es criden.
CREATE VIEW vista_credit_card_status AS
    SELECT
        card_id AS "ID_targeta",
        IF(SUM(declined) = 3, "Inactiva", "Activa") AS "Estat_targeta",
        SUM(declined) AS "Errors_3_ultims_usos"
    FROM
        (
        SELECT
            timestamp,
            declined,
            card_id,
            RANK() OVER(partition by card_id order by timestamp desc) AS ranking
        FROM transactions
        ) AS last_card_uses_ranked
    WHERE ranking BETWEEN 1 AND 3
    GROUP by card_id;
```

```
22  10:24:21  CREATE VIEW vista_credit_card_status AS  SELECT  card_id AS "ID_targeta",  ...  0 row(s) affected
```

Combinada amb la **funció finestra OVER** que ens permet aplicar una funció sobre un subgrup de files i en un ordre específic, podem fer servir la funció **RANK** per numerar les transaccions en order invers (és a dir, l'última transacció tindrà el valor 1) i així trobar les tres últimes transaccions per cada targeta (aquelles amb valors 1, 2 i 3).

Sumant el valor del camp *declined* per cada grup de tres últimes transaccions i sabent que és 1 per operacions rebutjades i 0 per acceptades, podem concloure que només estaran **inactives aquelles targetes amb una suma de valor 3**.

Es presenta una part dels resultats de la vista creada. L'última columna mostra el nombre d'errors (consecutius o no) que s'han produït en els tres darrers usos de cada targeta:

```
1 •  SELECT * FROM sprint4.vista_credit_card_status;
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	ID_targeta	Estat_targeta	Errors_3_ultims_usos
1	CcU-2938	Activa	0
2	CcU-2945	Activa	1
3	CcU-2952	Activa	1
4	CcU-2959	Activa	0
5	CcU-2966	Activa	1
6	CcU-2973	Activa	1
7	CcU-2980	Activa	1
8	CcU-2987	Activa	1
9	CcU-2994	Activa	0
10	CcU-3001	Activa	1

NIVELL 2 – Exercici 1.b

Quantes targetes estan actives?

Fent servir la vista creada **vista_credit_card_status**, es compten les targetes amb *Estat_targeta = "Activa"*.

```
-- Per mostrar el total de targetes actives
SELECT COUNT(ID_targeta) AS targetes_actives
FROM vista_credit_card_status
WHERE Estat_targeta = "Activa";
```

#	targetes_actives
1	275

```
23  10:24:21  SELECT COUNT(ID_targeta) AS targetes_actives  FROM vista_credit_card_status  WHERE Estat_tar...  1 row(s) returned
```

Totes les targetes de la base de dades (un total de **275**) es troben **actives**.

Addicionalment, **com a prova** del correcte funcionament, s'insereixen tres transaccions TEST amb el mateix codi de targeta i estat rebutjat (*declined = 1*).

Podem veure com ara aquesta targeta queda inactiva:

```
INSERT INTO transactions (id, card_id, timestamp, declined)
VALUES
    ("TEST1", "CcU-2938", NOW(), 1),
    ("TEST2", "CcU-2938", NOW(), 1),
    ("TEST3", "CcU-2938", NOW(), 1);
```

```
1 •  SELECT * FROM sprint4.vista_credit_card_status;
```

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

#	ID_targeta	Estat_targeta	Errors_3_ultims_usos
1	CcU-2938	Inactiva	3
2	CcU-2945	Activa	1
3	CcU-2952	Activa	1
4	CcU-2959	Activa	0
5	CcU-2966	Activa	1

Acabada la comprovació, eliminem les transaccions TEST1, TEST2, TEST3 per deixar les dades en el seu estat original.

NIVELL 3 – Exercici 1

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.

Es crea la taula productes:

```
CREATE TABLE IF NOT EXISTS products (
    id INT,
    product_name VARCHAR(200),
    price VARCHAR(20),
    colour VARCHAR(7),
    weight FLOAT,
    warehouse_id VARCHAR(10),
    PRIMARY KEY (id)
);
```

```
CREATE TABLE IF NOT EXISTS transaction_product AS ...   0 row(s) affected
                                                        Records: 0  Duplicates: 0  Warnings: 0
```

S'importen les dades des de l'arxiu products.csv:

```
-- s'importen les dades des de products.csv
LOAD DATA LOCAL INFILE "/home/ITAcademy/SPRINT4/csv/products.csv"
INTO TABLE products
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

```
25  10:24:21  LOAD DATA LOCAL INFILE "/home/...  100 row(s) affected
                                                 Records: 100  Deleted: 0  Skipped: 0  Warnings: 0  0,010 sec
```

Es crea la taula ***transaction_product*** que farà de **taula intermitja** entre la taula *transactions* i la taula *products*, ja que es tracta d'una **relació m:n** (en la que una transacció pot tenir més d'un producte i cada producte pot formar part de més d'una transacció).

1	**M**	**N**	1
transactions. id	**transaction_product. transaction_id**	**transaction_product. product_id**	products. id

Taula intermitja
transaction_product
M : N

```sql
CREATE TABLE IF NOT EXISTS transaction_product AS
    SELECT
        transactions.id as transaction_id,
        products.id as product_id
    FROM transactions
    CROSS JOIN products
    WHERE
        FIND_IN_SET(products.id , REPLACE(product_ids, ", ", ",")) > 0
        AND transactions.declined = 0;
```

La taula *transactions* no està normalitzada (no compleix que hi hagi solament un valor per columna): la columna *product_ids* conté un o més id de producte, separats per comes seguides d'espais en blanc. Es fa servir la funció REPLACE per eliminar aquests espais en blanc.

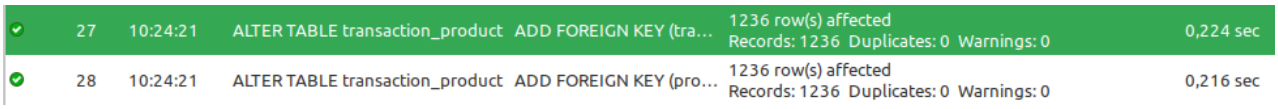
Creuant (CROSS JOIN) cada transacció amb cada possible producte, podem utilitzar **FIND_IN_SET** (ara que ja no tenim espais en blanc) per comprovar si *products.id* forma part del "SET" a *transactions.product_ids* i filtrar únicament els resultats on en formi part.

FIND_IN_SET retorna l'índex dintre del SET, o zero si no troba allò que es busca, per aquesta raó s'agafen únicament les files on el retorn és **superior a zero**.

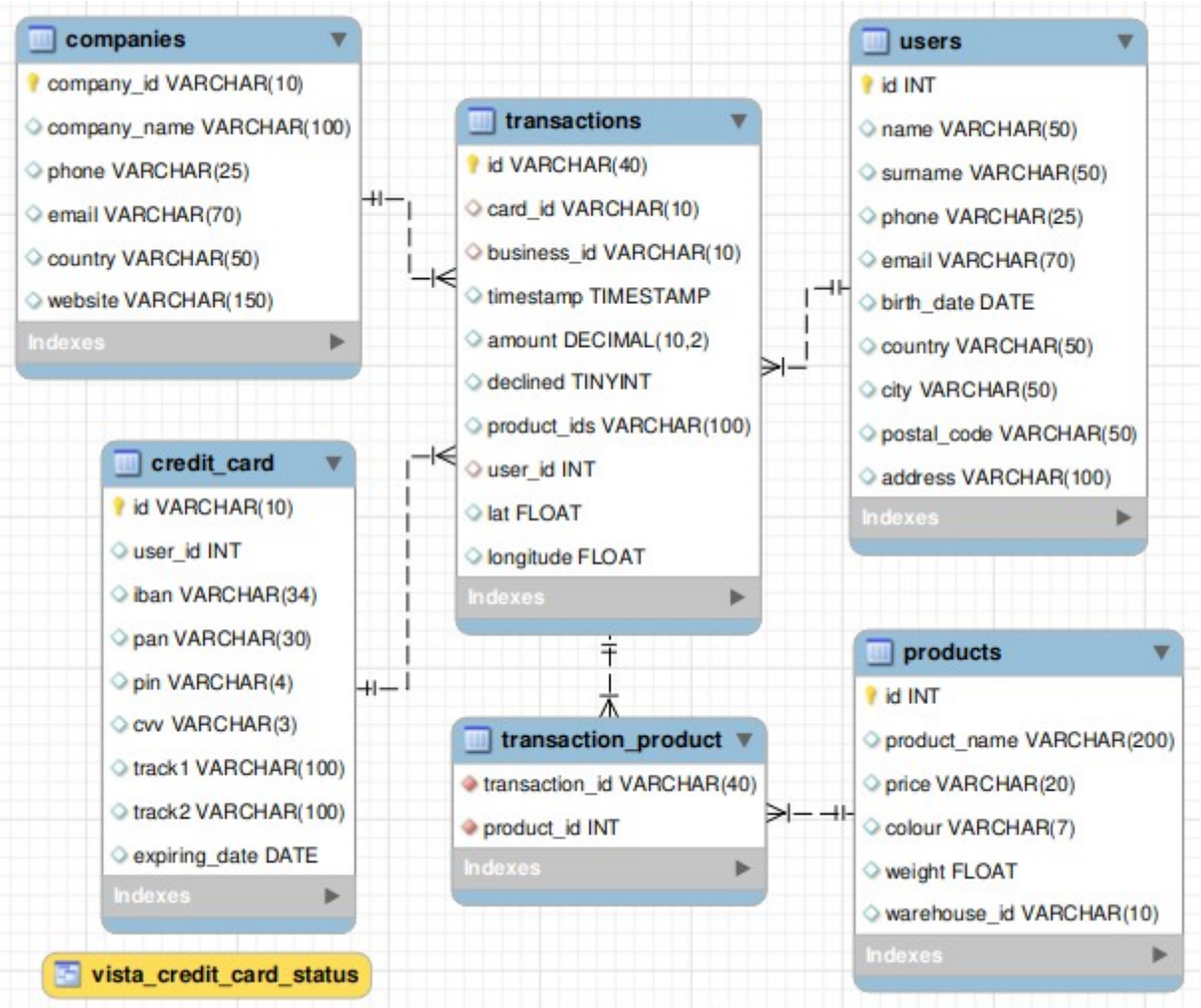
Una vegada més, només es tenen en compte les transaccions acceptades (*declined=0*).

Tenint en ment la relació entre taules ja presentada, s'afegeixen les claus foràanies que les referencien:

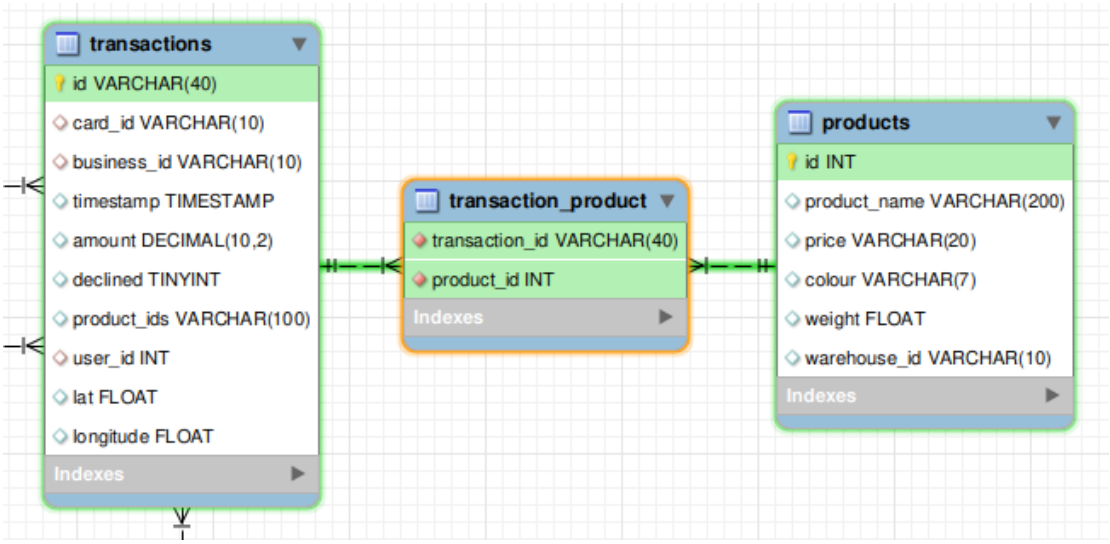
```sql
ALTER TABLE transaction_product
ADD FOREIGN KEY (transaction_id) REFERENCES transactions (id);

ALTER TABLE transaction_product
ADD FOREIGN KEY (product_id) REFERENCES products (id);
```



La relació final entre taules, queda com es presenta a continuació.



Cal parar especial atenció a com els costats M:N queden a la taula intermitja, i tenen una relació a 1 amb les taules *transactions* i *products*:



Genera la següent consulta: Necessitem conèixer el nombre de vegades que s'ha venut cada producte.

```sql
-- unitats venudes de cada producte
SELECT
    products.id AS id_del_producte,
    COUNT(IF(transactions.declined = 0, transactions.id, NULL)) AS unitats_venudes,
    products.product_name AS nom_del_producte
FROM products
LEFT JOIN transaction_product ON products.id = transaction_product.product_id
LEFT JOIN transactions ON transactions.id = transaction_product.transaction_id
GROUP BY products.id
ORDER BY products.id;
```

```
29  10:24:21  SELECT  products.id AS id_del_producte,  COUNT...  100 row(s) returned
```

Fent servir LEFT JOIN, deixant la taula products a l'esquerra, ens assegurem que es mostren totes les id de producte (100 ids diferents), també aquelles de productes que no han tingut transaccions o han estat rebutjades.

Amb el condicional IF es dona valor NULL a les transaccions rebutjades de forma que en el COUNT només s'han tingut en consideració les transaccions acceptades (és a dir, amb *declined = 0*). Així es té informació per cada identificador de producte fins i tot i quan el número d'unitats venudes sigui zero per aquest.

Junt amb l'identificador de producte i el número total d'unitats venudes, es mostra el nom del producte per fer la taula més descriptiva.

En la imatge a continuació, es pot veure una mostra de les primeres 10 entrades que conformen la taula, formada per un total de 100 files, una per cada *id_del_producte.*

#	id_del_producte	unitats_venudes	nom_del_producte
1	1	51	Direwolf Stannis
2	2	56	Tarly Stark
3	3	43	duel tourney Lannister
4	4	0	warden south duel
5	5	42	skywalker ewok
6	6	0	dooku solo
7	7	44	north of Casterly
8	8	0	Winterfell
9	9	0	Winterfell
10	10	0	Karstark Dorne

```
1  20:41:35  SELECT  products.id AS id_del_producte,  COUNT(IF(tra...  100 row(s) returned
```