

# MySQL

# VIEW

- Es una tabla virtual basada en el resultado de una sentencia SELECT.
- Contiene filas y columnas, como una tabla real. Los campos que muestra pertenecen a una o más tablas reales de la base de datos.
- Al ejecutar un SELECT de una vista el motor internamente ejecuta la query contenida en la misma (en general no se utilizan con INSERTS, UPDATE, DELETE).

# VIEW

1. Creo la vista peliculas\_no\_preferidas

```
CREATE VIEW peliculas_no_preferidas
AS
SELECT p.id, p.titulo
FROM pelicula AS p
WHERE NOT EXISTS (
    SELECT a.id
    FROM actor AS a
    WHERE a.id_pelicula_preferida = p.id
);
```

2. La utilizo mediante un SELECT

```
SELECT * FROM peliculas_no_preferidas;
```

3. Modifico su query interna

```
CREATE OR REPLACE VIEW peliculas_no_preferidas
AS
SELECT p.id, p.titulo
FROM pelicula AS p
    LEFT JOIN actor AS a ON a.id_pelicula_preferida = p.id
WHERE a.id IS NULL;
```

4. Elimino la vista

```
DROP VIEW peliculas_no_preferidas;
```

# VIEW

- Simplifica el uso de queries complejas que se utilicen desde distintos lugares.
- Permite implementar un esquema de seguridad tanto por filas como por columnas.
- Disponer la información de una manera determinada para cada uso

# Rutinas almacenadas (Procedimientos y Funciones)

- Conjunto de sentencias SQL que se pueden almacenar en el server (BD).
- Una vez almacenadas, se las puede utilizar siempre que se requieran (o corresponda).
- Permiten independizar al cliente (e.g.: PHP, usuario desde una GUI) de la lógica propia de la manipulación, almacenamiento y gestión de los datos.
  - Reutilización desde distintos lugares.
  - Esquema de seguridad sobre la capa de datos.

# Funciones

- Se pueden utilizar como parte de una expresión (dentro del SELECT, WHERE, ORDER BY, etc.) y que devuelvan un resultado.
- Funciones “built-in” → Vienen incorporadas en MySQL:
  - De tipo String: CONCAT(), LENGTH(), MID(), LEFT() ...
  - De tipo Date: DAY(), MONTH(), YEAR(), ADDDATE(), NOW() ...
  - De tipo Numeric: ABS(), MOD(), ROUND(), RAND() ...
  - De control de flujo: CASE, IF(), IFNULL() ...
  - ... y muchas más!
- El usuario de la BD puede crear y almacenar sus propias funciones

# Funciones built-in

```
4 SELECT nombre, apellido, rating,  
5     CONCAT('Siglas: ', LEFT(nombre,1), LEFT(apellido,1)) AS Siglas_Actor  
6 FROM actor  
7 WHERE ROUND(rating)=7;  
8
```

Result Grid



Filter Rows:

Export:



Wrap Cell Content:



	nombre	apellido	rating	Siglas_Actor
✓	Mark	Hamill	6.5	Siglas: MH
	Jeremy	Irons	7.2	Siglas: JI
	Jim	Parsons	6.9	Siglas: JP

# FUNCTION – Función creada por el usuario

```
DROP FUNCTION IF EXISTS Siglas_Actor;

CREATE FUNCTION Siglas_Actor (nombre VARCHAR(100), apellido VARCHAR(100))
RETURNS VARCHAR(50) DETERMINISTIC
-- Acá podría haber más código de la función
-- ...
RETURN CONCAT('Siglas: ', LEFT(nombre,1), LEFT(apellido,1));
```



# FUNCTION – Función creada por el usuario

```
DROP FUNCTION IF EXISTS Siglas_Actor;

DELIMITER $$






CREATE FUNCTION Siglas_Actor (nombre VARCHAR(100), apellido VARCHAR(100))
RETURNS VARCHAR(50) DETERMINISTIC
BEGIN
    DECLARE Siglas VARCHAR(50);

    SET Siglas = CONCAT('Siglas: ', LEFT(nombre,1), LEFT(apellido,1));

    RETURN Siglas;
END$$

DELIMITER ;
```

# FUNCTION – Función creada por el usuario

```
23 
24  SELECT nombre, apellido, rating,
25      Siglas_Actor(nombre, apellido) AS Siglas_Actor
26 FROM actor
27  WHERE ROUND(rating)=7;
28 
```

Result Grid



Filter Rows:

Export:



Wrap Cell Content:



nombre	apellido	rating	Siglas_Actor
Mark	Hamill	6.5	Siglas: MH
Jeremy	Irons	7.2	Siglas: JI
Jim	Parsons	6.9	Siglas: JP

# FUNCTION – Función creada por el usuario

## Ejercicio:

- Crear tabla 'Personas' con 3 columnas Nombre y Apellido, Edad y Sexo (char 'M' o 'F'). Insertarle al menos 5 registros con datos variados.
- Crear una FUNCTION que reciba los 3 valores de la Persona y utilizarla en un SELECT para “saludarla según corresponda”. Por ejemplo:

NomyApe	Edad	Sexo
Gonzalo Gonzalez	40	M
María Martinez	38	F
Fede Ferrari	15	M



FuncSaludar(NomyApe, Edad, Sexo)
Hola Sr. Gonzalo Gonzalez!
Hola Sra. María Martinez!
Hola Fede Ferrari!

# Procedimientos (Stored Procedures)

Un PROCEDURE se diferencia de una FUNCTION en:

- Puede retornar ninguno, 1 ó más valores por parámetros de salida (OUT).
- Puede ejecutar sentencias SELECT e incluso modificar datos en la BD.
- Permite recorrer los registros de una tabla y procesar sus datos.
- Se ejecuta en una sentencia específica. No puede ser llamado desde una expresión.

# Tablas para usar en ejemplos

```
DROP TABLE IF EXISTS cliente;  
CREATE TABLE cliente (  
    id            INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,  
    nombre_completo VARCHAR(200) NOT NULL,  
    balance       DECIMAL(10, 2) NOT NULL DEFAULT 0  
);
```

```
DROP TABLE IF EXISTS movimiento;  
CREATE TABLE movimiento (  
    id            INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,  
    id_cliente    INT UNSIGNED NOT NULL,  
    valor         DECIMAL(8,2) NOT NULL  
);
```

# PROCEDURE - Ejercicio

Crear un PROCEDURE 'Insertar\_movimiento' que reciba por parámetros ID\_Cliente y Valor (>0 ó <0) y luego realice las siguientes operaciones:

- Registrar el nuevo movimiento realizado por el Cliente
- Actualizar su Balance (o saldo)

```
DROP PROCEDURE IF EXISTS insertar_movimiento;  
DELIMITER //  
  
CREATE PROCEDURE insertar_movimiento (  
    IN v_id_cliente INT UNSIGNED,  
    IN v_valor      DECIMAL(8, 2)  
)  
BEGIN
```

# Solución propuesta

1. Insertar el movimiento realizado por el cliente
2. Sumar todos los movimientos para calcular el balance
3. Actualizar el balance con la Suma calculada
4. Ejecutar el SP mediante CALL

```
DROP PROCEDURE IF EXISTS insertar_movimiento;
DELIMITER //

CREATE PROCEDURE insertar_movimiento (
    IN v_id_cliente INT UNSIGNED,
    IN v_valor      DECIMAL(8, 2)
)
BEGIN
    INSERT INTO movimiento (id_cliente, valor)
    VALUES (v_id_cliente, v_valor);

    SELECT SUM(valor) INTO @balance
    FROM movimiento
    WHERE id_cliente = v_id_cliente;

    UPDATE cliente SET balance = @balance
    WHERE id = v_id_cliente;
END;

//
DELIMITER ;

CALL insertar_movimiento(1, 100);
```

# PROCEDURE

- Crear una FOREIGN KEY sobre id\_cliente del Movimiento para referenciar al id Cliente y asegurar la Integridad del modelo de datos
- Volver a ejecutar el SP, para un cliente que no existe.
- ¿Qué sucede ahora?
- ¿Cómo solucionarlo?



# Solución propuesta 2

1. Se verifica la existencia del Cliente
2. Si no existe, lo crea con un nombre Default ("SIN NOMBRE")
3. Luego sigue con las operaciones normales (registrar el movimiento y actualizar el balance)

## Otras alternativas:

- Recibir el Nombre por parámetro y usarlo cuando no exista el Cliente
- Realizar la verificación y devolver un mensaje de error si no la cumple
- Hacer que falle, capturar el error y devolver un mensaje

```
ALTER TABLE Movimiento
ADD CONSTRAINT FOREIGN KEY (id_cliente) REFERENCES Cliente(id);

DROP PROCEDURE IF EXISTS Insertar_Movimiento;
DELIMITER //
CREATE PROCEDURE Insertar_Movimiento (
    IN v_id_cliente INT UNSIGNED,
    IN v_valor      DECIMAL (8, 2)
)
BEGIN
    SELECT COUNT(*) INTO @CantClientes
    FROM Cliente
    WHERE id = v_id_cliente;

    IF @CantClientes = 0 THEN
        INSERT INTO Cliente (id, nombre_completo)
        VALUES (v_id_cliente, 'SIN NOMBRE');
    END IF;

    INSERT INTO Movimiento (id_cliente, valor)
    VALUES (v_id_cliente, v_valor);

    SELECT SUM(valor) INTO @balance
    FROM Movimiento
    WHERE id_cliente = v_id_cliente;

    UPDATE Cliente SET balance = @balance
    WHERE id = v_id_cliente;
END;
//
DELIMITER ;
```

## Stored PROCEDURE y FUNCTION

Conviene utilizar una FUNCTION cuando se requiere:

- Procesar los datos recibidos, realizar cálculos con los mismos y devolver un resultado
- Reutilizarlo en varias expresiones que precisen hacer lo mismo
- Modularizar (la función la crea otra persona)

Conviene utilizar un PROCEDURE cuando se requiere:

- Ejecutar sentencias SQL para procesar datos sobre tablas en la BD
- Mantener en un único lugar (por seguridad, por reutilización) diferentes operaciones que trabajen sobre los datos

# TRIGGER

Un Trigger es similar a un Stored Procedure con las siguientes particularidades:

- Está asociado a una tabla específica
- Es 'disparado' por eventos que ocurren sobre la tabla

Evento                    [ DELETE | UPDATE | INSERT ]

Momento                [ BEFORE | AFTER ]

# TRIGGER

```
DROP TRIGGER IF EXISTS Actualiza_Balance;
DELIMITER //
CREATE TRIGGER Actualiza_Balance
AFTER INSERT ON Movimiento
FOR EACH ROW BEGIN
    SELECT SUM(valor) INTO @balance
    FROM Movimiento
    WHERE id_cliente = NEW.id_cliente;

    UPDATE Cliente SET balance = @balance
    WHERE id = NEW.id_cliente;
END;
//
DELIMITER ;
```

# TRIGGER

```
INSERT INTO Cliente (nombre_completo) VALUES ('José Jaime'); -- Lo creó con ID=2
INSERT INTO Movimiento (id_cliente, valor) VALUES (2, 30);
INSERT INTO Movimiento (id_cliente, valor) VALUES (2, 50);
INSERT INTO Movimiento (id_cliente, valor) VALUES (2, -20);
```

```
SELECT * FROM Movimiento WHERE id_cliente = 2;
```

id	id_cliente	valor
7	2	30.00
8	2	50.00
9	2	-20.00

```
SELECT * FROM Cliente WHERE id = 2;
```

id	nombre_completo	balance
2	José Jaime	60.00

# TRIGGER

## Ejercicio:

- Sobre la base de películas, crear 2 triggers: uno asociado a la tabla 'actor\_pelicula' y otro a 'actor\_episodio'
- Por cada nueva película en la que actúe un actor, sumarle a su rating un 10%
- Por cada nuevo episodio de series, sumarle a su rating un 2%
- En caso que el rating supere los 10 puntos, dejarlo en 10 y ya no modificarlo

Variante extra: Si el rating de la película o episodio es menor al del actor, restarle el porcentaje en vez de sumárselo.

# TRIGGER – Solución propuesta

1. Obtener el Rating actual del actor
2. A ese valor, sumarle un 10%
3. Control para asegurarse que luego de sumarle un 10% el valor no supera los 10 puntos.
4. Actualizar el nuevo rating para el actor

```
USE peliculas_clase_4;

DROP TRIGGER IF EXISTS Actualiza_Rating_Por_Pelicula;

DELIMITER //
CREATE TRIGGER Actualiza_Rating_Por_Pelicula
AFTER INSERT ON actor_pelicula
FOR EACH ROW BEGIN
    SELECT rating INTO @rating
    FROM actor
    WHERE id = NEW.id_actor;

    SET @rating = @rating + 1.1;

    IF @rating > 10 THEN
        SET @rating = 10;
    END IF;

    UPDATE actor SET rating = @rating
    WHERE id = NEW.id_actor;
END;
//
DELIMITER ;
```

# TRUNCATE TABLE

Borrar todos los registros de una tabla:

```
DELETE FROM pelicula;
```

```
TRUNCATE TABLE pelicula;
```



# TRUNCATE TABLE

- Elimina la tabla y la vuelve a crear (requiere permiso de DROP y CREATE TABLE), sólo si no hay referencias a registros de otras tablas (ídem DELETE)
- Los AUTO\_INCREMENT vuelven a su valor inicial
- Triggers asociados a la tabla no se ejecutan (porque no hay evento DELETE)
- Es más rápido que el DELETE (sobretudo en tablas con muchos registros)
- No se permite el ROLLBACK

# Seguridad

- El Admin de la BD permite a usuarios **ver** (columnas, registros, metadata) y **hacer** (ejecutar SPs, acceder a objetos) sólo lo que le corresponda
- Usuario que accede a Vista con datos de Empleado menos los datos sensibles (ej: sueldo)
- SP que permite realizar modificaciones sobre tablas de forma acotada (ej: sólo modificar algunos campos en lugar de permitirle UPDATE sobre la tabla)

# Seguridad: GRANT / REVOKE

```
CREATE USER Pichi@localhost IDENTIFIED BY 'mipassword1';

GRANT USAGE ON peliculas_clase_4.* TO Pichi@localhost;
GRANT SELECT ON peliculas_clase_4.Pelicula TO Pichi@localhost;
GRANT SELECT ON peliculas_clase_4.Actor TO Pichi@localhost;
REVOKE SELECT ON peliculas_clase_4.Pelicula FROM Pichi@localhost;

SHOW GRANTS FOR Pichi@localhost;

SELECT * FROM mysql.user;
```

# Seguridad

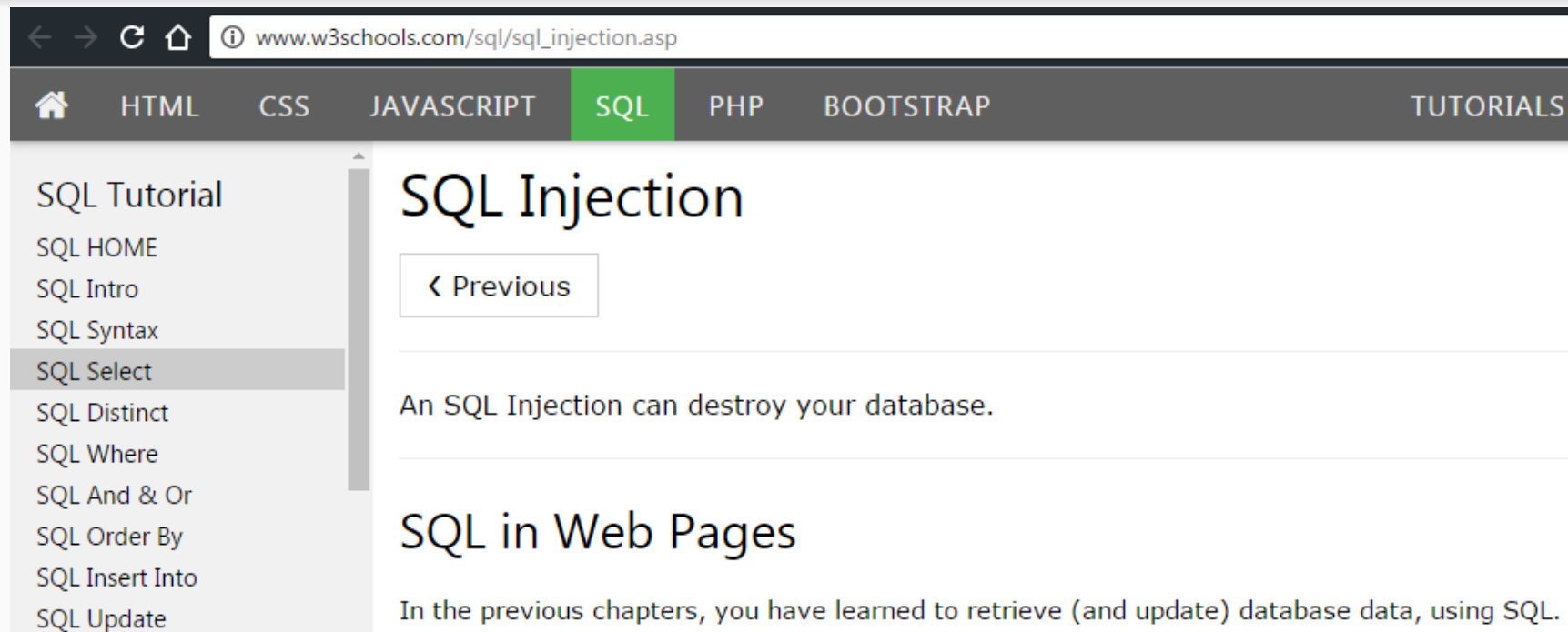
The screenshot shows a database management interface. On the left, the 'Navigator' pane displays a schema named 'películas\_clase\_4' containing a table 'actor'. The main editor, titled 'Query 1', contains the following SQL code:

```
1 use películas_clase_4;
2 select * from actor;
3 select * from pelicula;
4
```

The 'Output' pane at the bottom shows the execution results in a table with columns: #, Time, Action, and Message.

#	Time	Action	Message
✓ 1	15:13:21	use películas_clase_4	0 row(s) affected
✓ 2	15:13:21	select * from actor LIMIT 0, 1000	49 row(s) returned
✗ 3	15:13:30	select * from pelicula LIMIT 0, 1000	Error Code: 1142. SELECT command denied to user 'Pichi'@'localhost' for table 'pelicula'

# SQL Injection



The screenshot shows a web browser window with the URL `www.w3schools.com/sql/sql_injection.asp`. The navigation bar includes links for HTML, CSS, JAVASCRIPT, SQL (highlighted in green), PHP, BOOTSTRAP, and TUTORIALS. A left sidebar lists SQL topics, with 'SQL Select' currently selected. The main content area features the title 'SQL Injection', a 'Previous' button, a paragraph stating 'An SQL Injection can destroy your database.', the title 'SQL in Web Pages', and a paragraph stating 'In the previous chapters, you have learned to retrieve (and update) database data, using SQL.'

← → ↻ 🏠 `www.w3schools.com/sql/sql_injection.asp`

🏠 HTML CSS JAVASCRIPT **SQL** PHP BOOTSTRAP TUTORIALS

SQL Tutorial

- SQL HOME
- SQL Intro
- SQL Syntax
- SQL Select**
- SQL Distinct
- SQL Where
- SQL And & Or
- SQL Order By
- SQL Insert Into
- SQL Update

## SQL Injection

[← Previous](#)

---

An SQL Injection can destroy your database.

---

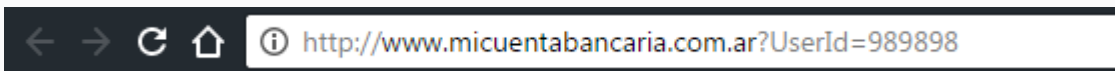
## SQL in Web Pages

In the previous chapters, you have learned to retrieve (and update) database data, using SQL.

# SQL Injection

- Técnica que permite a usuarios malintencionados “inyectar” o intercalar comandos SQL dentro de una sentencia SQL, vía parámetros de páginas web.
- Al alterar el comportamiento de las sentencias SQL puede comprometer la seguridad de la aplicación e incluso de la BD.
- Es una de las principales **vulnerabilidades** de los sitios web

# SQL Injection



El código de la página web recibe el Id del Usuario logueado y consulta sus

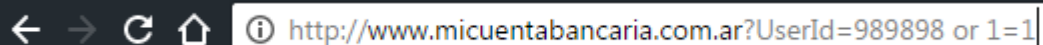
datos:

```
txtUserId = getQueryString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

En la BD se ejecutará la sentencia SQL:

```
SELECT * FROM Users WHERE UserId = 989898;
```

# SQL Injection

A screenshot of a web browser's address bar. It features navigation icons (back, forward, refresh, home) on the left and a search icon on the right. The address bar contains the URL: http://www.micuentabancaria.com.ar?UserId=989898 or 1=1. The text is in a light blue font on a dark background.

← → ↻ 🏠 ⓘ http://www.micuentabancaria.com.ar?UserId=989898 or 1=1

En la BD se ejecutará la sentencia SQL:

```
SELECT * FROM Users WHERE UserId = 989898 or 1=1;
```



# SQL Injection

Cómo protegerse del SQL Injection:

- Utilizar “binding” de parámetros SQL para asignar valores a sentencias SQL (en vez de concatenar la sentencia con variables del lenguaje web)
- Realizar las operaciones sobre la BD a través de Stored Procedures

# SQL Injection – Ejemplo de binding

```
<?php

$userid = $_GET['UserId'];
$stmt = $mysqli->prepare('SELECT UserId, name, password
    FROM Users
    WHERE UserId = ?');
$stmt->bindParam(1, $userid, PDO::PARAM_INT);
$stmt->execute();
$stmt->close();
?>
```

# Business Intelligence

## Definiciones

- La habilidad de comprender las **interrelaciones de los hechos** presentados de forma de orientar las acciones en dirección a un **objetivo**. -Wikipedia
- Business Intelligence significa utilizar los **datos** que posea la compañía para tomar mejores **decisiones de negocio**. Es sobre acceder, analizar y descubrir nuevas oportunidades. -IBM
- Conceptos y métodos para mejorar las **decisiones de negocio** utilizando como soporte sistemas **basados en hechos**. -Gartner

# Business Intelligence - Conceptos básicos

- **Indicadores / Métricas:** Venta Neta, Ingresos, Nuevos Clientes
- **Dimensiones:** País, Vendedor, Fecha/Hora
- **KPI (Key Performance Indicators):** “100 Clientes nuevos en 2015”, “Reducción de costos del 5% en el 4to trimestre del año”
- **Tablero de Control:** Conjunto de indicadores y gráficos que resumen y facilitan el seguimiento y gestión periódica de un área o de toda la empresa
- **Data warehouse:** Repositorio de datos relacionados, provenientes de distintos sistemas, integrado, no volátil y variable en el tiempo, orientado a consulta y análisis.
- **ETL:** Proceso de extracción de datos desde el origen, transformación (correcciones, cálculos, etc.) y carga en el repositorio destino (e.g., DWH)

**Data Sources**

**Data Flow**

**BI results**

Operational System



ERP



CRM



SQL



Flat Files,  
Spreadsheets

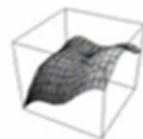


**ETL**

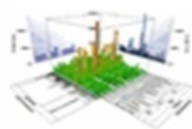
Extract, Transform,  
Load



**OLAP Analysis**



**Data Mining**



**Data Visualization**



**Reports**



**Dashboards**



**Alerts**

# Bases de datos NoSQL (“Not only SQL”)

- Difieren de las BD relacionales principalmente en su modelo de datos
- Open-source, Distribuidas y de Escalabilidad Horizontal
- Priorizan la rapidez (para escrituras y lecturas simples) y flexibilidad antes que la consistencia (no son ACID) → **C**onsistency, **A**vailability & **P**artition tolerance trade-off
- Soportan grandes volúmenes de datos (“Big Data”) que pueden ser semi o no estructurados (Text, Log Files, Click Streams, Blogs, Tweets, etc...)

# Bases de datos NoSQL (“Not only SQL”)

En vez de utilizar tablas relacionadas, con filas y columnas de similares características, permiten diseñar su Modelo de Datos mediante otros objetos. Los más comunes son:

- Key-Value
- Column
- Document

*¡Hay decenas de modelos de datos y más de 200 sistemas de BD NoSQL!*

# NoSQL – Key-Value

- Los datos consisten en una clave indexada (y única) y un valor asociado
- Lecturas/Escrituras muy rápidas si se cuenta con la Key
- Se utilizan como sistemas de almacenamiento caché

- Ejemplos:





# NoSQL – Column

- Una “columna” sería como la fila de una tabla relacional, en la cual hay yba cantidad de datos (valores) relacionados entre sí
- Se referencia a la columna mediante su nombre (único)
- Contiene un Timestamp para identificar cuán antiguo es y así poder obtener el más reciente

- Ejemplos:



# NoSQL – Document

- Almacena los datos en estructuras de tipo JSON o XML
- Un Document contiene diferentes campos tipificados con valores relacionados entre sí (en vez de referenciar a registros de otras tablas)
- Fácil de evolucionar agregando nuevos campos a un mismo Document
- Se puede consultar un Document mediante su key o cualquier otro campo
- Ejemplos:

