

# 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

```
✓ Import has been successfully finished. 6 queries executed. (script.sql)

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

CREATE DATABASE global_express_db;
[ Edit inline ] [ Edit ] [ Create PHP code ]

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

USE global_express_db;
[ Edit inline ] [ Edit ] [ Create PHP code ]

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

-- 1. Creación de la tabla ineficiente CREATE TABLE pedidos_brutos ( id_operacion VARCHAR(255), -- Debería ser INT PK fecha_operacion VARCHAR(255), -- Debería ser DATETIME cliente_nombre VARCHAR(255), -- Redundante cliente_email VARCHAR(255), -- Redundante producto_nombre VARCHAR(255), -- Redundante categoria_nombre VARCHAR(255), -- Redundante precio_unitario VARCHAR(255), -- Debería ser DECIMAL cantidad_pedido VARCHAR(255), -- Debería ser INT comentario_producto TEXT, -- Para búsquedas de texto codigo_pais VARCHAR(255) -- Debería ser CHAR(2) );
[ Edit inline ] [ Edit ] [ Create PHP code ]

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0101 seconds.)

CREATE PROCEDURE CargarDatosExamen() BEGIN DECLARE i INT DEFAULT 1; WHILE i <= 50000 DO INSERT INTO pedidos_brutos VALUES ( i, DATE_FORMAT(DATE_ADD('2023-01-01', INTERVAL FLOOR(RAND() * 365) DAY), '%d/%m/%Y %H:%i:%s'), CONCAT('Cliente ', FLOOR(RAND() * 5000)), -- Muchos pedidos por cliente CONCAT('user', FLOOR(RAND() * 5000), '@express.com'), CONCAT('Producto ', FLOOR(RAND() * 1000)), CASE FLOOR(RAND() * 4) WHEN 0 THEN 'Tecnología' WHEN 1 THEN 'Hogar' WHEN 2 THEN 'Deportes' ELSE 'Libros' END, ROUND(10 + RAND() * 500, 2), FLOOR(1 + RAND() * 5), CONCAT('Opinión del producto: ', IF(RAND() > 0.5, 'Excelente calidad y muy rápido', 'No me gustó mucho el acabado')), CASE FLOOR(RAND() * 3) WHEN 0 THEN 'ES' WHEN 1 THEN 'FR' [...] );
[ Edit ]

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 141.7493 seconds.)

-- 3. Ejecutar la carga y limpiar CALL CargarDatosExamen();
[ Edit inline ] [ Edit ] [ Create PHP code ]
```

### 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';

Your SQL query has been executed successfully.

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

[ Edit inline ] [ Edit ] [ Skip Explain SQL ] [ Create PHP code ]

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	pedidos_brutos	NULL	ALL	NULL	NULL	NULL	NULL	4952	10.00	Using where

Query results operations

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';

Your SQL query has been executed successfully.

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

[ Edit inline ] [ Edit ] [ Create PHP code ]

Extra options

**EXPLAIN**

-> Filter: (pedidos\_brutos.cliente\_email = 'juan.perez@email.com') (cost=5089 rows=4952) (actual time=130..130 rows=0 loops=1)  
-> Table scan on pedidos\_brutos (cost=5089 rows=49524) (actual time=0.0456..118 rows=50000 loops=1)

Query results operations

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

SELECT producto\_nombre, precio\_unitario FROM pedidos\_brutos WHERE categoria\_nombre = 'Deportes' AND precio\_unitario > 70 ORDER BY precio\_unitario DESC;

Showing rows 0 - 24 (6992 total, Query took 0.1166 seconds.) [precio\_unitario: 509.99... - 508.78...]

SELECT producto\_nombre, precio\_unitario FROM pedidos\_brutos WHERE categoria\_nombre = 'Deportes' AND precio\_unitario > 230 ORDER BY precio\_unitario DESC;

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

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

Extra options

producto_nombre	precio_unitario
Producto 173	509.99
Producto 582	509.98
Producto 182	509.96
Producto 293	509.9
Producto 141	509.73
Producto 528	509.62
Producto 410	509.61

EXPLAIN SELECT producto\_nombre, precio\_unitario FROM pedidos\_brutos WHERE categoria\_nombre = 'Deportes' AND precio\_unitario > 230 ORDER BY precio\_unitario DESC;

Your SQL query has been executed successfully.

EXPLAIN SELECT producto\_nombre, precio\_unitario FROM pedidos\_brutos WHERE categoria\_nombre = 'Deportes' AND precio\_unitario > 230 ORDER BY precio\_unitario DESC;

[ Edit inline ] [ Edit ] [ Skip Explain SQL ] [ Create PHP code ]

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	pedidos_brutos	NULL	ALL	NULL	NULL	NULL	NULL	49524	3.33	Using where; Using filesort

Query results operations

Print Copy to clipboard Create view

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;

Your SQL query has been executed successfully.

EXPLAIN ANALYZE SELECT producto\_nombre, precio\_unitario FROM pedidos\_brutos WHERE categoria\_nombre = 'Deportes' AND precio\_unitario > 230 ORDER BY precio\_unitario DESC;

[ Edit inline ] [ Edit ] [ Create PHP code ]

Extra options

**EXPLAIN**

-> Sort: pedidos\_brutos.precio\_unitario DESC (cost=5089 rows=49524) (actual time=118..119 rows=6992 loops=1)

-> Filter: ((pedidos\_brutos.categoria\_nombre = 'Deportes') and (pedidos\_brutos.precio\_unitario > 230)) (cost=5089 rows=49524) (actual time=0.075..107 rows=6992 loops=1)

-> Table scan on pedidos\_brutos (cost=5089 rows=49524) (actual time=0.0398..86.8 rows=50000 loops=1)

Query results operations

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%';

✓ Showing rows 0 - 24 (25042 total, Query took 0.0008 seconds.)

SELECT producto\_nombre FROM pedidos\_brutos WHERE comentario\_producto LIKE '%excelente%';

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

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

Extra options

**producto\_nombre**  
Producto 566  
Producto 230  
Producto 437

EXPLAIN SELECT producto\_nombre FROM pedidos\_brutos WHERE comentario\_producto LIKE '%excelente%';

Your SQL query has been executed successfully.

EXPLAIN SELECT producto\_nombre FROM pedidos\_brutos WHERE comentario\_producto LIKE '%excelente%';

[ Edit inline ] [ Edit ] [ Skip Explain SQL ] [ Create PHP code ]

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	pedidos_brutos	NULL	ALL	NULL	NULL	NULL	NULL	49524	11.11	Using where

Query results operations

[Print](#) [Copy to clipboard](#) [Create view](#)

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

Your SQL query has been executed successfully.

EXPLAIN ANALYZE SELECT producto\_nombre FROM pedidos\_brutos WHERE comentario\_producto LIKE '%excelente%';

[ Edit inline ] [ Edit ] [ Create PHP code ]

Extra options

**EXPLAIN**  
-> Filter: (pedidos\_brutos.comentario\_producto like '%excelente%') (cost=5089 rows=5502) (actual time=0.0512..147 rows=25042 loops=1)  
-> Table scan on pedidos\_brutos (cost=5089 rows=49524) (actual time=0.0412..82.4 rows=50000 loops=1)

Query results operations

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

✓ Showing rows 0 - 2 (3 total, Query took 0.1672 seconds.)

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

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

☐ Show all | Number of rows: 25 | Filter rows:

Extra options

codigo_pais	SUM(precio_unitario * cantidad_pedida)
ES	12761571.80999995
FR	13088478.60000004
PT	13164686.589999922

☐ Show all | Number of rows: 25 | Filter rows:

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

Your SQL query has been executed successfully.

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

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Skip Explain SQL \]](#) [\[ Create PHP code \]](#)

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	pedidos_brutos	NULL	ALL	NULL	NULL	NULL	NULL	49524	100.00	Using temporary

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

Your SQL query has been executed successfully.

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

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)

Extra options

**EXPLAIN**

```
-> Table scan on <temporary> (actual time=180..180 rows=3 loops=1)
-> Aggregate using temporary table (actual time=180..180 rows=3 loops=1)
    -> Table scan on pedidos_brutos (cost=5089 rows=49524) (actual time=0.0897..88.7 rows=50000 loops=1)
```

## 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 una 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;
```

✓ Showing rows 0 - 3 (4 total, Query took 0.0007 seconds.)

SELECT \* FROM `categorias`

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

☐ Show all | Number of rows: 25 | Filter rows:

Extra options

				id_categoria	nombre_categoria
<input type="checkbox"/>	Edit	Copy	Delete	1	Tecnología
<input type="checkbox"/>	Edit	Copy	Delete	2	Deportes
<input type="checkbox"/>	Edit	Copy	Delete	3	Hogar
<input type="checkbox"/>	Edit	Copy	Delete	4	Libros

## 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:

Extra options

				id_cliente	nombre	email
<input type="checkbox"/>				1	Cliente 4916	user709@express.com
<input type="checkbox"/>				2	Cliente 650	user3907@express.com
<input type="checkbox"/>				3	Cliente 567	user595@express.com
<input type="checkbox"/>				4	Cliente 719	user2188@express.com
<input type="checkbox"/>				5	Cliente 4258	user4887@express.com
<input type="checkbox"/>				6	Cliente 73	user3305@express.com
<input type="checkbox"/>				7	Cliente 4869	user2059@express.com
<input type="checkbox"/>				8	Cliente 769	user2488@express.com
<input type="checkbox"/>				9	Cliente 729	user7@express.com
<input type="checkbox"/>				10	Cliente 4372	user378@express.com
<input type="checkbox"/>				11	Cliente 678	user2419@express.com
<input type="checkbox"/>				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 ;
```