UD5 - EXAMEN DE OPTIMIZACIÓN DE BASES DE DATOS (MySQL)

1. Contexto Real

Un marketplace de tecnología en plena expansión, está sufriendo graves problemas de rendimiento en su plataforma. La base de datos actual fue creada de forma rápida por un equipo que no aplicó criterios de optimización. Actualmente, con cerca de 1 millón de registros, las búsquedas de clientes tardan segundos, el servidor de disco está saturado y el proceso de generación de reportes mensuales bloquea el sistema.

Tu objetivo como experto en bases de datos es auditar, proponer cambios estructurales y demostrar con métricas la mejora de rendimiento tras la optimización.

2. Estructura Ineficiente (Script Inicial)

Ejecuta el script para crear el escenario base.

mysql -u root -p < script.sql

o importando por phpmyadmin

3. Consultas más frecuentes (Workload)

Estas son las consultas que el departamento de ventas y atención al cliente ejecuta constantemente y que actualmente van muy lentas:

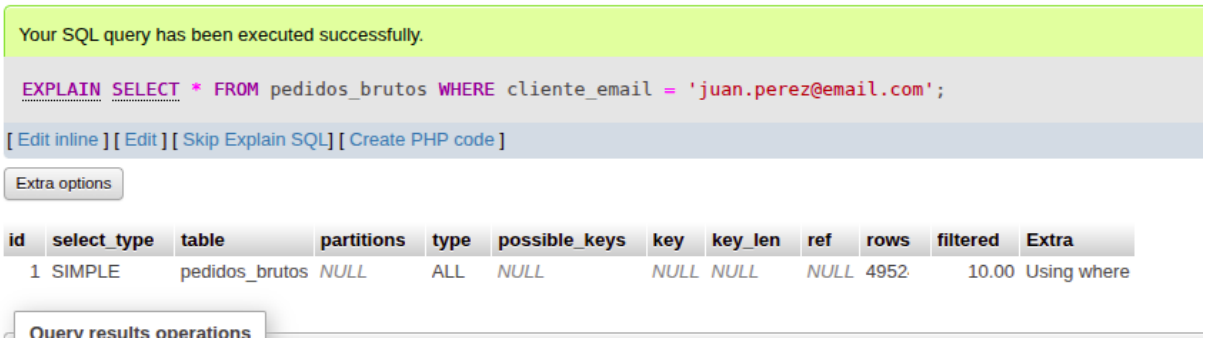
1. **Búsqueda de pedidos por cliente:**
   ```
   SELECT * FROM pedidos_brutos WHERE cliente_email = 'juan.perez@email.com';
   ```
2. **Filtrado por categoría y rango de precios:**
   ```
   SELECT producto_nombre, precio_unitario FROM pedidos_brutos WHERE categoria_nombre = 'Laptops' AND precio_unitario > 1000 ORDER BY precio_unitario DESC;
   ```
3. **Búsqueda de palabras clave en la descripción:**
   ```
   SELECT producto_nombre FROM pedidos_brutos WHERE comentario_producto LIKE '%potente%';
   ```
4. **Reporte de ventas totales por país:**
   ```
   SELECT codigo_pais, SUM(precio_unitario * cantidad_pedida) FROM pedidos_brutos GROUP BY pais_codigo;
   ```

4. Tareas del Alumno

Fase 1: Análisis Inicial (2 puntos)

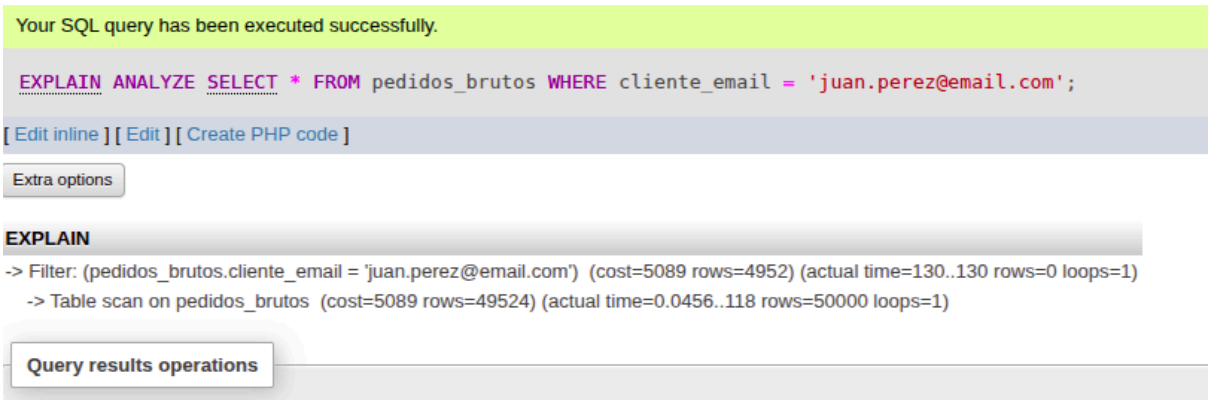
- Ejecuta las consultas anteriores usando `EXPLAIN` y `EXPLAIN ANALYZE`.
- Documenta los puntos críticos.

EXPLAIN SELECT * FROM pedidos_brutos WHERE cliente_email = 'juan.perez@email.com';



En Type es ALL, lo que significa que la base de datos tiene que leer las 50k columnas y no cuenta con ninguna key

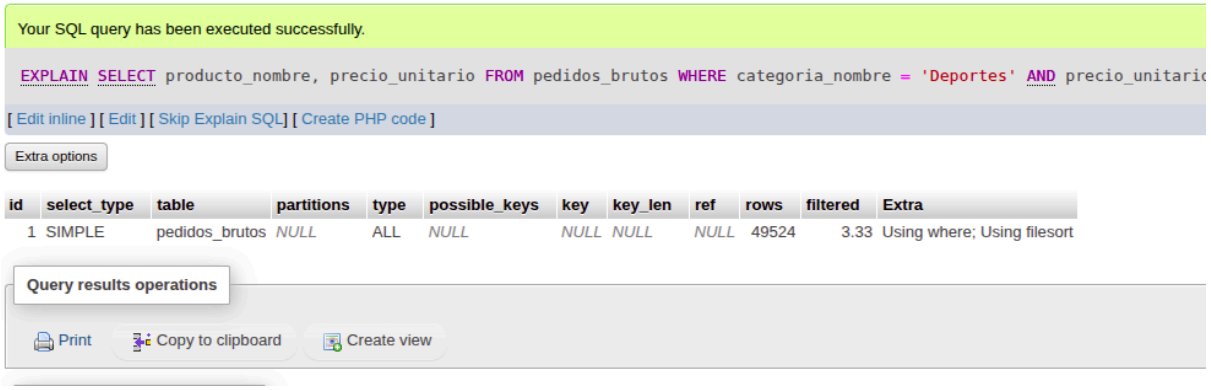
EXPLAIN ANALYZE SELECT * FROM pedidos_brutos WHERE cliente_email = 'juan.perez@email.com';



En este explain analizy se puede ver el gran coste que tiene, de 5089. deberia tener un indice

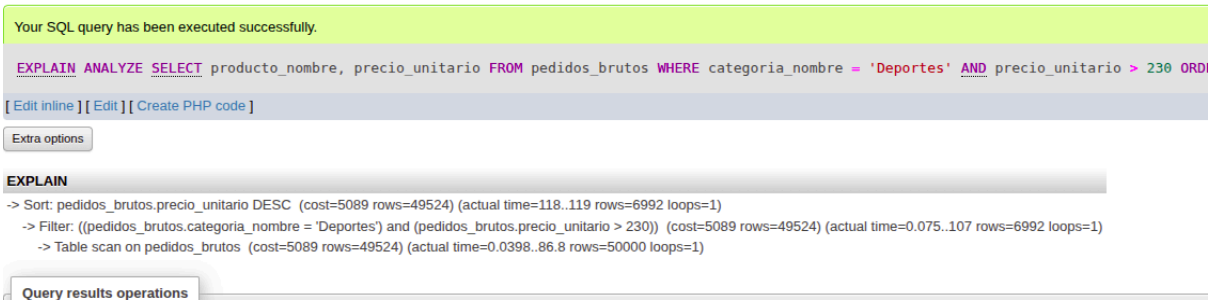
SELECT producto_nombre, precio_unitario FROM pedidos_brutos WHERE categoria_nombre = 'Deportes' AND precio_unitario > 70 ORDER BY precio_unitario DESC;

EXPLAIN SELECT producto_nombre, precio_unitario FROM pedidos_brutos WHERE categoria_nombre = 'Deportes' AND precio_unitario > 230 ORDER BY precio_unitario DESC;



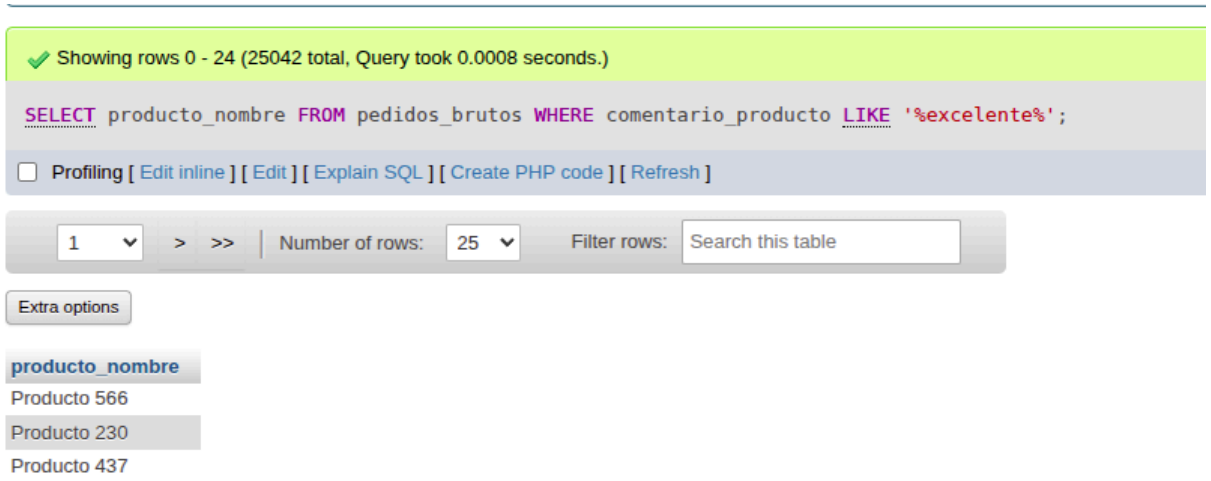
Se puede ver como tiene que volver a leer las 50k rows.

EXPLAIN ANALYZE SELECT producto_nombre, precio_unitario FROM pedidos_brutos WHERE categoria_nombre = 'Deportes' AND precio_unitario > 230 ORDER BY precio_unitario DESC;



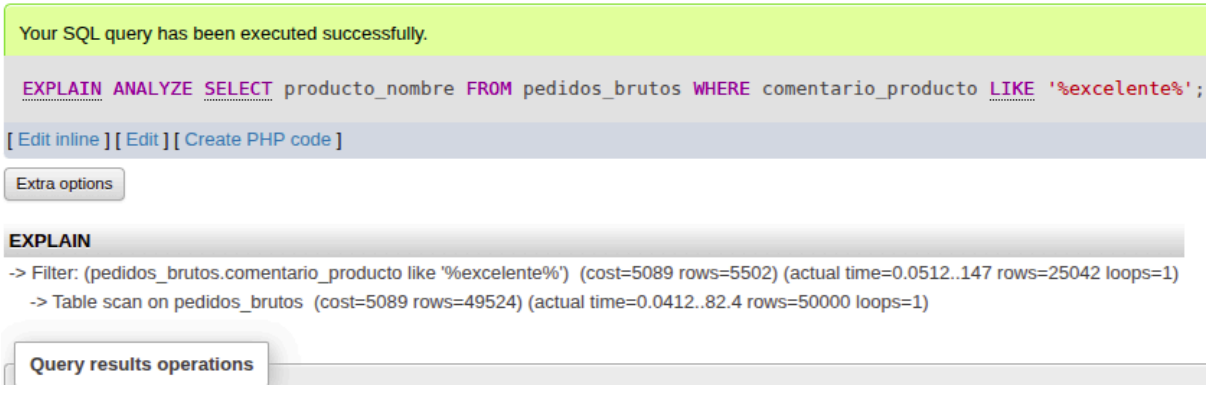
en el explain analyze se puede ver nuevamente el elevando coste de cada accion

SELECT producto_nombre FROM pedidos_brutos WHERE comentario_producto LIKE '%excelente%';

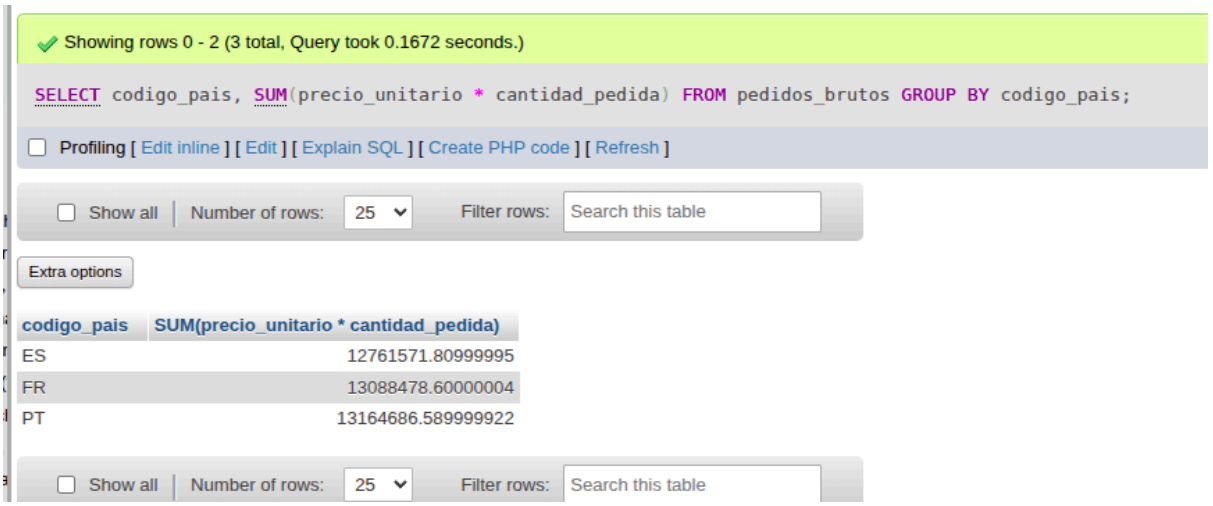


EXPLAIN SELECT producto_nombre FROM pedidos_brutos WHERE comentario_producto LIKE '%excelente%';

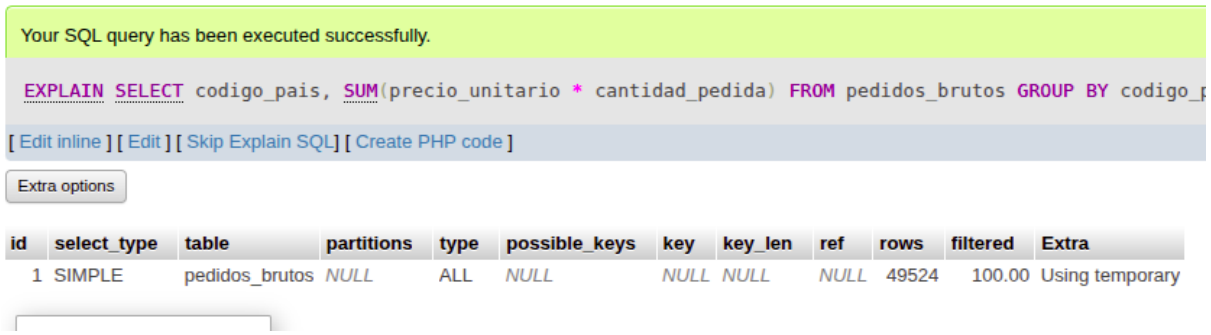
mas de lo mismo, deberia existir una tabla para la reseñas en este caso.



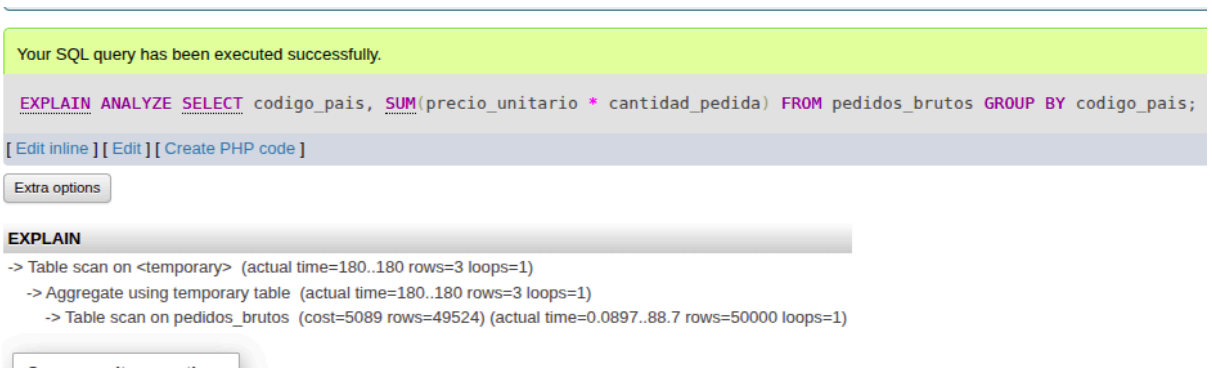
SELECT codigo_pais, SUM(precio_unitario * cantidad_pedida) FROM pedidos_brutos GROUP BY codigo_pais;



EXPLAIN SELECT codigo_pais, SUM(precio_unitario * cantidad_pedida) FROM pedidos_brutos GROUP BY codigo_pais;



ANALYZE EXPLAIN SELECT codigo_pais, SUM(precio_unitario * cantidad_pedida) FROM pedidos_brutos GROUP BY codigo_pais;



Fase 2: Optimización Estructural (8 puntos)

- Tras el diagnóstico inicial, transforma la tabla masiva `pedidos_brutos` en una estructura relacional eficiente.
- Explica detalladamente el motivo de cada cambio realizado (por qué esa tabla, por qué ese tipo de dato o esa relación)
- Aplica el nuevo script a la base de datos.

Tras analizar la tabla se puede ver que prácticamente todos los campos son varchar(255) o contienen datos redundantes. Esto es altamente ineficiente, ya que si buscas por id_operacion por ejemplo buscar un varchar tiene un coste mucho más elevado que buscar por int

```
CREATE TABLE clientes (

id_cliente INT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(32) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL

);
```

Creamos una tabla por clientes, con un ID que usa INT y que se auto-incrementa

nombre, un varchar de 32, suficiente para un nombre

email un varchar de 100, único, para que no existan conflictos

```
CREATE TABLE categorias (

id_categoria INT AUTO_INCREMENT PRIMARY KEY,

nombre_categoria VARCHAR(100) NOT NULL

);
```

las categorías eran redundante y se repetía todo el rato, mejor creamos un tabla a parte y que se referencie, con un INT auto_increment y un varchar de 100

```
CREATE TABLE productos (
id_producto INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(150) NOT NULL,
precio_unitario DECIMAL(10, 2) NOT NULL,
id_categoria INT,
FOREIGN KEY (id_categoria) REFERENCES Categorias(id_categoria)
);
```

tabla para cada producto, un id int con auto incrementación,

nombre un varchar de 150, suficiente para los productos.

precio unitario: un decimal de lo que cuesta cada uno,

id_Categoria: hace referencia a la tabla categoría.

```
CREATE TABLE reviews (
id INT AUTO_INCREMENT PRIMARY KEY,
id_producto INT,
id_cliente INT,
comentario TEXT,
FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente),
FOREIGN KEY (id_producto) REFERENCES Productos(id_producto)
);
```

Tabla reviews para los comentarios que hacen los clientes. tiene un id autoincremental para cada review, un id producto referenciando a la tabla productos para conocer cada producto, y un id_cliente para conocer qué cliente hace la review, que evidentemente será un int haciendo referencia a la tabla clientes.

Y por último, comentario, campo texto, para almacenar el comentario del cliente.

```
CREATE TABLE paises(
id_pais INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(16),
siglas VARCHAR(2)
)
```

tabla para los países, ya que estaba siendo redundante en todos los pedidos

```
CREATE TABLE pedidos (
id_pedido INT AUTO_INCREMENT PRIMARY KEY,
id_cliente INT,
fecha DATETIME DEFAULT CURRENT_TIMESTAMP,
id_pais INT,
FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente),
FOREIGN KEY (id_pais) REFERENCES paises(id_pais),
);
```

La tabla para los pedidos, con un id que identifica cada pedido, usando int nuevamente y autocrimental, id_cliente, FK para conocer el cliente que realiza el pedido.

fecha: un campo DATE, que optimiza la búsqueda vs el varchar que teníamos antes y que era muy ineficiente.

```
CREATE TABLE detalles_Pedido (

id_detalle INT AUTO_INCREMENT PRIMARY KEY,

id_pedido INT,

id_producto INT,

cantidad INT NOT NULL,

FOREIGN KEY (id_pedido) REFERENCES Pedidos(id_pedido),

FOREIGN KEY (id_producto) REFERENCES Productos(id_producto)

);
```

Creamos una tabla para los detalles de cada pedido, con los productos pedidos y su cantidad, y haciendo las correspondientes referencias.

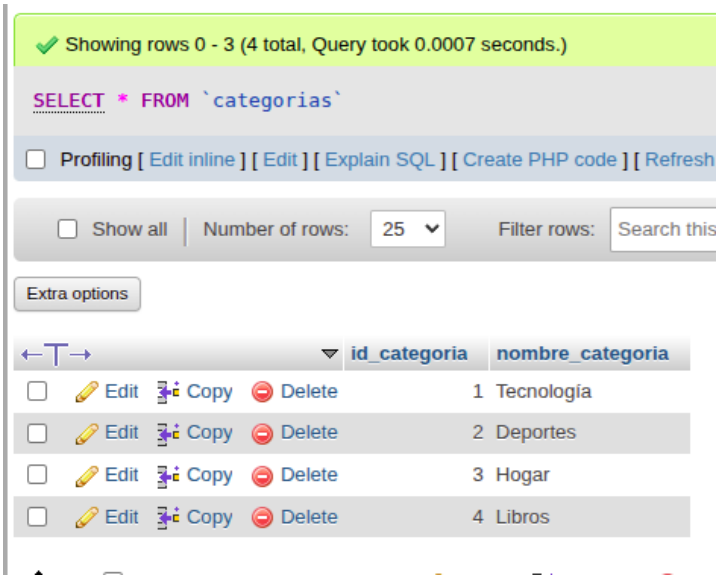
CREACIÓN DE ÍNDICES.

```
CREATE INDEX idx_fecha ON pedidos(fecha);

CREATE INDEX idx_email ON Clientes(email);

CREATE INDEX idx_pais ON pedidos(codigo_paisl);
```

CARGAR DATOS:

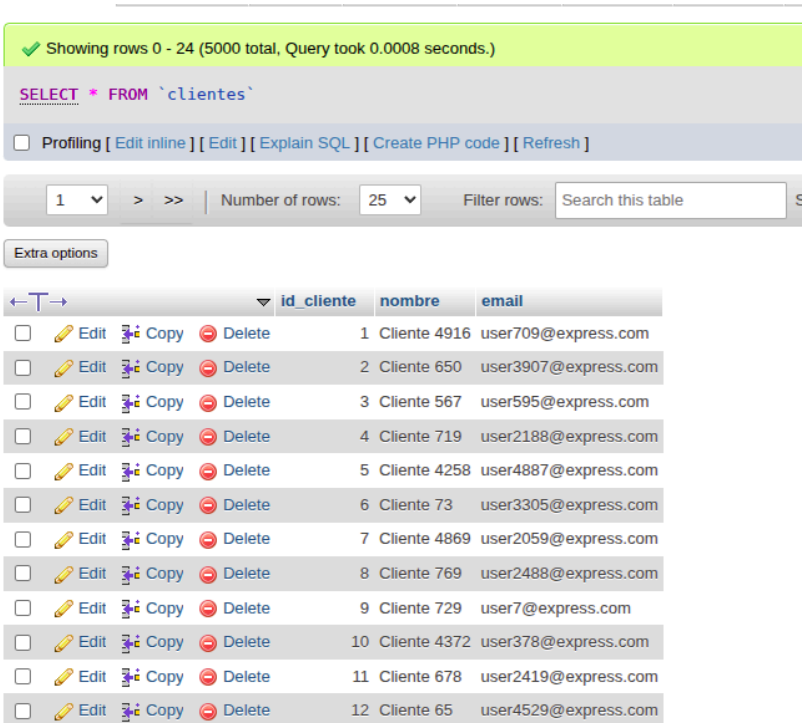
CATEGORÍAS:

```
INSERT INTO categorias (nombre_categoria)
SELECT DISTINCT categoria_nombre
FROM pedidos_brutos;
```



CLIENTES

INSERT INTO clientes (nombre, email) SELECT MAX(cliente_nombre), cliente_email FROM pedidos_brutos GROUP BY cliente_email;



Showing rows 0 - 24 (5000 total, Query took 0.0008 seconds.)

```
SELECT * FROM `clientes`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Number of rows: 25 Filter rows: Search this table

Extra options

	id_cliente	nombre	email
Edit Copy Delete	1	Cliente 4916	user709@express.com
Edit Copy Delete	2	Cliente 650	user3907@express.com
Edit Copy Delete	3	Cliente 567	user595@express.com
Edit Copy Delete	4	Cliente 719	user2188@express.com
Edit Copy Delete	5	Cliente 4258	user4887@express.com
Edit Copy Delete	6	Cliente 73	user3305@express.com
Edit Copy Delete	7	Cliente 4869	user2059@express.com
Edit Copy Delete	8	Cliente 769	user2488@express.com
Edit Copy Delete	9	Cliente 729	user7@express.com
Edit Copy Delete	10	Cliente 4372	user378@express.com
Edit Copy Delete	11	Cliente 678	user2419@express.com
Edit Copy Delete	12	Cliente 65	user4529@express.com

PRODUCTOS

INSERT INTO productos (nombre,precio_unitario id_categoria) SELECT MAX(pb.producto_nombre), c.id_categoria FROM pedidos_brutos pb JOIN categorias c ON pb.categoria_nombre = c.nombre GROUP BY pb.producto_nombre, c.id_categoria ;