Algoritmo de empresa WineHouse

Repositorio GitHub: https://github.com/tomassale/SQL.git

El algoritmo consta de la creación de una base de datos donde se almacena la información cuando el usuario inicia sesión en su cuenta previamente registrada, la cual forma parte de la DB y comienza el proceso de compilación de datos. La empresa junta información, en la que, mediante cookies y el browser reúne los datos de búsqueda, sitios recurrentes o información personal para tener una interfaz más personalizada y adaptada al consumidor. El proceso comienza llenando las tablas de address, personal y account, llevando mediante Primary Keys a la tabla user, la cual también cuenta con una tabla history que registra la Foreign Key de la tabla page. Después de este proceso lleva la información a la tabla data la cual es solicitada por las empresas siendo registradas en la tabla company.

Índice

Tablas 3

DER 5

Vistas 6

Vista logs 6

Vista user 6

Vista page 7

Vista personal 7

Vista company 7

Vista join 8

Importación 9

Funciones 13

Funcion user 13

Funcion company 13

Funcion page 14

Funcion data 14

Funcion logs 15

Stored Procedures 16

Stored Procedure orden 16

Stored Procedure agregar datos 17

Stored Procedures eliminar datos 21

Triggers 23

Triggers before insert 23

Triggers before delete 23

Triggers before update 24

Tablas

Tabla para logs de usuarios en la db

	logs(T.D.)	
id_logs (PK)	id logs	INT
table_logs	tabla	VARCHAR(20)
dml	sentencia	VARCHAR(20)
registered_logs	fecha	DATETIME
user	usuario	VARCHAR(50)
db	bd	VARCHAR(20)
version	version	VARCHAR(20)

Tabla para datos guardados de usuario

	user(T.H.)	
id_user (PK)	id usuario	INT
user_mail (FK)	mail	VARCHAR(50)
id_personal (FK)	id personal	INT
ip_user (FK)	ip	VARCHAR(20)
id_history (FK)	id historial	INT

Tabla para datos personales de usuario

	personal (T.D.)	
id_personal (PK)	id personal	INT
gender	genero	VARCHAR(10)
user_first_name	nombre	VARCHAR(20)
user_last_name	apellido	VARCHAR(30)
age	edad	INT

Tabla para cuenta de usuario

	account (T.D.)	
user_mail (PK)	mail	VARCHAR(50)
user_password	contraseña	VARCHAR(50)
register_account	fecha registro	DATETIME

Tabla para direccion de usuario

	address (T.D.)	
ip_user (PK)	ip	VARCHAR(20)
country_code	codigo pais	VARCHAR(10)
city	ciudad	VARCHAR(50)
street	calle	VARCHAR(60)

Tabla para historial de usuario

	history (T.D.)	
id_history (PK)	id historial	INT
person_history	historial pers.	INT
name_page (FK)	nombre pag.	VARCHAR(50)

Tabla para paginas

	page (T.D.)	
name_page (PK)	nombre pag.	VARCHAR(50)
date_registered_page	fecha registrada	DATETIME
info	descripcion	TEXT

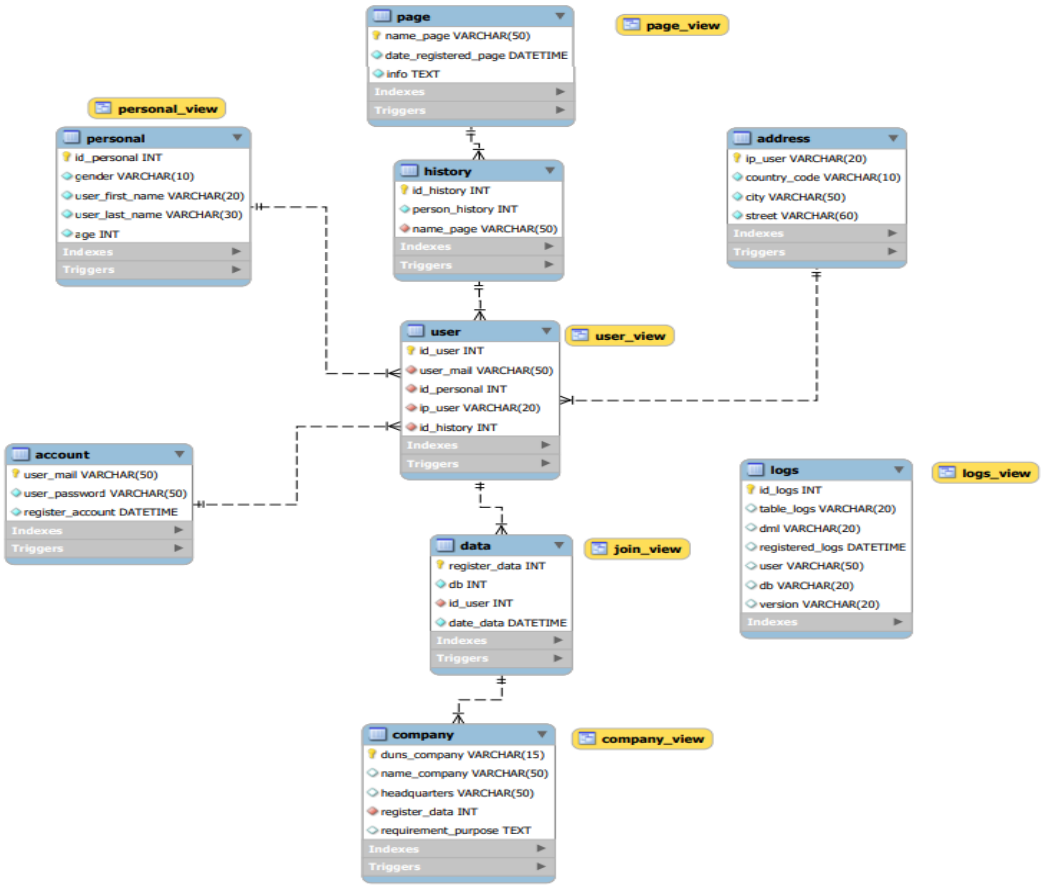
Tabla para datos guardados

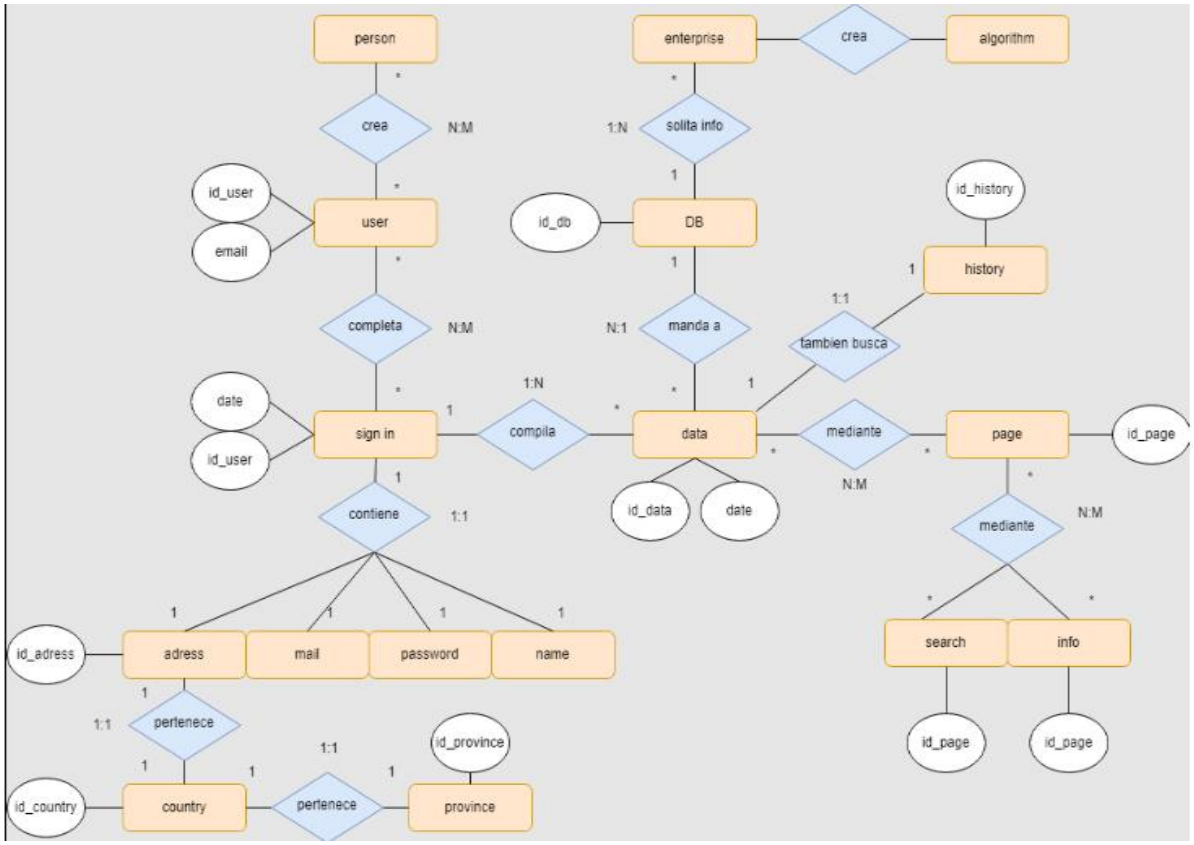
	data (T.H.)	
register_data (PK)	id data	INT
db	bd	INT
id_user (FK)	id usuario	INT
date_data	fecha data	DATETIME

Tabla para las empresas

	company (T.D.)	
duns_company (PK)	duns empresa	VARCHAR(15)
name_company	nombre	VARCHAR(50)
headquarters	sede central	VARCHAR(50)
register_data (FK)	id data	INT
requirement_purpose	requerimiento	TEXT

DER


page
name_page VARCHAR(50)
date_registered_page DATETIME
info TEXT
Indexes
Triggers
page_view
personal_view
personal
id_personal INT
gender VARCHAR(10)
user_first_name VARCHAR(20)
user_last_name VARCHAR(30)
age INT
Indexes
Triggers
history
id_history INT
person_history INT
name_page VARCHAR(50)
Indexes
Triggers
address
ip_user VARCHAR(20)
country_code VARCHAR(10)
city VARCHAR(50)
street VARCHAR(60)
Indexes
Triggers
user
id_user INT
user_mail VARCHAR(50)
id_personal INT
ip_user VARCHAR(20)
id_history INT
Indexes
Triggers
user_view
account
user_mail VARCHAR(50)
user_password VARCHAR(50)
register_account DATETIME
Indexes
Triggers
logs
id_logs INT
table_logs VARCHAR(20)
dml VARCHAR(20)
registered_logs DATETIME
user VARCHAR(50)
db VARCHAR(20)
version VARCHAR(20)
Indexes
logs_view
data
register_data INT
db INT
id_user INT
date_data DATETIME
Indexes
Triggers
join_view
company
duns_company VARCHAR(15)
name_company VARCHAR(50)
headquarters VARCHAR(50)
register_data INT
requirement_purpose TEXT
Indexes
Triggers
company_view


person
enterprise
crea
algorithm
crea
N:M
1:N
solita info
id_user
email
user
id_db
DB
id_history
history
completa
N:M
N:1
manda a
1:1
tambien busca
date
id_user
sign in
1:N
compila
data
mediante
page
id_page
N:M
contiene
1:1
id_data
date
mediante
N:M
id_adress
adress
mail
password
name
search
info
pertenece
1:1
id_province
id_country
country
pertenece
province
id_page
id_page

Vistas

1- Vista que trae id registros. registros de tablas, sentencia dml y fecha del registro de la tabla logs.

```
CREATE OR REPLACE VIEW logs_view as
    SELECT id_logs, table_logs, dml, registered_logs, user
    FROM logs;
```

id_logs	table_logs	dml	registered_logs	user
1	personal	Insert	2022-07-19 14:40:02	root@localhost
2	personal	Insert	2022-07-19 14:40:02	root@localhost
3	personal	Insert	2022-07-19 14:40:02	root@localhost
4	personal	Insert	2022-07-19 14:40:02	root@localhost
5	personal	Insert	2022-07-19 14:40:02	root@localhost
6	personal	Insert	2022-07-19 14:40:02	root@localhost
7	personal	Insert	2022-07-19 14:40:02	root@localhost
8	personal	Insert	2022-07-19 14:40:02	root@localhost
9	personal	Insert	2022-07-19 14:40:02	root@localhost
10	personal	Insert	2022-07-19 14:40:02	root@localhost

2- Vista que trae id del usuario, mail, id personal, ip y el id de historial de la tabla user.

```
CREATE OR REPLACE VIEW user_view as
    SELECT *
    FROM user;
```

id_user	user_mail	id_personal	ip_user	id_history
1	abelison3@hc360.com	2	199.52.19.14	2
2	mglowinski0@answers.com	15	125.207.210.105	15
3	dmcinility6@rediff.com	1	170.2.31.135	1
4	sbartkiewicze@shutterfly.com	4	23.206.0.188	4
5	bbearward7@globo.com	6	29.100.248.181	6
6	afattorec@hud.gov	7	102.231.167.128	7
7	ganwyl4@wordpress.com	10	182.104.207.141	10
8	tdelany5@arstechnica.com	12	240.253.8.236	12
9	nblade8@creativecommons.org	5	107.191.247.221	5
10	dcassells1@bloomberg.com	9	198.1.168.51	9

3- Vista que trae nombre de la página y fecha registrada de la tabla page.

```
CREATE OR REPLACE VIEW page_view as
    SELECT name_page, date_registered_page
    FROM page;
```

name_page	date_registered_page
Douglas-Walsh	2022-01-15 07:39:57
Erdman	2022-05-04 07:59:01
Graham and Wuckert	2021-09-30 15:21:13
Gutkowski-Franecki	2022-04-02 13:11:30
Heaney-Wolf	2021-12-25 03:34:42
Hoppe-Harber	2022-06-09 02:14:28
Kub-Crist	2021-12-03 03:30:13
Leffler-Dickens	2021-12-26 11:09:50
Lubowitz and Luettgen	2022-05-14 21:15:37
Maggio and Pagac	2022-04-09 05:23:48

4-Vista que trae nombre y apellido de la tabla personal.

```
CREATE OR REPLACE VIEW personal_view as
    SELECT user_first_name, user_last_name
    FROM personal;
```

user_first_name	user_last_name
Jaimie	Dufton
Jasun	Rossant
Audrye	Guise
Julietta	Frances
Reiko	Forcer
Kerwinn	Walkinshaw
Doti	Moller
Aveline	Thorndale

5- Vista que trae duns de la empresa, nombre y motivo de la tabla company.

```
CREATE OR REPLACE VIEW company_view as
    SELECT duns_company, name_company, requirement_purpose
    FROM company;
```

duns_company	name_company	requirement_purpose
04-413-3487	Jaxspan	vestibulum ante ipsum primis in faucibus orci luct...
07-679-8593	Realpoint	blandit mi in porttitor pede justo eu massa done...
13-844-8586	Mudo	posuere nonummy integer non velit donec diam ...
19-854-1813	Wikido	at turpis donec posuere metus vitae ipsum aliqu...
33-055-2198	Browsecat	libero quis orci nullam molestie nibh in lectus pell...
35-063-1428	Feedspan	amet eleifend pede libero quis orci nullam molest...
37-113-3087	Feedmix	varius integer ac leo pellentesque ultrices mattis...
40-965-0993	Vidoo	rhoncus mauris enim leo rhoncus sed vestibulum...
41-017-6232	Yata	dictumst morbi vestibulum velit id pretium iaculis ...
43-365-5872	Twinte	rhoncus sed vestibulum sit amet cursus id turpis...

6- Vista que trae el id de usuario, mail, id de historial y ip de usuario de las tablas address, user, account, history y address.

```
CREATE OR REPLACE VIEW join_view as
    SELECT DISTINCT u.id_user, ac.user_mail, h.id_history, a.ip_user
        FROM user u
        INNER JOIN address a
        ON u.ip_user = a.ip_user
        INNER JOIN account ac
        ON ac.user_mail = u.user_mail
        INNER JOIN history h
        ON h.id_history = u.id_history;
    ;
```

id_user	user_mail	id_history	ip_user
1	abelison3@hc360.com	2	199.52.19.14
2	mglowinski0@answers.com	15	125.207.210.105
3	dmcinility6@rediff.com	1	170.2.31.135
4	sbartkiewicze@shutterfly.com	4	23.206.0.188
5	bbearward7@globo.com	6	29.100.248.181
6	afattorec@hud.gov	7	102.231.167.128
7	ganwyl4@wordpress.com	10	182.104.207.141
8	tdelany5@arstechnica.com	12	240.253.8.236
9	nblade8@creativecommons.org	5	107.191.247.221
10	dcassells1@bloomberg.com	9	198.1.168.51

Importación

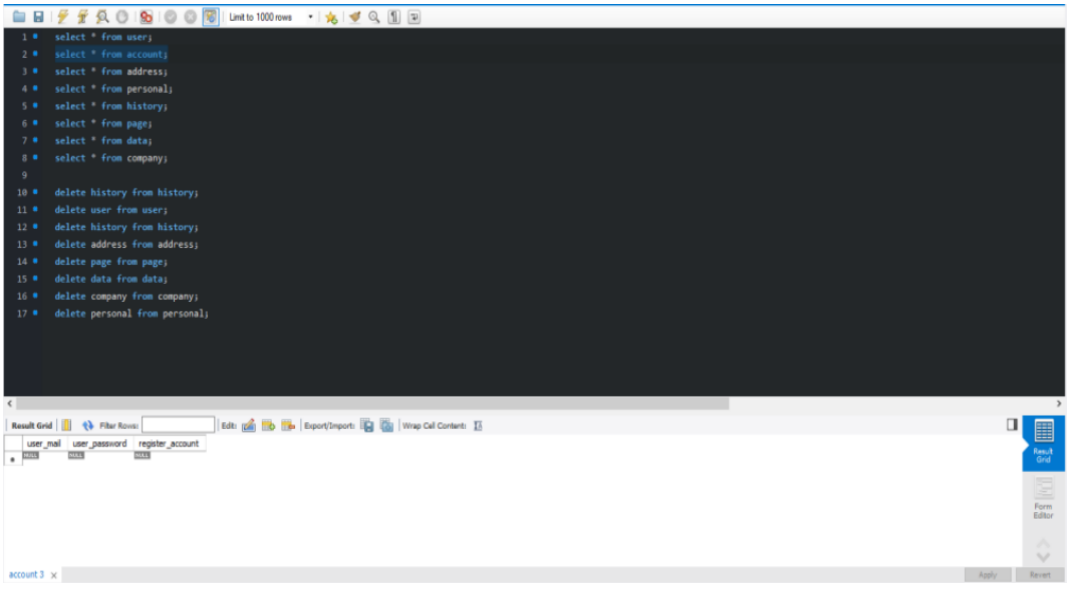
1. Se crea el archivo .csv que contiene la información a importar

account.csv

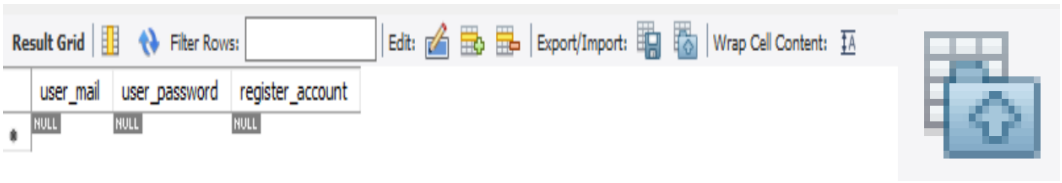
E: > PRACTICA > CODERHOUSE > SQL > SQL files > .csv > account.csv

```
user_mail,user_password,register_account
mglowinski0@answers.com,ibbPz2v5Crk,2021-07-06 12:35:15
dcassells1@bloomberg.com,4tKv989S1Lmk,2021-12-20 14:17:16
scasol2@paypal.com,77RAlVtvA,2021-09-21 05:11:59
abelison3@hc360.com,DncpgPW,2022-05-08 05:51:13
ganwyl4@wordpress.com,qp3hpI8MBi,2022-05-02 02:49:30
tdelany5@arstechnica.com,uzVT84w40,2021-08-04 22:13:49
dmcinility6@rediff.com,u6jyjofjH,2021-11-04 23:19:33
bbearward7@globo.com,7HI9JpnYMPRS,2022-05-16 03:30:05
nblade8@creativecommons.org,DEYEVdeTfHz,2021-07-24 17:37:47
hdoche9@diigo.com,0qbkEs0G,2022-02-21 13:12:03
speacocka@g.co,p8yg6M,2021-12-25 18:47:14
fexonb@japanpost.jp,rIsxqOT,2022-06-26 14:16:24
afattorec@hud.gov,cNCT54Okf,2021-07-09 19:27:02
twetternd@1und1.de,QcxNh6R,2021-12-21 19:37:56
sbartkiewicze@shutterfly.com,e6riweUowTA,2022-02-07 11:19:51
```

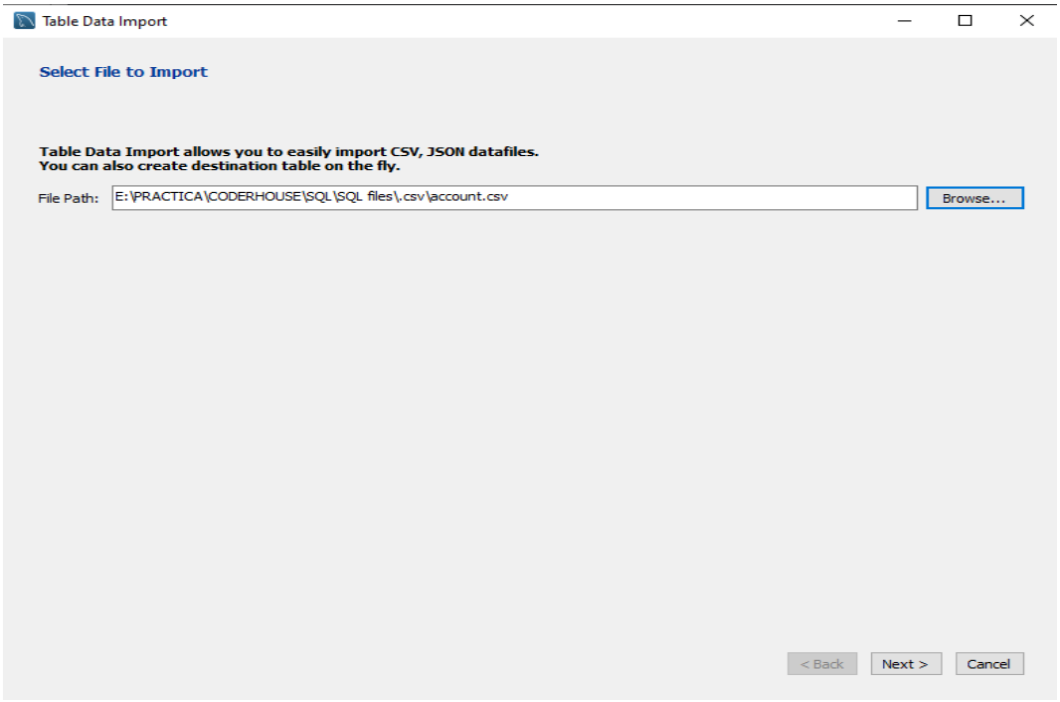
2. Se selecciona la tabla previamente creada a la que se importaran los datos



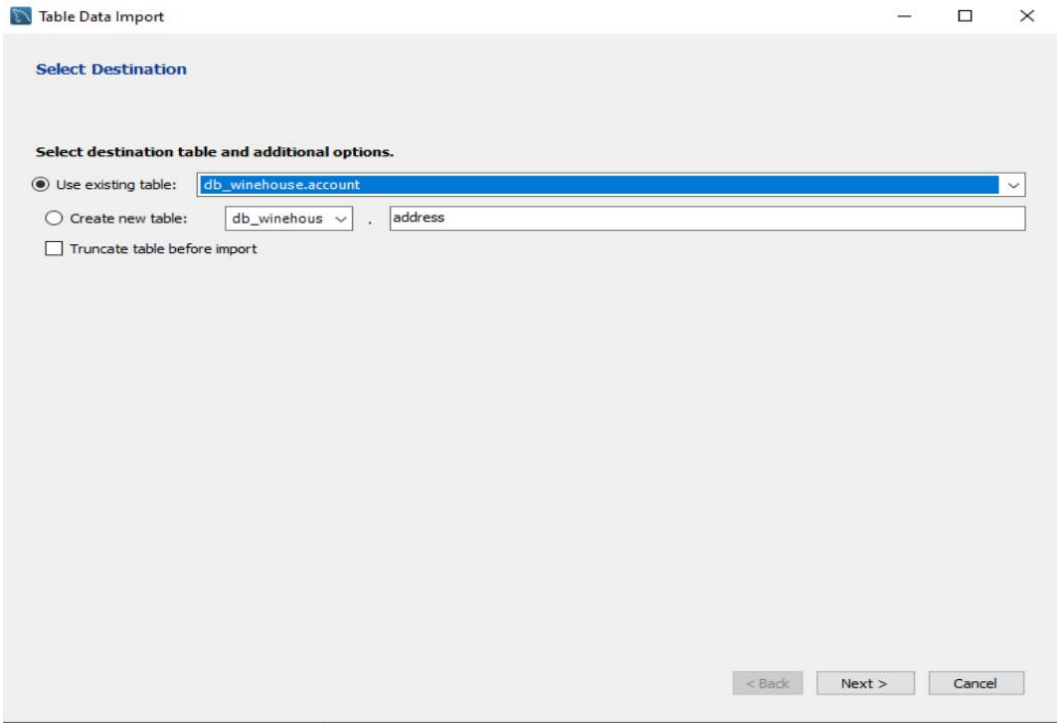
3. En la barra superior se selección la herramienta de importación



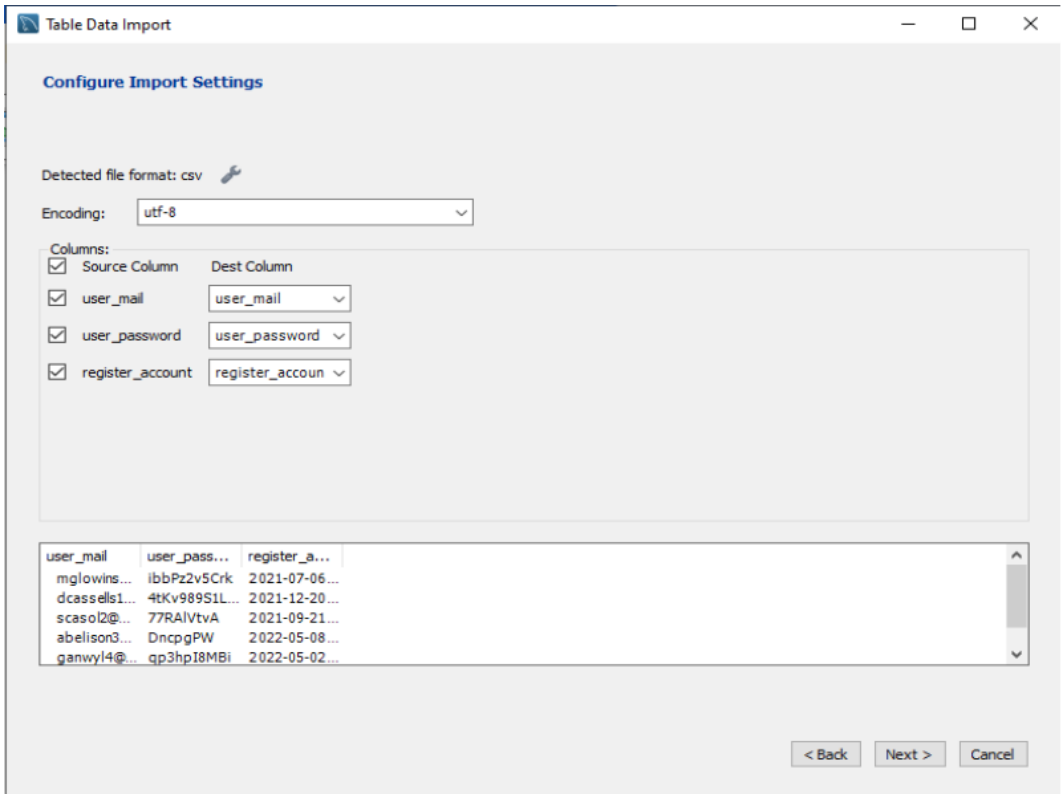
4. Se selecciona el archivo a importar y se presiona next



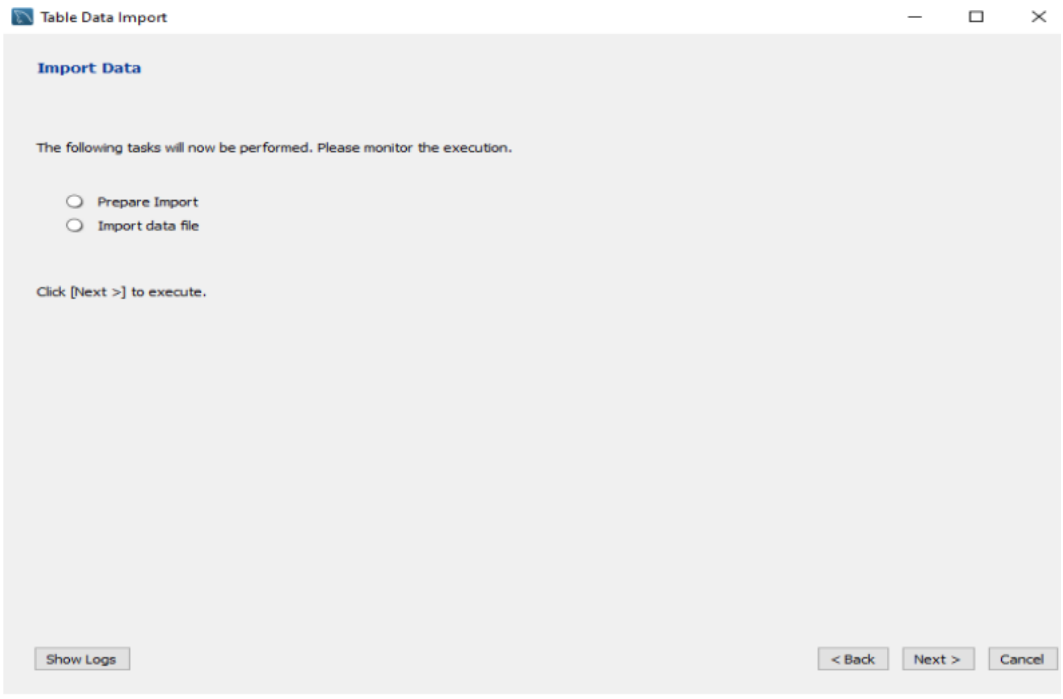
5. Se selecciona en que tabla se desea importar o se crea si es inexistente y se presiona next

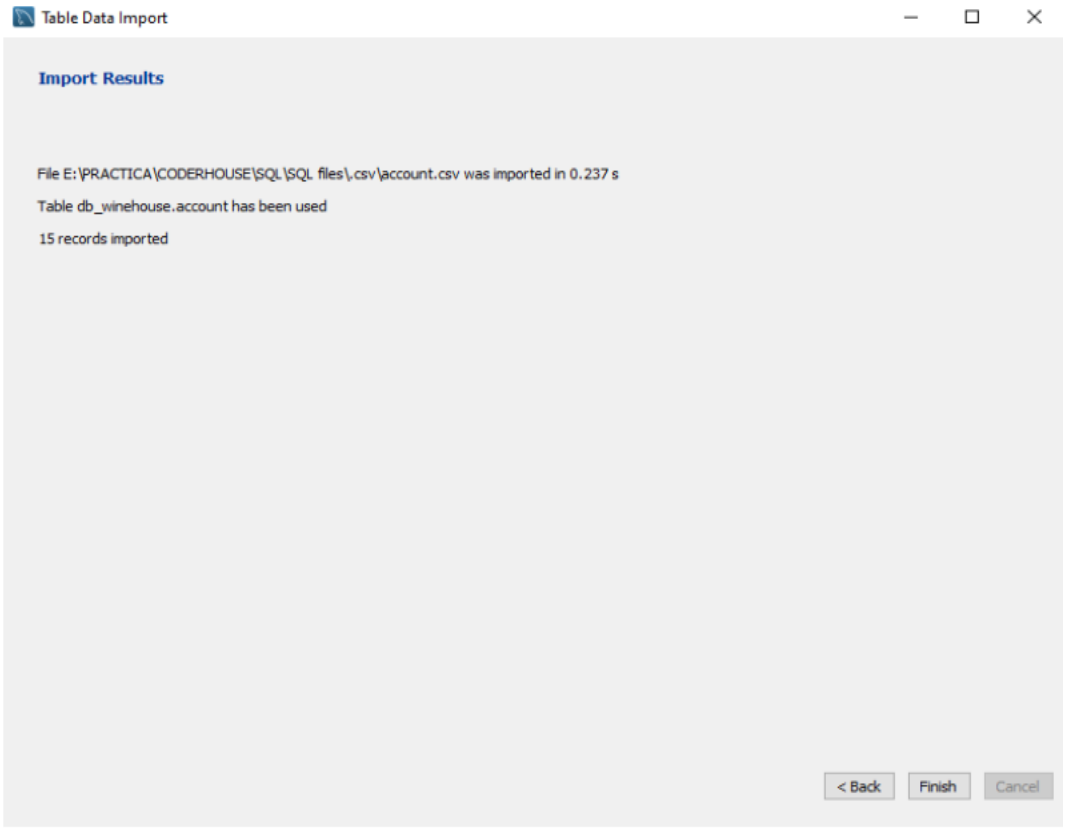


6. Aquí veremos una previsualización de los datos y se puede seleccionar si alguno no se importa. Se presiona nuevamente next

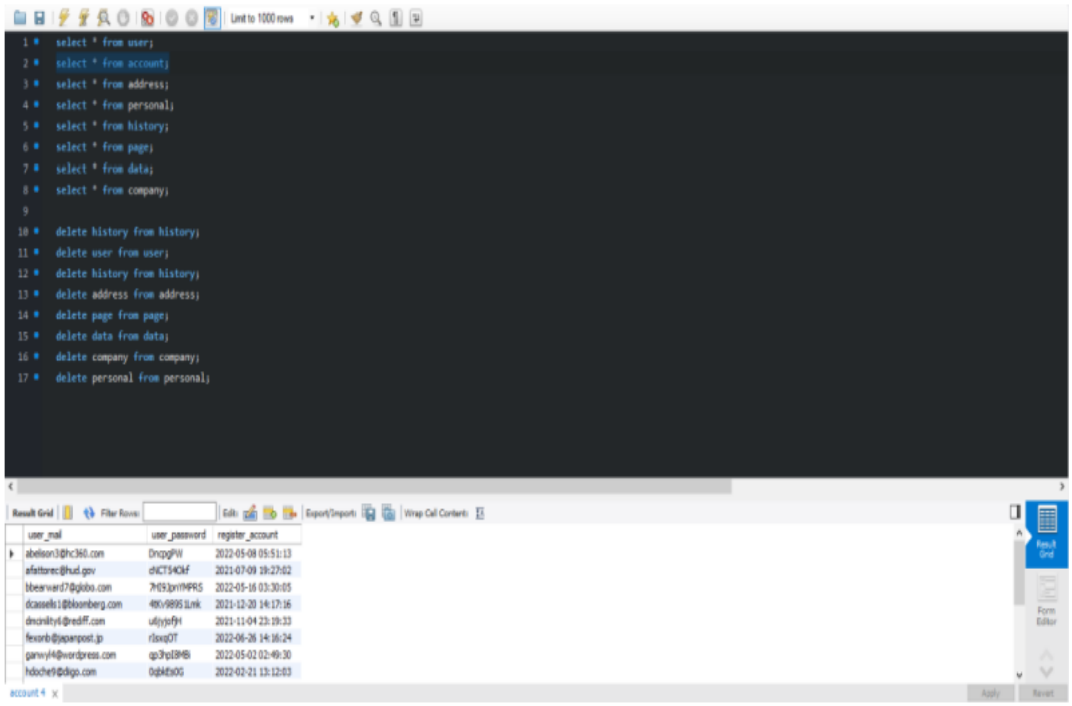


7. Nuevamente next para confirmar la importación.





Presionamos finish y visualizamos los datos importados en MySQL Workbench



Funciones

1-Función buscadora de id mediante parámetros de nombre (name) y apellido (last_name) en tabla user.

```
DROP FUNCTION IF EXISTS `user_id`;
DELIMITER $$
CREATE FUNCTION `user_id`(name VARCHAR(20), last_name VARCHAR(20)) RETURNS VARCHAR(50)
READS SQL DATA
BEGIN
    DECLARE id_u VARCHAR(50);
    SET id_u = (SELECT id_personal FROM personal WHERE
                    user_first_name IN (SELECT user_first_name FROM personal WHERE user_first_name = name)
                        AND
                    user_last_name IN (SELECT user_last_name FROM personal WHERE user_last_name = last_name)
                );
    IF isnull(id_u) THEN
        RETURN CONCAT('Nombre o apellido invalido');
    ELSE
        RETURN CONCAT('ID usuario: ',id_u);
    END IF;
END$$
```

2- Función buscadora de duns por parámetro de nombre de la empresa (name) en la tabla company.

```
DROP FUNCTION IF EXISTS `company_duns`;
DELIMITER $$
CREATE FUNCTION `company_duns`(name VARCHAR(20)) RETURNS VARCHAR(50)
READS SQL DATA
BEGIN
    DECLARE duns_c VARCHAR(50);
    SET duns_c = (SELECT duns_company FROM company WHERE
                    name_company IN (SELECT name_company FROM company WHERE name_company = name)
                );
    IF isnull(duns_c) THEN
        RETURN CONCAT('Nombre invalido');
    ELSE
        RETURN CONCAT('duns empresa: ',duns_c);
    END IF;
END$$
```

3- Función trae descripción de página por parámetro de nombre (name) de la tabla page.

```
DROP FUNCTION IF EXISTS `page_info`;
DELIMITER $$
CREATE FUNCTION `page_info`(name VARCHAR(20)) RETURNS TEXT
READS SQL DATA
BEGIN
    DECLARE info_p TEXT;
    SET info_p = (SELECT info FROM page WHERE
                    name_page IN (SELECT name_page FROM page WHERE name_page = name)
                );
    IF isnull(info_p) THEN
        RETURN CONCAT('Nombre invalido');
    ELSE
        RETURN CONCAT('Descripcion empresa: ',info_p);
    END IF;
END$$
```

4- Función trae a que db está vinculada la cuenta por parámetro de id del usuario (id_u) de la tabla data.

```
DROP FUNCTION IF EXISTS `data_db`;
DELIMITER $$
CREATE FUNCTION `data_db`(id_u INT) RETURNS VARCHAR(50)
READS SQL DATA
BEGIN
    DECLARE db_d INT;
    SET db_d = (SELECT db FROM data WHERE
                    id_user IN (SELECT id_user FROM user WHERE id_user = id_u)
                );
    IF isnull(db_d) THEN
        RETURN CONCAT('Id invalido');
    ELSE
        RETURN CONCAT('Base de datos: ', db_d);
    END IF;
END$$
```

5- Función trae cantidad de sentencias ejecutada por el usuario mediante parámetro de nombre de usuario (p_user), sentencia (p_dml) y tabla (p_table_logs) de la tabla logs.

```
DROP FUNCTION IF EXISTS `logs_id`;
DELIMITER $$
CREATE FUNCTION `logs_id`(p_user VARCHAR(50), p_dml VARCHAR(20), p_table_logs VARCHAR(20)) RETURNS VARCHAR(50)
READS SQL DATA
BEGIN
    DECLARE id_l INT;
    SET id_l = (SELECT COUNT(id_logs) FROM logs WHERE
                    user IN (SELECT user FROM logs WHERE user = p_user)
                                    AND
                    dml IN (SELECT dml FROM logs WHERE dml = p_dml)
                                    AND
                    table_logs IN (SELECT table_logs FROM logs WHERE table_logs = p_table_logs)
                );
    IF isnull(id_l) THEN
        RETURN CONCAT('User, sentencia o tabla invalido');
    ELSE
        RETURN CONCAT('Cantidad de movimientos: ', id_l);
    END IF;
END$$
```

Stored Procedure

*Procedimiento almacenado para traer una fila ordenada de la tabla recibida por parámetro (p_table), el campo recibido por parámetro (p_field) y en el orden recibido de la misma forma (p_ord).

```
DROP PROCEDURE IF EXISTS `sp_get_order`;
DELIMITER $$
CREATE PROCEDURE `sp_get_order`(IN p_table VARCHAR(50), IN p_field VARCHAR(50), IN p_ord VARCHAR(20))
BEGIN
    IF p_table = '' THEN
        SELECT 'Seleccione tabla' ERROR;
    ELSE
        SET @order_table = CONCAT(' ', p_table);
        IF p_field = '' THEN
            SET @order_field = '';
        ELSE
            IF p_ord = '' THEN
                SET @order_way = '';
            ELSE
                SET @order_field = CONCAT(' ORDER BY ', p_field);
                SET @order_way = CONCAT(' ', upper(p_ord));
            END IF;
        END IF;
    END IF;
    SET @clausula = CONCAT('SELECT * FROM ', @order_table, @order_field, @order_way);
    PREPARE runSQL FROM @clausula;
    EXECUTE runSQL;
    DEALLOCATE PREPARE runSQL;
END$$
```

*Procedimiento almacenado para guardar datos en las tablas (todas las tablas incluidas en su sp correspondiente) simplificadora de insert.

```
DROP PROCEDURE IF EXISTS `sp_insert_address`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_address`(
    IN p_ip_user VARCHAR(20),
    IN p_country_code VARCHAR(50),
    IN p_city VARCHAR(50),
    IN p_street VARCHAR(50))
BEGIN
    IF p_ip_user = '' OR p_country_code = '' OR p_city = '' OR p_street = '' THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO address (`ip_user`,`country_code`,`city`,`street`)
        VALUES (p_ip_user, p_country_code, p_city, p_street);
        SELECT * FROM address a WHERE a.ip_user = p_ip_user;
    END IF;
END$$
```

```
DROP PROCEDURE IF EXISTS `sp_insert_history`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_history`(
    IN p_person_history INT,
    IN p_name_page VARCHAR(50))
BEGIN
    IF p_person_history = 0 OR p_name_page = '' THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO history (`id_history`,`person_history`,`name_page`)
        VALUES (NULL, p_person_history, p_name_page);
        SELECT MAX(id_history) id, person_history, name_page FROM history;
    END IF;
END$$
```

```sql
DROP PROCEDURE IF EXISTS `sp_insert_page`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_page`(
    IN p_name_page VARCHAR(50),
    IN p_date_registered_page DATETIME,
    IN p_info TEXT)
BEGIN
    IF p_name_page = '' OR p_date_registered_page = '' OR p_info = '' THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO page (`name_page`,`date_registered_page`,`info`)
        VALUES (p_name_page, p_date_registered_page, p_info);
        SELECT * FROM page p WHERE p.name_page = p_name_page;
    END IF;
END$$
```

```sql
DROP PROCEDURE IF EXISTS `sp_insert_data`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_data`(
    IN p_db INT,
    IN p_id_user INT)
BEGIN
    IF p_db = '' OR p_id_user = '' THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO data (`register_data`,`db`,`id_user`,`date_data`)
        VALUES (NULL, p_db, p_id_user, NULL);
        SELECT MAX(register_data) id, db, id_user, date_data date FROM data;
    END IF;
END$$
```

```
DROP PROCEDURE IF EXISTS `sp_insert_company`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_company`(
    IN p_duns_company VARCHAR(15),
    IN p_name_company VARCHAR(50),
    IN p_headquarters VARCHAR(50),
    IN p_register_data INT,
    IN p_requirement_purpose VARCHAR(200))
BEGIN
    IF p_duns_company = '' OR p_name_company = '' OR p_headquarters = '' OR p_register_data = 0 OR p_requirement_purpose = '' THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO company (`duns_company`,`name_company`,`headquarters`,`register_data`,`requirement_purpose`)
        VALUES (p_duns_company, p_name_company, p_headquarters, p_register_data, p_requirement_purpose);
        SELECT * FROM company c WHERE c.duns_company = p_duns_company;
    END IF;
END$$
```

```
DROP PROCEDURE IF EXISTS `sp_insert_user`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_user`(
    IN p_user_mail VARCHAR(50),
    IN p_id_personal INT,
    IN p_ip_user VARCHAR(20),
    IN p_id_history INT)
BEGIN
    IF p_user_mail = '' OR p_id_personal = 0 OR p_ip_user = '' OR p_id_history = 0 THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO user (`duns_company`,`name_company`,`headquarters`,`register_data`,`requirement_purpose`)
        VALUES (p_duns_company, p_name_company, p_headquarters, p_register_data, p_requirement_purpose);
        SELECT MAX(id_user) id, user_mail mail, id_personal, ip_user ip, id_history FROM user;
    END IF;
END$$
```

```
DROP PROCEDURE IF EXISTS `sp_insert_personal`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_personal`(
    IN p_gender VARCHAR(10),
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(30),
    IN p_age INT)
BEGIN
    IF p_gender = '' OR p_first_name = '' OR p_last_name = '' OR p_age = 0  THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO personal (`id_personal`,`gender`,`user_first_name`,`user_last_name`,`age`)
        VALUES (NULL, p_gender, p_first_name, p_last_name, p_age);
        SELECT MAX(id_personal) id, gender, user_first_name first_name, user_last_name last_name, age FROM personal;
    END IF;
END$$
```

```
DROP PROCEDURE IF EXISTS `sp_insert_account`;
DELIMITER $$
CREATE PROCEDURE `sp_insert_account`(
    IN p_user_mail VARCHAR(50),
    IN p_user_password VARCHAR(50),
    IN p_register_account DATETIME)
BEGIN
    IF p_user_mail = '' OR p_user_password = '' OR p_register_account = '' THEN
        SELECT 'Parametro faltante o invalido' ERROR;
    ELSE
        INSERT INTO account (`user_mail`,`user_password`,`register_account`)
        VALUES (p_user_mail, p_user_password, p_register_account);
        SELECT * FROM account a WHERE a.user_mail = p_user_mail;
    END IF;
END$$
```

*Procedimiento almacenado para eliminar datos en las tablas (todas las tablas incluidas en su sp correspondiente) simplificadora de delete.

```
DROP PROCEDURE IF EXISTS `sp_delete_address`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_address`(IN p_ip_user VARCHAR(15))
BEGIN
    DELETE FROM user u WHERE u.ip_user = p_ip_user;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$

DROP PROCEDURE IF EXISTS `sp_delete_history`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_history`(IN p_id_history VARCHAR(15))
BEGIN
    DELETE FROM history h WHERE h.id_history = p_id_history;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$

DROP PROCEDURE IF EXISTS `sp_delete_page`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_page`(IN p_name_page VARCHAR(15))
BEGIN
    DELETE FROM page p WHERE p.name_page = p_name_page;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$
```

```
DROP PROCEDURE IF EXISTS `sp_delete_data`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_data`(IN p_register_data VARCHAR(15))
BEGIN
    DELETE FROM data d WHERE d.register_data = p_register_data;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$

DROP PROCEDURE IF EXISTS `sp_delete_company`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_company`(IN p_duns_company VARCHAR(15))
BEGIN
    DELETE FROM company c WHERE c.duns_company = p_duns_company;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$
```

```sql
DROP PROCEDURE IF EXISTS `sp_delete_personal`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_personal`(IN p_id_personal VARCHAR(15))
BEGIN
    DELETE FROM personal p WHERE p.id_personal = p_id_personal;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$

DROP PROCEDURE IF EXISTS `sp_delete_user`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_user`(IN p_id_user VARCHAR(15))
BEGIN
    DELETE FROM user u WHERE u.id_user = p_id_user;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$

DROP PROCEDURE IF EXISTS `sp_delete_account`;
DELIMITER $$
CREATE PROCEDURE `sp_delete_account`(IN p_user_mail VARCHAR(15))
BEGIN
    DELETE FROM account a WHERE a.user_mail = p_user_mail;
    SELECT 'Elemento eliminado exitosamente' EXITOSO;
END$$
```

Triggers

*Evento trigger before insert para llevar el log a la tabla logs con su dml, tabla afectada, fecha, usuario, base de datos, y versión

```
CREATE TRIGGER BEF_INS_personal_logs
BEFORE INSERT ON personal
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "personal", "Insert", NOW(), USER(), DATABASE(), VERSION());

CREATE TRIGGER BEF_INS_address_logs
BEFORE INSERT ON address
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "address", "Insert", NOW(), USER(), DATABASE(), VERSION());

CREATE TRIGGER BEF_INS_account_logs
BEFORE INSERT ON account
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "account", "Insert", NOW(), USER(), DATABASE(), VERSION());
```

*Evento trigger before delete para llevar el log a la tabla logs con su dml, tabla afectada, fecha, usuario, base de datos, y versión

```
CREATE TRIGGER BEF_DEL_personal_logs
BEFORE DELETE ON personal
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "personal", "Delete", NOW(), USER(), DATABASE(), VERSION());

CREATE TRIGGER BEF_DEL_address_logs
BEFORE DELETE ON address
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "address", "Delete", NOW(), USER(), DATABASE(), VERSION());

CREATE TRIGGER BEF_DEL_account_logs
BEFORE DELETE ON account
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "account", "Delete", NOW(), USER(), DATABASE(), VERSION());
```

*Evento trigger before update para llevar el log a la tabla logs con su dml, tabla afectada, fecha, usuario, base de datos, y versión

```
CREATE TRIGGER BEF_UPD_personal_logs
BEFORE UPDATE ON personal
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "personal", "Update", NOW(), USER(), DATABASE(), VERSION());

CREATE TRIGGER BEF_UPD_address_logs
BEFORE UPDATE ON address
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "address", "Update", NOW(), USER(), DATABASE(), VERSION());

CREATE TRIGGER BEF_UPD_account_logs
BEFORE UPDATE ON account
FOR EACH ROW
INSERT INTO logs VALUES (NULL, "account", "Update", NOW(), USER(), DATABASE(), VERSION());
```