

# Consultas Analíticas 10 veces más rápidas



**Pierre-Yves Duquesnoy**

Sales Engineer

[Pierre-Yves.Duquesnoy@intersystems.com](mailto:Pierre-Yves.Duquesnoy@intersystems.com)

**InterSystems Global Summit**

June 22, 2022

# Agenda

---

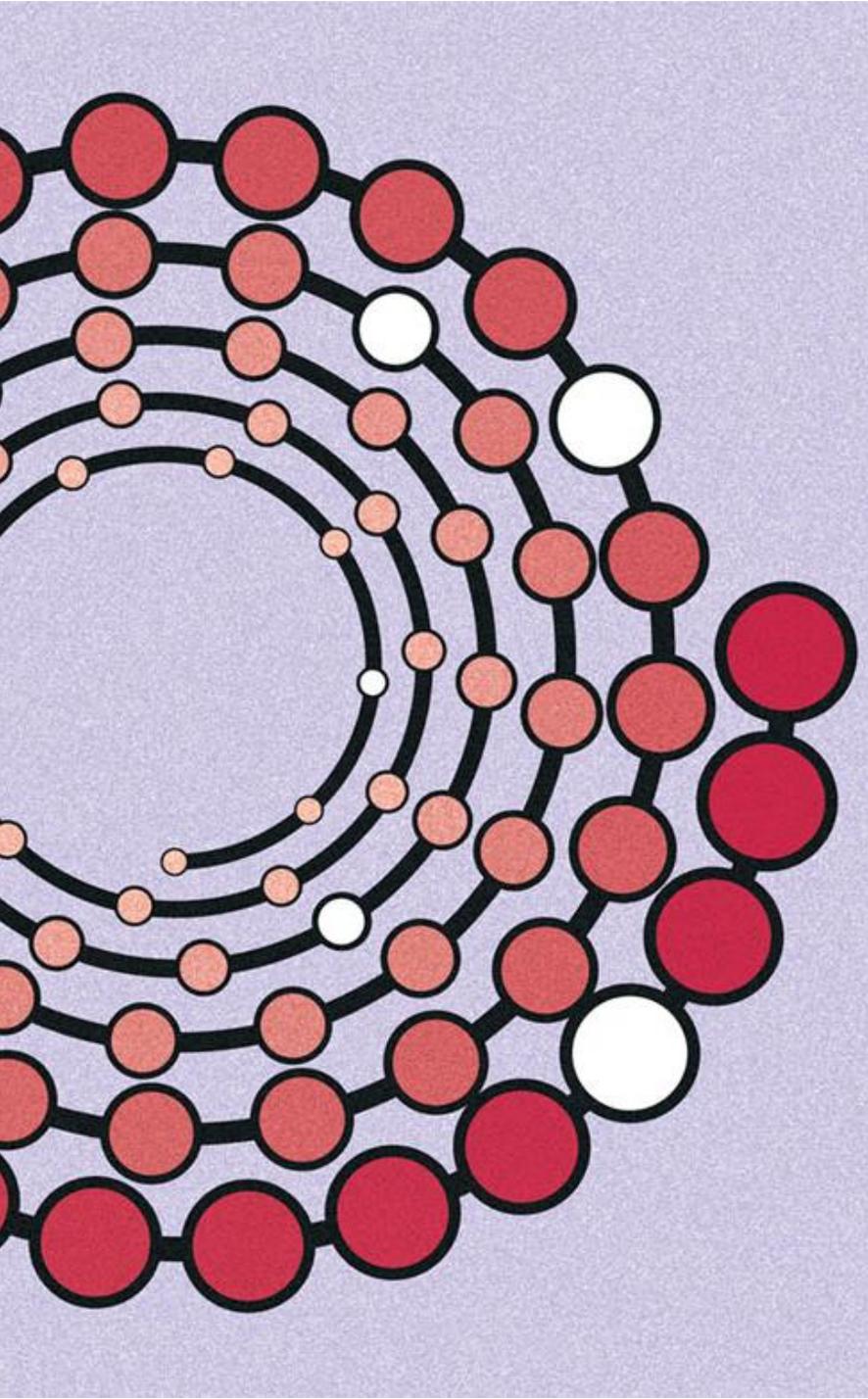


- 1. Puesta en escena**
- 2. Matar 64000 pájaros de un tiro**
- 3. Beneficios adicionales**
- 4. Sintaxis**
- 5. Demo**
- 6. Hoja de ruta**

# Puesta en escena

---

**Consultas 10 X más rápidas con almacenamiento en columnas**



# Aplicaciones

---

```
SELECT TOP 10 * FROM tx WHERE acct = 123  
ORDER BY txTime DESC
```

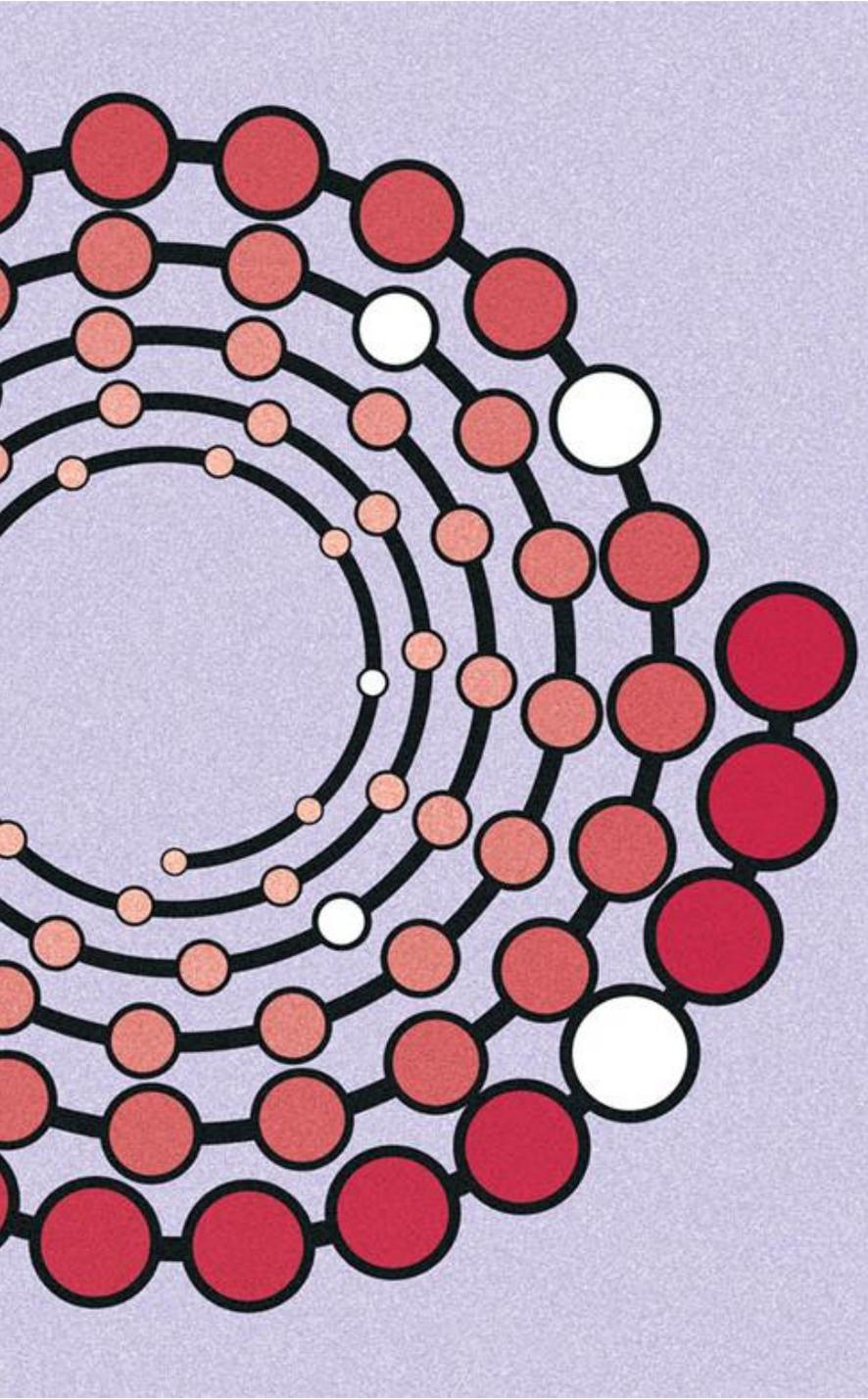
## START TRANSACTION

```
INSERT INTO tx (txTime, acct, type, amount)  
VALUES ( NOW(), 123, 'DEBIT', -1000 );  
INSERT INTO tx (txTime, acct, type, amount)  
VALUES ( NOW(), 456, 'CREDIT', 1000 );
```

```
UPDATE acct SET balance = balance - 1000  
WHERE ID = 123;
```

```
UPDATE acct SET balance = balance + 1000  
WHERE ID = 456;
```

## COMMIT



# Aplicaciones

---

**Velocidad de Inserciones & Modificaciones**

**Acceso a filas completas**

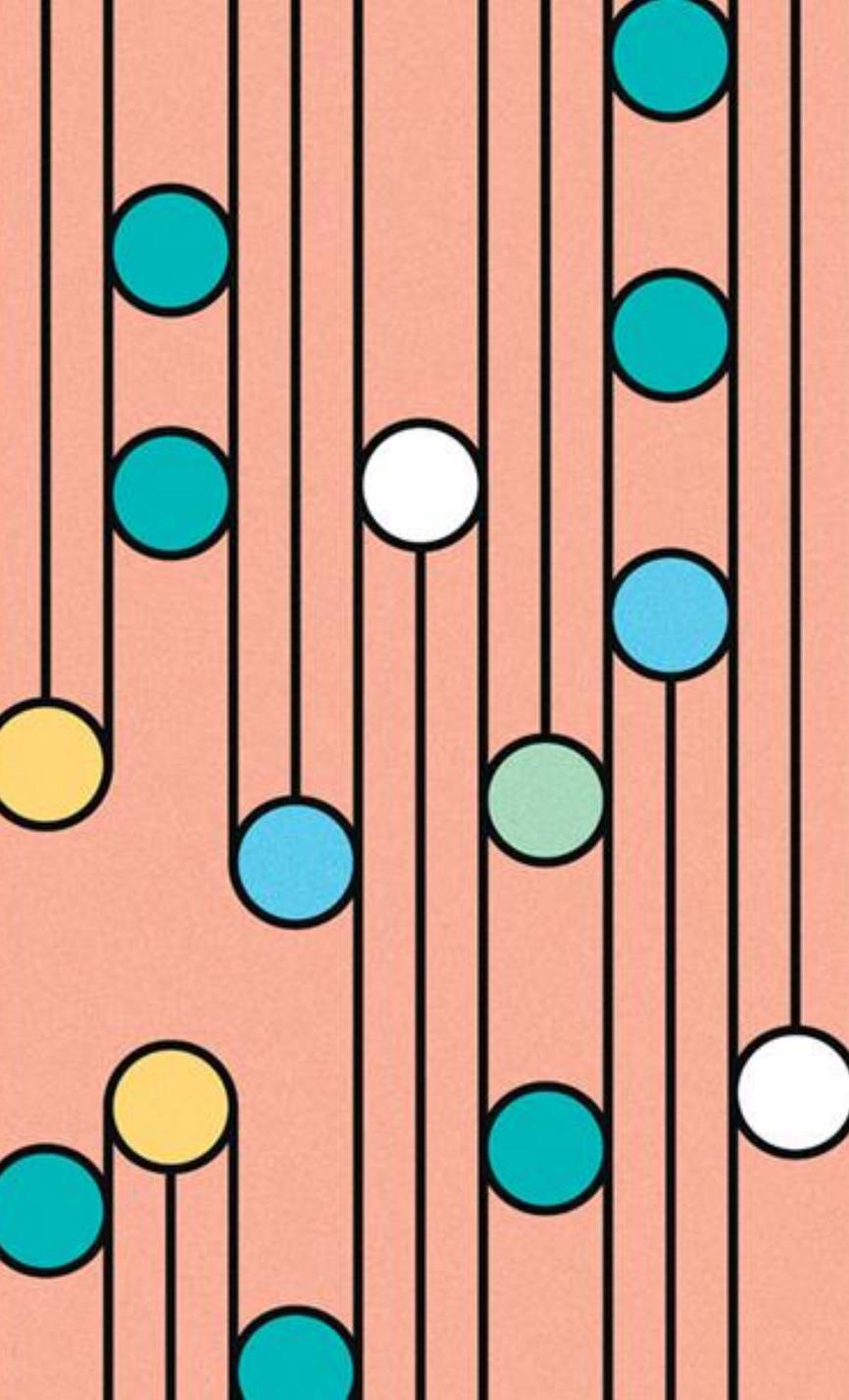
**Foco en la **latencia****

**Almacena datos como se usan:  
**fila a fila****

# Peticiones Analíticas

---

```
SELECT MONTH(txTime) AS TxMonth,  
       type           AS TxType,  
       AVG(amount)   AS AverageAmount,  
       MAX(amount)   AS MaxAmount  
  
FROM tx  
  
WHERE acct = 123  
      AND txTime > DATEADD('YY', -1, NOW())  
  
GROUP BY MONTH(txTime),  
         type
```



# Peticiones Analíticas

---

**Peticiones complejas sobre tablas grandes**

**Devuelven agregados en vez de filas**

**Foco en **rendimiento****

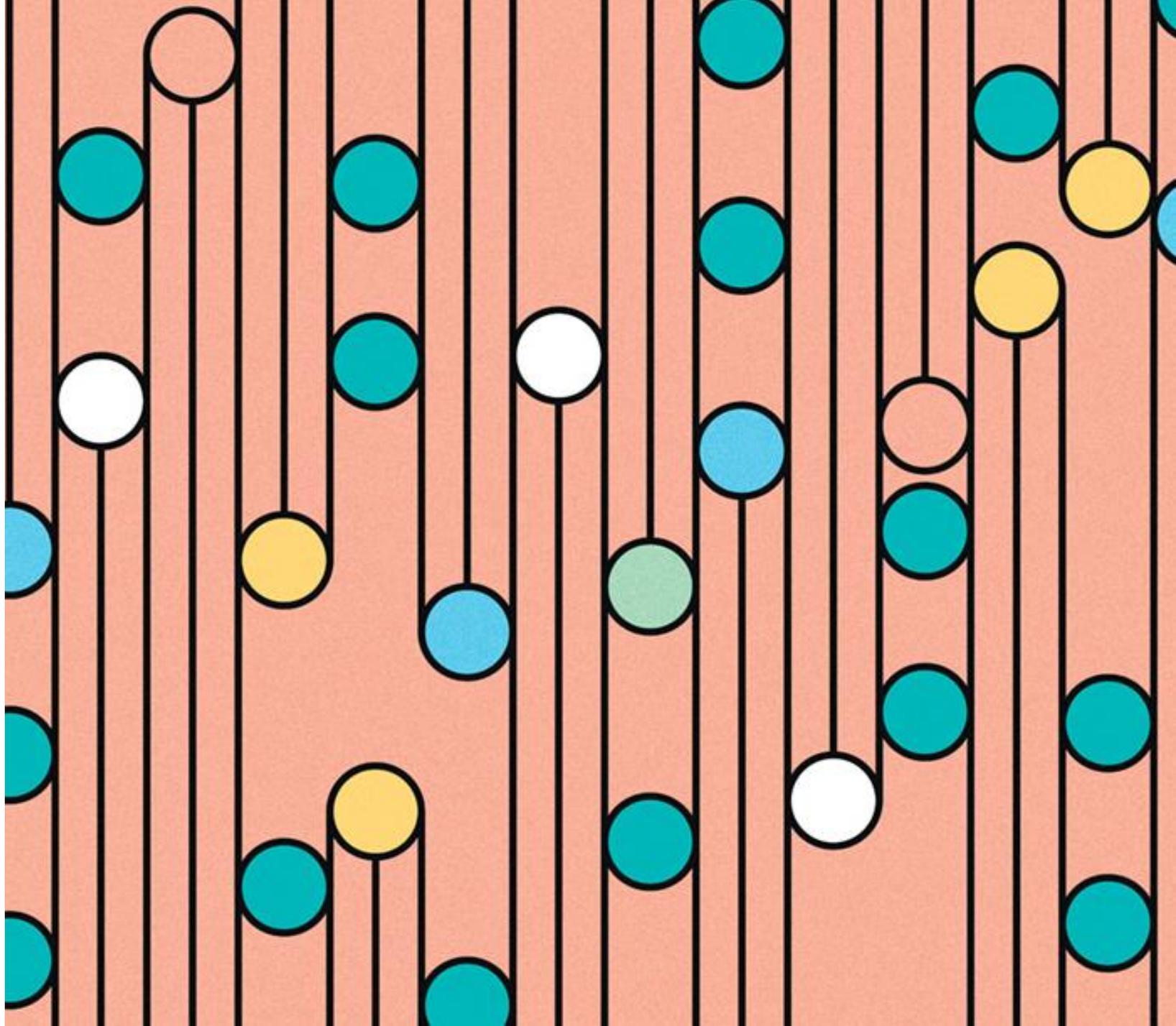
# Limitaciones

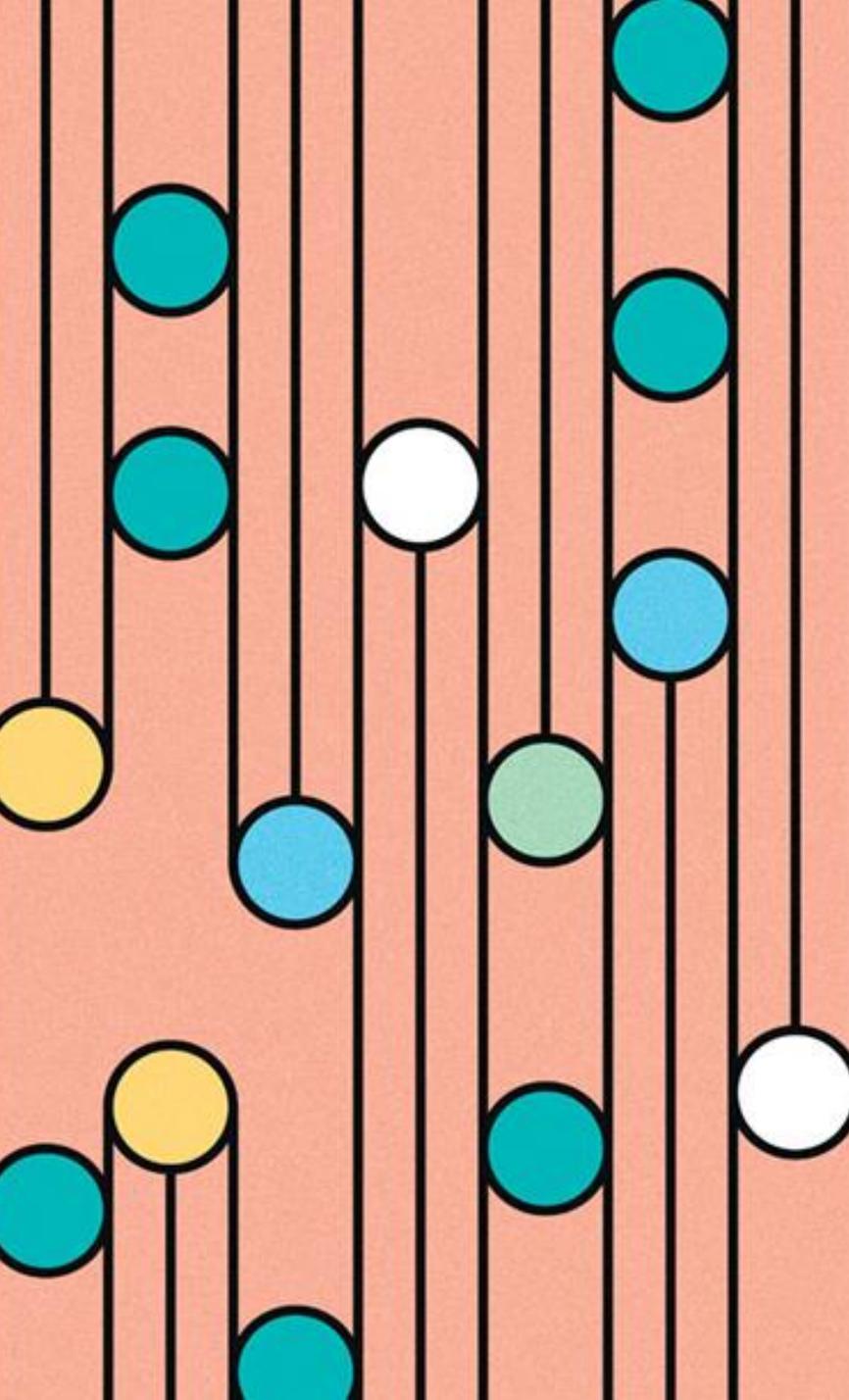
---

**88,000,000**

**40,000**

**10**





# Peticiones Analíticas

---

**Peticiones complejas sobre tablas grandes**

**Devuelven agregados en vez de filas**

**Foco en rendimiento**

**Almacenar los datos como se usan:  
columna a columna**

# Matar 64000 pájaros de un tiro

---

Consultas 10 X más rápidas con almacenamiento en columnas

# Bitmaps: los pioneros

ID	Producto	Ciudad	Cantidad	...
1	Mac	Valencia	1	
2	Cable	Valencia	2	
3	Ratón	Madrid	1	
4	Bolsa	Valencia	3	
5	Cable	Madrid	1	

Index tradicional

Ciudad	ID
Madrid	3
Madrid	5
Valencia	1
Valencia	2
Valencia	4

Index BITMAP

Ciudad	ID
Madrid	00101 (3,5)
Valencia	11010 (1,2,4)

# Bitmaps: los pioneros

Los **Índices Bitmap** fueron pioneros en los conceptos clave necesarios para un procesamiento de consultas analítico eficiente

- Junta datos de múltiples filas en una operación de IO
- Procesa varias filas en 1 operación

## Regular Index

```
^idx("Madrid", 3) = ""
^idx("Madrid", 5) = ""
^idx("Valencia", 1) = ""
^idx("Valencia", 2) = ""
^idx("Valencia", 4) = ""
..
^idx("Valencia", 64008)=""
^idx("Valencia", 64009)=""
```

## Bitmap Index

```
^idx("Madrid",1)= $bit(3, 5)
^idx("Valencia",1) = $bit(1,2,4)
..
^idx("Valencia", 2) = $bit(8, 9)
```



# Bitmaps: los pioneros

---

**\$bit**: tipo de datos dedicado a secuencias de bits, usado en índices bitmaps

## Almacenamiento Optimizado

- Estructura interna de \$bit permite compresión
- Bloques de 64 000 bits

## Cómputo Optimizado

- Operaciones dedicadas de lógica booleana & transversal
- Soporte para actualizaciones atómicas, ECP y Journal

# Almacenamiento Optimizado

Logica

ID	Producto	Ciudad	Cantidad	...
1	Mac	Valencia	1	
2	Cable	Valencia	2	
3	Ratón	Madrid	1	
4	Bolsa	Valencia	<null>	
5	Cable	Madrid	1	

En Filas

Física

```
^d(1) = $list("Mac", "Valencia", 1, ...)  
^d(2) = $list("Cable", "Valencia", 2, ...)  
^d(3) = $list("Ratón", "Madrid", 1, ...)  
^d(4) = $list("Bolsa", "Valencia",  
<null>, ...)  
^d(5) = $list("Cable", "Madrid", 1, ...)
```

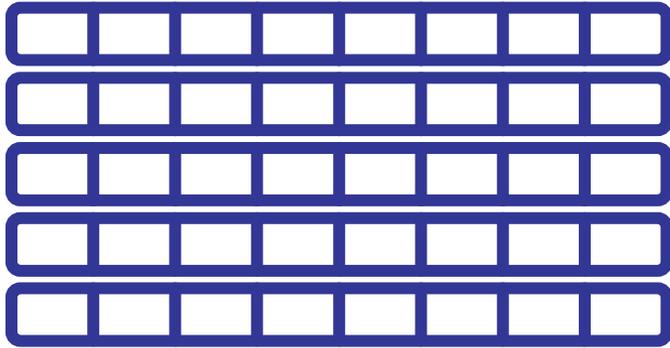
En Columnas

```
^d.V1(1) = $vector(<string>: "Mac", "Cable",  
"Ratón", "Bolsa", "Cable", ...)  
^d.V2(1) = $vector(<string>:  
"Valencia", "Valencia", "Madrid", "Valencia",  
"Madrid", ...)  
^d.V3(1) = $vector(<integer>: 1, 2, 1,  
<null>, 1, ...)
```

# Almacenamiento Optimizado

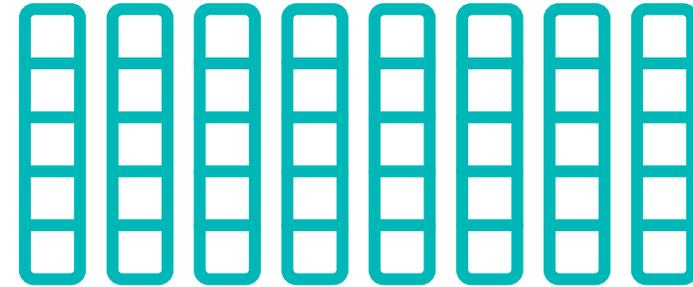
---

## En Filas

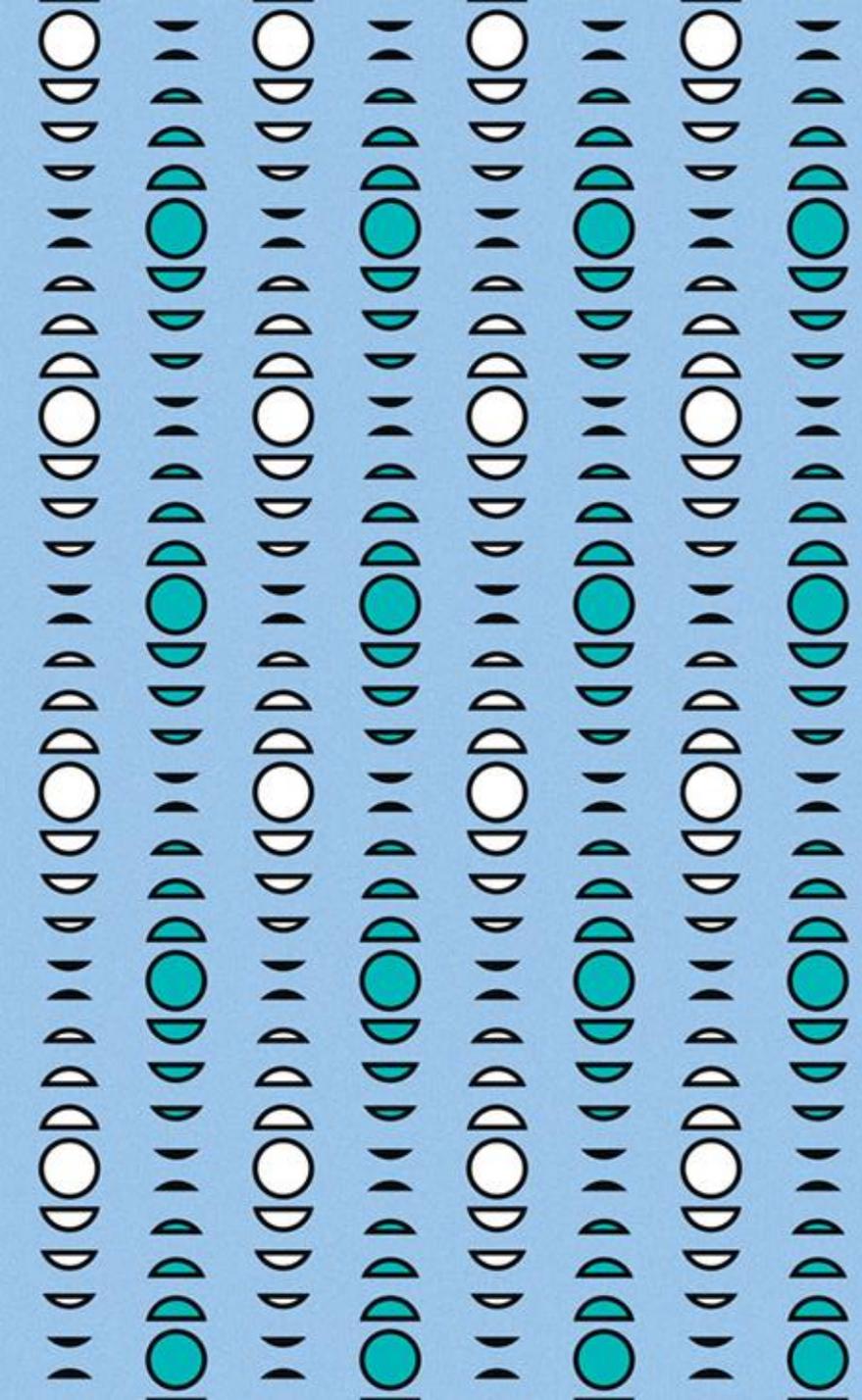


- Agrupado para acceso puntual: **Latencia**
- Juntado para acceso dinámico: **Flexibilidad**
- Cache de todos los campos de una pocas filas: **Transacciones**

## En Columnas



- Agrupado para E/S masiva: **Rendimiento**
- Juntado para un acceso predecible: **Rendimiento**
- Cache para algunos campos de muchas filas: **Rendimiento**



# \$vector

---

Nuevo tipo de datos para almacenar grandes listas de valores del mismo tipo

Gestión eficiente de datos dispersos

- Distinción interna entre regiones densas y dispersas utilizando run-length encoding

**Codificación eficiente dependiente del tipo de datos**

- Codificación de diccionario para Strings
- Longitud adaptiva para Integers

Soporte para actualizaciones **atómicas y masivas**

- Incluyendo ECP y Journaling

# Computo Optimizado

---

La CPUs modernas están optimizadas para **bucles cortos**:

```
for (i = 0; i < BUF_SIZE; i++) { c[i] = a[i] + b[i]; }
```

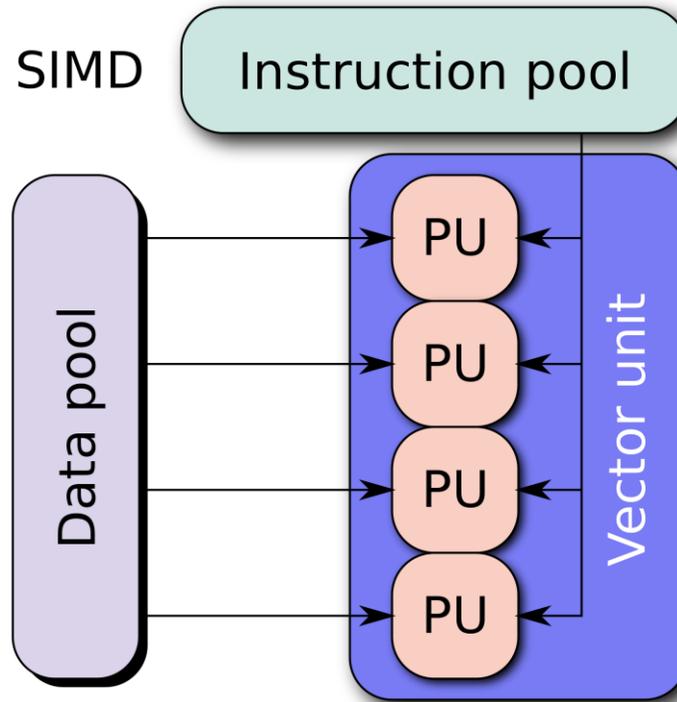
Las unidades **SIMD** se amplían y admiten más operaciones

Los compiladores mejoran la **vectorización automática**

Operaciones directas sobre **datos codificados**:

- RLE: `int sum(RLUnit u) { return u.length * u.value; }`
- Aprovecha los diccionarios cuando posible

# SIMD



Scalar Operation

$$\begin{array}{l} A_1 \times B_1 = C_1 \\ A_2 \times B_2 = C_2 \\ A_3 \times B_3 = C_3 \\ A_4 \times B_4 = C_4 \end{array}$$

SIMD Operation

$$\begin{array}{l} A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \times \begin{array}{l} B_1 \\ B_2 \\ B_3 \\ B_4 \end{array} = \begin{array}{l} C_1 \\ C_2 \\ C_3 \\ C_4 \end{array}$$

# \$vectorop()

Nuevo conjunto de funciones internas para operar con datos \$vector

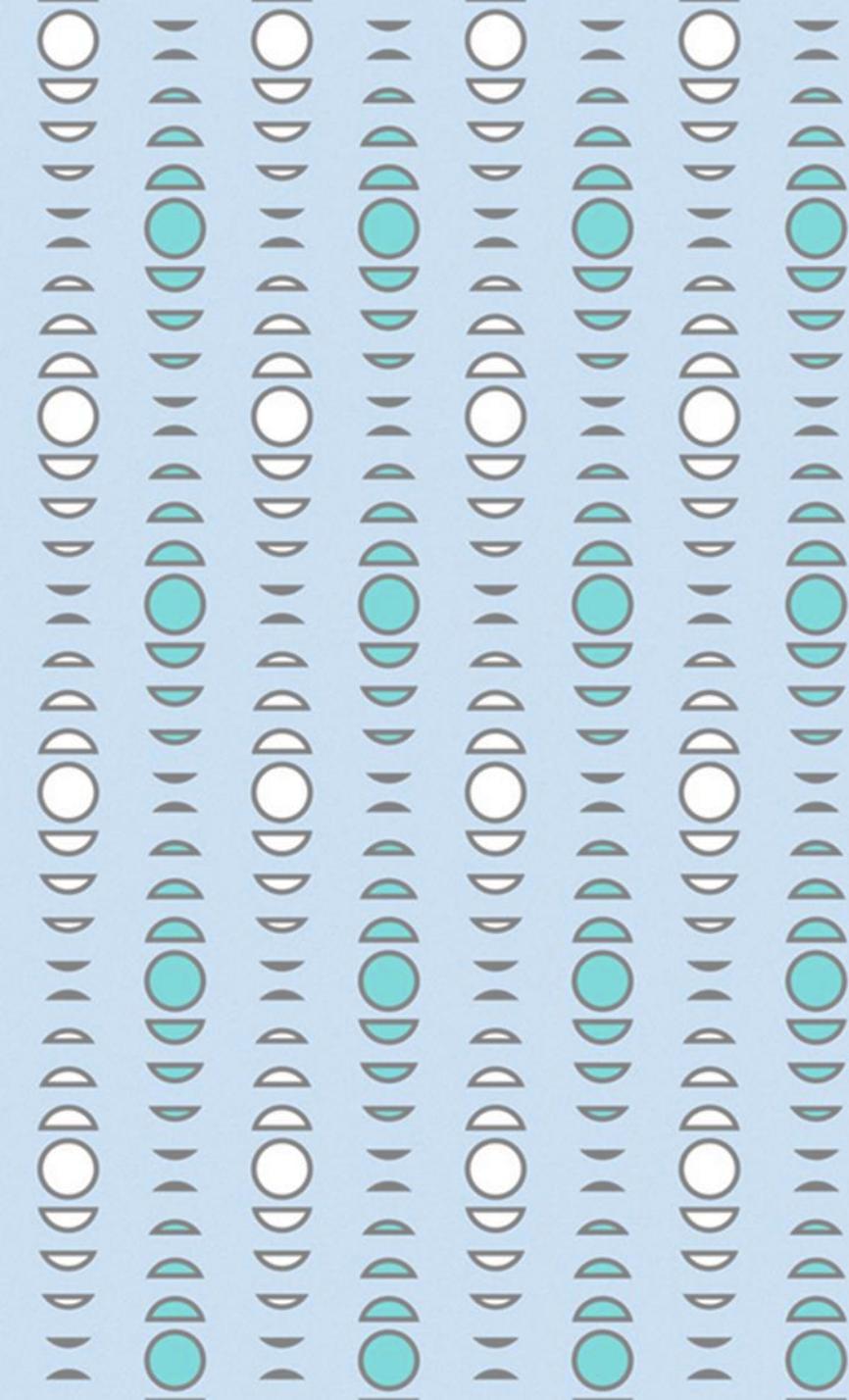
- Bucles, auto-vectorizados, SIMD, RLE, ...:

Funciones para agregados, filtros, agrupaciones, ...

ID	Producto	Ciudad (^d.V2)	Cantidad	Total (^d.V4)
1	Mac	Valencia	1	1230.50

Select Sum(Total) where Ciudad='Valencia'

```
set i = "", sum = 0
for {
  s i = $order(^d.V2(i), 1, col1)  q:i=""
  s filter = $vectorop("=", col1, "Valencia")
  s sum = sum + $vectorop("sum", ^d.V4(i), filter)
}
```



# \$vector desde SQL

---

Los planes de ejecución SQL se apoyan en estas funciones vectorizadas

- Uso de \$vectorops sobre bloques de 64K valores
- Lee solo las columnas necesarias
- Materialización tardía de la fila, cuando necesario

Procesamiento en bloques es la base para un modelo de **Ejecución paralela adaptativa**

SQL obtiene beneficios de las optimizaciones realizadas en SQL, en el kernel, el compilador y las CPUs.

# Beneficios adicionales!

---

**Consultas 10 X más rápidas con almacenamiento en columnas**

# Ahorros en Almacenamiento

---

Gestión transparente de Regiones de datos dispersos

- Elementos indefinidos no ocupan espacio
- La codificación Run-Length ofrece compresión de los valores repetidos

Codificación dependiente de tipos de datos ahorra más espacio

- Diccionario para cadenas de tipo String
- Escala adaptativa para Integers

# Ahorros en Almacenamiento

Depende de la aplicación:

	NYTaxi	Ciente 1	Ciente 2
Numero Filas	40M	13M	2.5B
Por Columnas	3.1GB	402MB	265GB
Por Filas, sin índices	5.8GB	1GB	
Por Filas + índices	6.6GB	1.3GB	2.1TB*

\* = Tabla tipo OLAP, completamente indexada

Caso de Columna Sí/No, < 5% nulls:

- Formato fila \$list : 120MB = 3 bytes / fila
- \$vector y su global completo: 5.3MB = **0.15 bytes / fila**

# Un alcance global

---

El Optimizador SQL fue diseñado para ayudar a los clientes existentes a proyectar sus estructuras de datos existente

- Admite estructuras y codificaciones de datos pre-existentes

Flexibilidad de estructura Fila/Columna en **un solo motor:**

- Por columna o por fila
- Index por columna sobre estructura por fila
- Mezcla de fila/columna en una misma tabla

# Sintaxis

---

**Consultas 10 X más rápidas con almacenamiento en columnas**



# Sintaxis

---

- Toda la tabla (Master Map) Almacenada por Columnas

```
CREATE TABLE NYTaxi.Rides (  
    VendorID Integer,  
    total_amount Numeric(8,2)  
    ...  
)  
STORAGETYPE=COLUMNAR
```

- Copia Completa de tabla:

```
CREATE TABLE NYTaxi.Rides as select *  
from NYTaxi.RowRides  
STORAGETYPE=COLUMNAR
```



# Sintaxis

---

- Una sola columna en Almacenada por Columnas

```
CREATE TABLE NYTaxi.RowRides (  
    VendorID Integer,  
    total_amount Numeric(8,2)  
    STORAGETYPE=COLUMNAR,  
    ...  
)
```

- Un Index por Columnas

```
CREATE COLUMNAR INDEX ixTotalSales  
on NYTaxi.RowRides (total_amount)
```

# ¡Acción!

---

**Consultas 10 X más rápidas con almacenamiento en columnas**

# Versiones y Futuro

---

**Consultas 10 X más rápidas con almacenamiento en columnas**

# Versiones

---

## Ya Disponible!

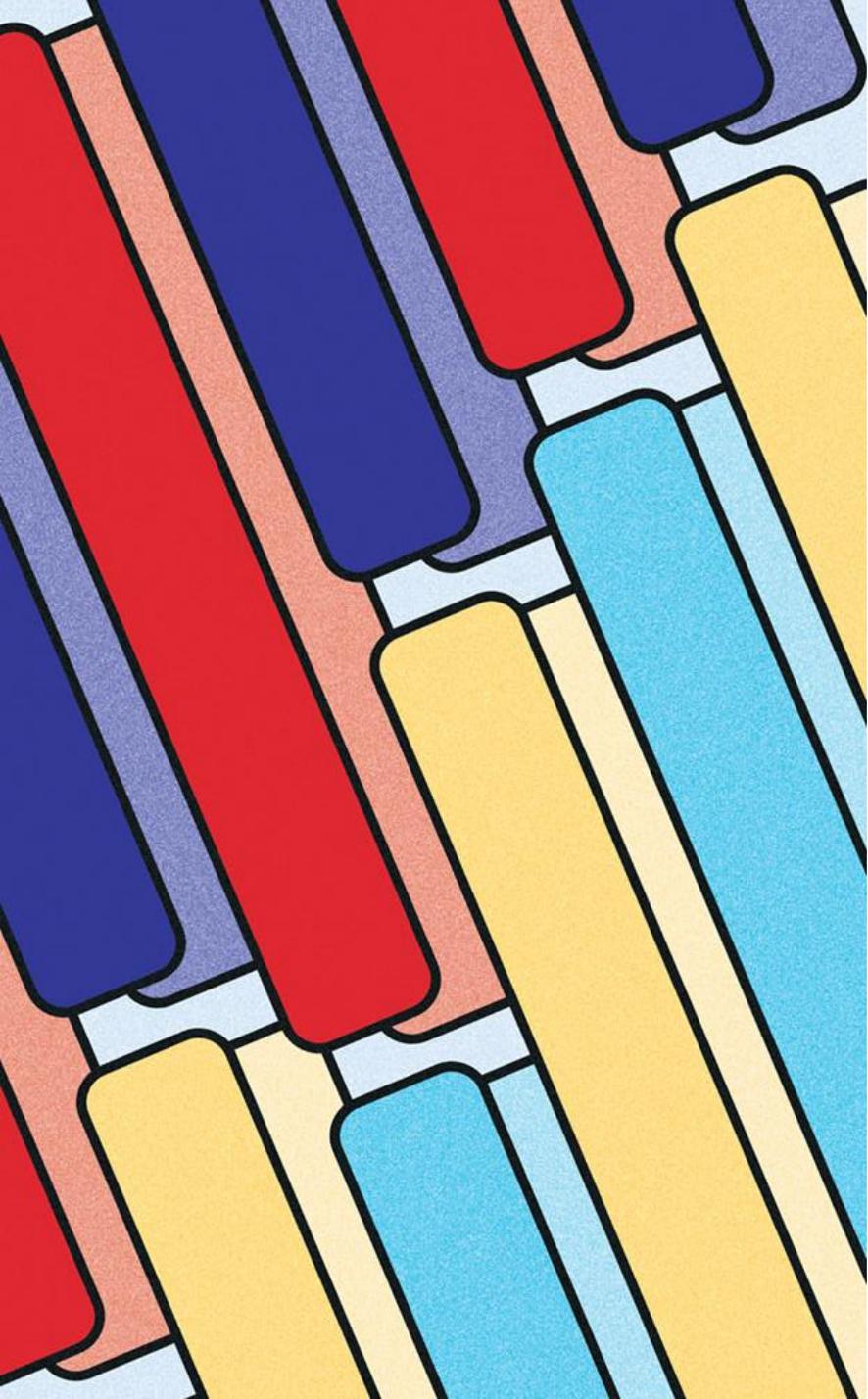
InterSystems IRIS 2022.2 CI/CD Release

- Programa de **Early Access**:  
<https://gs2022.isccloud.io/#early-access>
- Community Edition 2022.2

## Futuro

2023.1: Soporte Producción





# Hoja de ruta

---

## **Cientes xDBC**

- Carga de datos acelerada

## **Optimizador SQL**

- Espectro completo de mapas de Index / Datos Columnares

## **Ejecución paralela Adaptativa**

- Extender a planes de ejecución no orientados a columnas

# Gracias!