

# BASE DE DATOS APLICADAS

UNLaM

Materia de la carrera Ingeniería en Informática

Resúmenes, apuntes y modelos de exámenes del segundo nivel de base de datos

Gonzalez, Nazarena Araceli  
araceli009uo@gmail.com



BASE DE DATOS APLICADAS.....	7
<b>UNIDAD N°1: CONCEPTOS BASICOS .....</b>	<b>7</b>
DBMS (Database Management System) .....	7
¿Siempre conviene usar un DBMS? .....	7
Administrador de base de datos (DBA).....	7
Programador/desarrollador de base de datos.....	7
Modelo de datos: abstracción .....	8
Abstracción de datos .....	8
Modelo de datos.....	8
Estructura de la base .....	8
Constraints.....	8
Definición y tipos de datos .....	8
Esquemas (schema) .....	9
Catálogos .....	9
Catálogo.Esquema.Objeto .....	9
DDL: Data Definition Language .....	9
ALTER.....	9
TRUNCATE .....	10
DML: Data Manipulation Language .....	10
INSERT .....	10
UPDATE .....	10
DELETE.....	10
MERGE.....	10
Tabla, fila y columna .....	10
Tipos de datos.....	10
Integridad referencial, claves primarias y foráneas, restricciones. ....	11
Integridad referencial.....	11
Clave primaria .....	11
Clave foránea.....	11
Restricciones .....	12
UNIQUE .....	12
CHECK.....	12
NOT NULL.....	12
DEFAULT .....	12
NULL .....	12
Consultas básicas .....	13
SELECT .....	13
FROM.....	13
WHERE.....	13
CREATE .....	13
Juntas y uniones .....	13
UNION .....	13
JOIN .....	13
SQL es un lenguaje declarativo .....	14
Subconsultas .....	14
Vistas.....	15
Funciones de agregado .....	15
Procedimientos almacenados.....	16
Triggers .....	16
Funciones .....	17



Window Functions .....	18
Common Table Expressions .....	19
SQL Dinámico .....	20
PIVOT .....	20
Notas de clases .....	21
• ¿Inicializar variables con Set o con Select? .....	21
• Set nocount on .....	22
• Top / Limit (depende del motor si es uno o el otro) .....	22
• Truncate .....	22
• Tiempo de vida de las variables .....	22
• Generar valores aleatorios .....	22
• Percentil .....	22
• Fibonacci con recursividad .....	23
• Función .....	23
• CTE .....	23
• ROW_NUMBER: eliminar duplicados .....	23
• CONSTRAINT .....	24
 <b>UNIDAD N°2: BASES DE DATOS RELACIONALES .....</b>	 25
Bases de datos transaccionales .....	25
Características .....	25
Ventajas .....	25
Desventajas .....	25
¿Cómo funcionan y para qué sirven? .....	25
Uso y aplicaciones .....	25
Base de datos relacional vs transaccional .....	26
Ejemplos .....	26
Transferencias bancarias .....	26
Sistemas de venta online .....	26
Arquitectura de Bases de datos Distribuidas .....	26
Clúster .....	26
Clúster en la nube .....	26
Clúster de servidores dedicados .....	26
Base de datos - Data Warehouse – Data Lake .....	27
Base de Datos .....	27
Data Warehouse (DWH) .....	27
Data Lake (DL) .....	27
¿Data Lake o Data Warehouse? .....	28
Motor de base de datos (RDBMS o SGBD) .....	28
Características opcionales en MS SQL Server .....	28
Análisis previo a la instalación .....	29
Aspectos clave de la instalación .....	29
Planifique cuidadosamente esta configuración considerando: .....	30
Conexión a un motor de base de datos .....	31
Paso1: Conexión al Motor de base de datos .....	31
Paso 2: Conexión desde otro equipo .....	31
Collate – Intercalación .....	31
Consideraciones particulares .....	32
Bases de datos en memoria .....	32
¿Por qué se necesitan BBDDs en memoria? .....	32
Implementación de SQL Server .....	33



ODBC y JDBC .....	34
ODBC.....	34
JDBC.....	34
Formatos de intercambio de datos.....	34
EDI (Electronic Data Interchange).....	35
CSV: Comma separated values .....	36
Ancho fijo, etc.....	36
XML: extended Markup Language .....	36
JSON: JavaScript Object Notation .....	37
YAML: YAML Ain't a Markup Language .....	38
Práctica sugerida.....	39
API: Application Programming Interface.....	39
Ejemplo .....	40
APIs .....	40
APIs abiertas o públicas:.....	40
APIs internas o privadas: .....	40
APIs de socios: .....	40
APIs: Postman .....	40
Notas de clases.....	40
Perlitas de Jair:.....	40
Control + R .....	40
Tecla ALT.....	40
Shutdown.....	41
Sp_help .....	41
@@ .....	41
@@rowcant.....	41
#Tablas temporales.....	41
##Tablas temporales globales .....	41
<b>UNIDAD N°3: ASPECTOS AVANZADOS.....</b>	<b>42</b>
El plan de ejecución .....	42
¿Qué ocurre cuando se ejecuta una consulta? .....	42
¿Qué ocurre cuando se ejecuta una consulta? .....	42
¿Qué ocurre cuando se ejecuta una consulta? .....	43
Plan estimado y plan real (actual).....	43
¿Qué mirar en un plan de ejecución? .....	43
Índices .....	43
Índices clúster .....	44
Índices no clúster .....	44
Índices de cobertura .....	45
Índices: Fill factor .....	46
Optimización de índices .....	47
Rebuild .....	47
Reorganize .....	47
Estadísticas.....	48
• Encabezado.....	48
• Grafo de densidad.....	48
• Histograma.....	48
Optimización de consultas .....	50
Mi consulta tarda mucho... .....	51
Tablas temporales.....	51



Rowstore vs Columnstore .....	51
Transaction Control Language (TCL) .....	53
Atomicidad.....	53
Consistencia.....	54
Aislamiento .....	54
Durabilidad .....	54
SQL Server .....	55
Transacciones implícitas .....	55
Transacciones explícitas .....	55
Control de concurrencia.....	55
Bloqueos .....	56
SET TRANSACTION ISOLATION LEVEL .....	57
READ UNCOMMITTED .....	57
READ COMMITTED.....	57
REPEATABLE READ .....	58
SERIALIZABLE .....	58
SNAPSHOT .....	58
Observaciones .....	58
Transacciones distribuidas .....	59
Monitoreo .....	61
Métricas de performance .....	61
Recuento de filas (row counts) .....	61
Entrada/salida de archivos (database file IO) .....	62
El modelo de recuperación es una propiedad de la DB que controla cómo se mantiene el log de transacciones.....	62
Modelo de recuperación SIMPLE.....	62
Modelo de recuperación FULL.....	63
Tamaño del backup del log de transacciones (transaction log backup size) .....	63
Bloqueos/esperas .....	63
Obtención y análisis de las métricas .....	63
Notas de clases.....	64
 <b>UNIDAD N°4: BASES DE DATOS NO RELACIONALES</b> .....	 65
Surgimiento de base de datos NoSQL.....	65
Base de datos NoSQL .....	66
Escalabilidad: .....	66
Flexibilidad:.....	66
Alta disponibilidad: .....	66
Rendimiento: .....	66
Características de bases de datos NoSQL.....	66
Arquitectura.....	66
La decisión sobre cual arquitectura implementar se basa en: .....	66
Datos estructurados, semi estructurados y no estructurados .....	67
Datos estructurados .....	67
Datos semi estructurados.....	67
Datos no estructurados .....	67
Almacenamiento de datos - Procesamiento de datos.....	67
• Reconocimiento Óptico de Caracteres (OCR).....	67
• Procesamiento del Lenguaje Natural (NLP) .....	67
Datos estructurados, semi estructurados y no estructurados .....	67
Propiedades de los modelos de BBDD .....	68



ACID .....	68
BASE.....	68
Teorema CAP .....	69
C - Consistencia (Consistency) .....	69
A - Disponibilidad (Availability) .....	69
P - Tolerancia a particiones (Partition tolerance):.....	69
Modelos de base de datos NoSQL .....	69
Base de datos de pares clave -valor.....	69
Bases de datos de documentos .....	70
Bases de datos de grafos .....	71
Bases de datos Columnas y familias de columnas.....	71
Notas de clases.....	72
Crear base de datos en mongo .....	72
Ver desde la línea de comando lo que acabo de crear .....	73
Insertar un documento .....	73
Insertar varios documentos: insertMany .....	73
Tipos de datos básicos de mongo .....	74
String .....	74
Integer .....	74
Doublé .....	74
Booleano .....	74
Null .....	74
Array.....	74
Objeto.....	74
Date (fecha).....	75
Update un solo campo.....	75
Update un varios campos .....	76
Borrar un registro .....	76
Borrar varios registros .....	76
MongoDB Compass Aggregation Framework .....	76
Crear agregaciones con MongoDB Compass .....	77
\$lookup.....	78
\$unwind.....	79
\$replaceRoot:.....	80
\$addFields .....	80
\$match .....	83
\$sort .....	84
\$sortByCount.....	84
\$count .....	84
\$limit .....	84
\$skip .....	84
\$out .....	84
\$group .....	84
<b>UNIDAD N°5: SEGURIDAD DE DATOS .....</b>	<b>90</b>
Seguridad vs Protección.....	90
Seguridad:.....	90
Protección:.....	90
Principio del mínimo privilegio (POLP) .....	90
Reducción del área expuesta .....	90
Permisos, usuarios y roles.....	91



❑ Entidades de seguridad (principals) .....	91
❑ El usuario sa (sysadmin) .....	91
❑ Elementos protegibles (securables) .....	91
❑ Directivas de contraseñas .....	91
Principals & Securables.....	92
❑ Usuario de base de datos .....	93
❑ Roles fijos de servidor .....	93
❑ Roles de nivel base de datos .....	94
❑ Propietarios .....	94
❑ DCL Data Control Language:.....	95
GRANT .....	95
REVOKE.....	95
DENY.....	95
❑ Jerarquía de permisos .....	96
❑ Permisos mediante código basado en procedimiento .....	96
❑ ¿Qué es una transacción? .....	97
Modelo de recuperación SIMPLE .....	97
Modelo de recuperación FULL .....	97
Registro de Transacciones (RT) .....	98
❑ ¿Qué es el registro de transacciones?.....	98
❑ ¿Qué almacena un registro de transacciones SQL Server? .....	98
❑ Archivos de registro virtuales (VLF).....	99
❑ Mantenimiento del log de transacciones .....	100
❑ Backup .....	100
Respaldo de la base de datos .....	101
❑ Clasificación .....	101
❑ Otros tipos de backup .....	101
❑ Completo (FULL) .....	101
❑ Diferencial .....	101
❑ Registro de transacciones .....	102
❑ Estrategia de backups .....	102
❑ Tail log .....	102
❑ Copy only .....	102
❑ Otras opciones .....	103
❑ Buenas prácticas .....	103
Al restaurar podrá indicar si desea:.....	103
Alta disponibilidad (HA) .....	104
Rélicas de la DB.....	105
Transaccional:.....	106
Log Shipping (despacho o envío de registro de transacciones) .....	106
Rélicas vs Alta disponibilidad .....	107
Encriptación .....	107
<b>LINK CLASES .....</b>	<b>108</b>



## BASE DE DATOS APLICADAS

### UNIDAD N°1: CONCEPTOS BASICOS

#### DBMS (Database Management System)

Es un Software que permite crear y mantener una base de datos.

- Permitir las tareas de construcción, manipulación y compartir bases de datos a distintos usuarios y aplicaciones
- Proteger contra fallos de HW y SW, accesos no autorizados o malintencionados.
- Permitir al sistema evolucionar a lo largo del ciclo de vida del SW.
- Eficiente, confiable, conveniente, seguro, multiusuario y capaz de almacenar cantidades masivas de datos en forma persistente.

#### ¿Siempre conviene usar un DBMS?

Existen escenarios donde es contraproducente usar DBMS. Ej:

- Inversión inicial elevada para HW, SW y capacitación
- Overhead resultante de los mecanismos de seguridad, concurrencia, control, recuperación e integridad.
- Datos que no cambiarán a lo largo del ciclo de vida del SW.
- Capacidad limitada como en sistemas embebidos.
- No existe necesidad de multiusuario.

#### Administrador de base de datos (DBA)

Función: garantizar la continuidad del negocio. Respaldar el crecimiento.

Recursos a su cargo: las DB y el DBMS y software relacionado.

- Autoriza y regula el acceso.
- Monitorea y optimiza el uso.
- Adquiere SW y HW necesarios.
- Asegura disponibilidad.

Generalmente se enfoca en pocas tecnologías de DB

El DBA va a ser el responsable de la confiabilidad de los datos

Va a ser el responsable de la performance de la base de datos. Esta se analiza teniendo en cuenta que herramientas se le brinda al administrador en una determinada empresa.

Si la base de datos requiere más software y más hardware el DBA es el encargado de solicitar a la empresa más recursos.

#### Programador/desarrollador de base de datos

Función: crear e implementar bases de datos.

Recursos a su cargo: la base de datos, servicios y APIs relacionados.

- Determina el mejor DBMS para un sistema en particular.



- Diseña y crea los objetos de la base de datos.
- Desarrolla componentes de software (vistas, procedimientos, funciones, triggers, etc.)
- Asegura la eficiencia del sistema y su performance.

Puede manejar varias tecnologías de DB.

El programador va a ser responsable de crear e implementar una bdd (por crear se hace referencia a escribir “CREATE DB... etc.”)

Solo se va a preocupar por la performance del sistema que se encuentre desarrollando, lo de mas no es de su incumbencia

El programador va a saber que tablas deben contener muchas filas y cuáles no. Entonces si llegase a ocurrir un problema, este va a identificar cual es la tabla que está guardando más información que la que debería (el DBA no va a saber esto porque no es el encargado de implementar los sistemas)

Hay consultas de bdd que no tienen buena performance, pero como solo se ejecutan MUY pocas veces al año el programador no va a invertir más tiempo en mejorarlas porque no es algo que se haga seguido (devuelta, como el DBA no es el que lo implementa NO se daría cuenta de esto).

## Modelo de datos: abstracción

*Abstracción de datos* se refiere a la eliminación de los detalles de la organización y almacenamiento en favor de las características esenciales para una comprensión mejorada de los datos.

*Modelo de datos* –colección de conceptos que pueden usarse para describir la estructura de una DB- provee los medios para lograr dicha abstracción.

*Estructura de la base* de datos refiere a los tipos de datos, relaciones y restricciones que aplican a los datos.

## *Constraints*

Las bases de datos relacionales constan de muchos tipos de relaciones.

Las restricciones (constraints) dependen del estado de la DB.

Las restricciones se pueden clasificar en:

- Inherentes al modelo de datos o implícitas.

Las impuestas por tipos de datos. “Solo números”

- Expresadas en el esquema o explícitas.

Generadas por relaciones (PK-FK), constraints, check. “El cliente ya debe existir”

- Semánticas o reglas de negocio (se manejan en la aplicación).

2da unidad al 70% no válido en Mendoza ni Chandon.

## *Definición y tipos de datos*

Tabla, fila y columna corresponden a relación, tupla y atributo del modelo relacional.

CREATE es la declaración para generar esquemas, tablas, y otros objetos de la DB.

Se utiliza la declaración CREATE TABLE para generar relaciones nuevas, definir sus atributos y restricciones iniciales.

```
CREATE TABLE nombreEsquema.nombreTabla (
    Campo1 tipoDatos [restricciones]
    ...
);
```

#### *Esquemas (schema)*

Equivalen a un namespace: jerarquía de objetos de la DB

Las primeras versiones de SQL no incluían el concepto de esquema.

Un esquema se identifica por nombre.

(Postgres admite borrado en cascada de un esquema y todo lo que contiene.)

```
CREATE SCHEMA nombreEsquema;
```

#### *Catálogos*

Catálogo: un conjunto de esquemas en un entorno.

Un entorno es típicamente una instalación del RDBMS.

El catálogo contendrá información de todos los esquemas de un entorno (o instancia).

Pueden definirse restricciones entre esquemas del mismo catálogo.

En un sistema de computación pueden instalarse varias instancias, tal que cada una escuche en un puerto distinto.

Un RDBMS puede permitirnos trabajar con varios catálogos a la vez

#### *Catálogo.Esquema.Objeto*

Las conexiones al RDBMS se realizan en un contexto de seguridad (usuario).

Cada usuario puede tener un catálogo y esquema predeterminado.

Cada objeto tiene un nombre completo compuesto por

**catálogo.esquema.nombreObjeto.**

Se puede trabajar con catálogo y esquema predeterminados.

#### *DDL: Data Definition Language*

Define y modifica el esquema de la base de datos.

#### *ALTER*

- Modifica un objeto de la base de datos.

### TRUNCATE

- Elimina el contenido de una tabla pero mantiene la estructura.

### DML: Data Manipulation Language

Permite manipular los datos: agregar, eliminar, modificar.

### INSERT

Crea registros en una tabla.

### UPDATE

Modifica uno o varios registros en una tabla.

### DELETE

Elimina uno o más registros de una tabla.

### MERGE

Realiza inserción, actualización o borrado según el resultado de una junta con una tabla de origen.

Los motores modernos no hacen distinción entre DDL, DML (ni DQL –consultas-, DCL –control-, TCL –transacciones-).

### Tabla, fila y columna

Se utiliza la declaración CREATE TABLE para generar relaciones nuevas, definir sus atributos y restricciones iniciales

```
CREATE TABLE nombreEsquema.nombreTabla (
    Campo1 tipoDato [restricciones]
    ...
);
Ejemplo:
CREATE TABLE ddbba.alumno (
    Apellido char(50)
);
```

### Tipos de datos

Existen tipos básicos definidos en ANSI y otros provistos por el RDBMS.

- Enteros (1 byte: tinyint; 2 bytes: smallint; 4 bytes: int; 8 bytes: bigint).
- Booleano: bit.
- Punto fijo decimal: decimal / numeric. Se define total de dígitos y los reservados para decimales (total, decimales).
- Fecha y hora: datetime, smalldatetime, date, time
- Punto flotante: simple y doble precisión.
- Cadenas de carácter: char. Si admite Unicode: nchar. Si son de longitud variable: varchar/nvarchar.

## *Integridad referencial, claves primarias y foráneas, restricciones.*

**Integridad referencial:** mecanismo por el cual el RDBMS asegura que un valor que aparece en una relación para un conjunto de atributos determinado también aparezca en otra relación para un cierto conjunto de atributos.

Cada relación tendrá **restricciones** de integridad referencial.

Las operaciones de inserción, borrado y actualización se **comprueban para asegurar que no violen** las reglas de integridad referencial.

**Clave primaria:** conjunto mínimo de campos que identifican de forma única un registro. No pueden ser nulos (restricción implícita).

Es recomendable definir una clave primaria (simple o compuesta) para toda tabla, incluso las tablas temporales y variables.

En cada inserción se validará que no esté repetido (restricción implícita).

Será el(los) campo(s) del índice clúster, dictando el orden físico de los registros.

Es buena idea que sea un valor inherentemente creciente.

Es mala idea utilizar un campo que se pueda modificar.

```

CREATE TABLE ddbba.alumno (
    DNI      int primary key,
    Apellido char(50)
);
CREATE TABLE ddbba.alumno (
    DNI      int,
    Apellido char(50),
    constraint pk_alumno primary key clustered (DNI)
);

-- dos formas de lograr lo mismo

```

*Se podría definir un índice no clúster para la PK y un índice clúster para otro(s) campo(s).  
El default es que sea el clúster.*

**Clave foránea:** ocurrencia en una tabla dada de la clave primaria de otra tabla dada que registran una relación única entre ambas. Referencia los atributos que forman la clave primaria de la tabla referenciada.

Se asume que no admite valores nulos.

Generan una restricción implícita.

Puede indicarse una operación de actualización o borrado en cascada.

No se generan índices automáticamente ante su creación.

Es buena idea usar un criterio definido para identificar los campos.

Es importante asignarles el mismo tipo de dato.



```
CREATE TABLE ddbba.comision (
    ID      int identity(1,1) primary key,
    IdMateria  int REFERENCES ddbba.materia (Id)
);
-- ejemplo de FK de un campo

CREATE TABLE ddbba.examen (
    DNI      int,
    IDCurso  int,
    Nota     tinyint,
    CONSTRAINT FK_InscripcionComision (IDCurso, DNI)
    REFERENCES ddbba.Curso (ID, DNI)
);
-- ejemplo de FK de más de un campo
```

**Restricciones:** Reglas que el RDBMS aplicará a los valores de la tabla. Además de las que se generan por claves primarias (valor único) y claves foráneas (valor existente en la tabla referenciada), pueden definirse otras restricciones.

#### UNIQUE

Impide la carga de duplicados.

#### CHECK

Puede utilizarse para restringir valores (por ejemplo solo positivos), o que el valor se ajuste a un formato específico.

#### NOT NULL

Impide que el valor quede en nulo.

#### DEFAULT

Determina un valor por defecto.

```
Create Table ddbba.alumno (
    DNI      int CHECK (DNI > 0),
    nombre   char(20) UNIQUE,
    patente  char(7) ,
    CONSTRAINT CK_patente CHECK (
        patente LIKE '[A-Z][A-Z][0-9][0-9][0-9][A-Z][A-Z]'
        OR patente LIKE '[A-Z][A-Z][A-Z][0-9][0-9][0-9]’
        ) - solo autos!
);
```

#### NULL

Puede interpretarse como dato ausente, desconocido o que no aplica.

- No existe una teoría relacional satisfactoria que lo incluya.
- Cada RDBMS puede hacer un manejo distinto.

La comparación NULL == NULL siempre será FALSE.

Existen condiciones del tipo valor is [not] null para ello, o reemplazos del tipo isnull(valor,0)

- En un JOIN no se muestran los registros con NULL en un campo de la junta.
- ¿Se cuenta un NULL en un count()?
- ¿Se contabiliza en un AVG()?
- Concatenar cadenas de texto y NULL... generará un NULL

### *Consultas básicas*

```
SELECT    campo1, campo2, ..., campoN
FROM      esquema.tabla
WHERE     condición
```

**SELECT**: corresponde a la proyección en álgebra relacional.

**FROM**: corresponde al producto cartesiano del álgebra relacional.

**WHERE**: corresponde al predicado selección del álgebra relacional.

Primero se resuelve el FROM, luego el WHERE y finalmente el SELECT.... ¿O no?

**NOO en todas las consultas y todos los motores**

### **CREATE**

- Crea una base de datos o algún objeto (tabla, vista, índice, SP, trigger, etc.)

### **DROP**

- Elimina una base de datos o algún objeto.

### *Juntas y uniones*

```
SELECT    t1.campo1, t2.campo1
FROM      esquema.tabla t1 INNER JOIN
                  esquema.tabla2 t2 ON t1.campoX = t2.campoY
```

### **UNION**

```
SELECT    t3.campo1, t4.campo1
FROM      esquema.tabla3 t3 LEFT JOIN
                  esquema.tabla4 t4 ON t3.campoX = t4.campo
```

**UNION**: Combina resultados. Suprime duplicados salvo que se indique ALL.

**JOIN**: Cruza resultados (variantes LEFT, RIGHT, OUTER).



<b>INNER JOIN</b> 	<code>SELECT * FROM A INNER JOIN B ON A.key = B.key</code>	<b>RIGHT JOIN (sin intersección de A)</b> 	<code>SELECT * FROM A RIGHT JOIN B ON A.key = B.key WHERE A.key IS NULL</code>
<b>LEFT JOIN</b> 	<code>SELECT * FROM A LEFT JOIN B ON A.key = B.key</code>	<b>FULL JOIN</b> 	<code>SELECT * FROM A FULL OUTER JOIN B ON A.key = B.key</code>
<b>LEFT JOIN (sin intersección de B)</b> 	<code>SELECT * FROM A LEFT JOIN B ON A.key = B.key WHERE B.key IS NULL</code>	<b>FULL JOIN (sin intersección)</b> Admite también FULL OUTER JOIN 	<code>SELECT * FROM A FULL OUTER JOIN B ON A.key = B.key WHERE A.key IS NULL OR B.key IS NULL</code>
<b>RIGHT JOIN</b> 	<code>SELECT * FROM A RIGHT JOIN B ON A.key = B.key</code>		



## SQL es un lenguaje declarativo

Se indica qué queremos obtener... no la forma de hacerlo.

El optimizador de consultas analiza la consulta, genera un plan de ejecución, lo guarda en caché y lo pasa al execution engine.

Analizar el plan de ejecución nos permite ver de qué forma se ejecuta la consulta.

En base a eso podemos optimizarla.

Podemos pensar en el plan de ejecución como un encadenamiento de funciones o procesos donde cada uno accede a los datos y/o realiza una tarea de preparación.

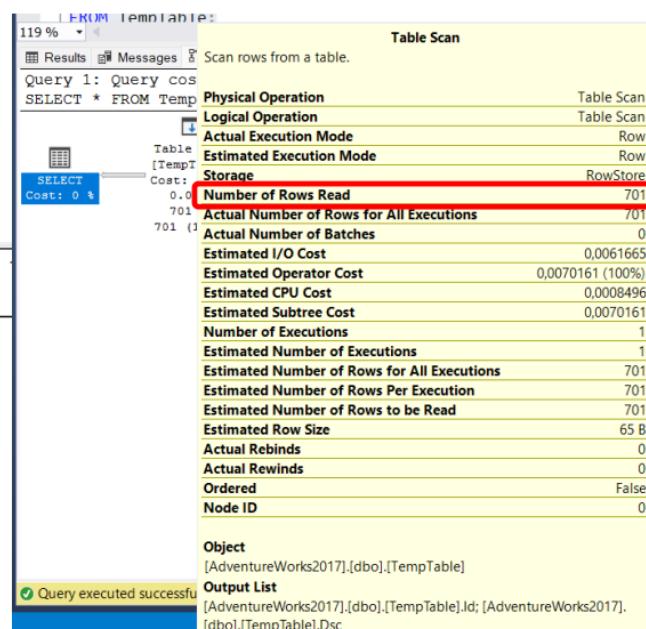
```
SELECT *
FROM TempTable;
```

Esto le pedimos

```
Query 1: Query cost (relative)
SELECT * FROM TempTable
```

```
Table Scan
[TempTable]
SELECT
Cost: 0 %
0.000s
701 of
701 (100%)
```

Esto ejecuta



## Subconsultas

Expresiones del tipo `SELECT...FROM...WHERE` que se anidan en otras expresiones `SELECT...FROM...WHERE`.

Pueden usarse subconsultas en el where, en el from y en el select.

También pueden utilizarse en otras sentencias DML: update, delete, etc).

```
SELECT campo1, campo2  
FROM esquema.tabla1  
WHERE campo3 in (    SELECT campoX  
                      FROM esquema.tabla2  
                     WHERE campoY IS NULL)
```

### Vistas

Tipo de dato derivado: es una tabla derivada (de tabla(s) o vista(s))

- Solo se almacena la consulta.
- Se puede consultar igual que una tabla.
- Se puede utilizar para Insert, Update, Delete con limitaciones.
- Permite mantener retrocompatibilidad con estructuras modificadas.
- Pueden usarse para limitar el acceso de usuarios a campos o registros
- La opción SCHEMABINDING bloquea cambios en las tablas utilizadas.

```
CREATE OR ALTER VIEW ddbba.cursosCopados  
WITH SCHEMABINDING  
AS  
      SELECT      nombreCatedra, diaCursada,  
                  turno, sum(inscriptos)  
      FROM        ddbba.cursos  
      WHERE       puntuacion > 3  
      GROUP BY   nombreCatedra, diaCursada, turno
```

### Funciones de agregado

Realizan operaciones de resumen sobre grupos de filas.

- Contar, sumar, mínimo, máximo, promedio.
- Debemos indicar el criterio de agrupado (GROUP BY).
- No distinguen registros duplicados (uso de DISTINCT).
- COUNT genera un tipo entero, las demás funciones respetan el tipo de dato del campo resumido.
- Se puede usar HAVING para agregar condiciones sobre el resultado de las funciones de agregado.

```

SELECT      nombreCatedra,
            sum(inscriptos)  I
FROM        ddbba.curso
GROUP  BY  nombreCatedra
HAVING     sum(inscriptos) > 90

```

### *Procedimientos almacenados*

Módulos de programa (o rutinas compiladas) almacenados en la DB.

- Se ejecuta en el mismo RDBMS.
- Son útiles para normalizar el acceso a datos desde distintas aplicaciones.
- Reducen el tráfico de datos entre cliente-servidor.
- Permite parámetros de entrada y salida, de los mismos tipos de datos que los campos de las tablas.
- Cada RDBMS implementa una variante de lenguaje de programación (PL/SQL, TSQL, PL-pgSQL, y más...).
- En PostgreSQL admiten sobrecarga.

```
CREATE OR ALTER PROCEDURE esquema.NombreSP
```

```

    @parametro1 int,
    @parametro2 char(20),
    @parametro3 int=0,
    @parámetro4 datetime OUTPUT
AS
BEGIN
    -- comentario
END

```

### *Triggers*

Tipo especial de stored procedure que se ejecuta en forma automática. Permiten especificar acciones que realizará el RDBMS frente a ciertos eventos y condiciones.

- Los ejecuta el RDBMS antes o después del evento interceptable (DML, DDL).
- Pueden configurarse sobre la DB misma.
- En su ámbito de ejecución se dispone de las tablas inserted y deleted.
- Oracle admite comportamiento FOR EACH ROW (SQL Server no; PostgreSQL SI)
  - No permite modificar otro registro excepto el “each row”



- Oracle admite tiempo como disparador (SQL Server usa Jobs);

```
CREATE OR ALTER TRIGGER esquema.NombreTG ON  
esquema.Tabla  
AFTER INSERT, UPDATE AS -- no tienen parámetros  
BEGIN  
    declare @cantidad int  
    set @cantidad = (Select count(1) from inserted)  
    print @cantidad  
END
```

## *Funciones*

Se pueden clasificar en varios tipos.

- Agregado: Consideradas previamente.
- Incluidas en el sistema (de biblioteca).
- Definidas por el usuario.

Se programan de forma similar a los stored procedures:

- Solo tienen parámetros de entrada (no de salida)
- Admiten un valor de retorno escalar o vectorial.
- Pueden usarse en el SELECT, FROM o WHERE.
- Pueden utilizarse en expresiones.
- Según el RDBMS se invocan asignando su valor a una variable o salida.

```
CREATE OR ALTER Function [dbo].[ValidaCUIT] (@cuit varchar(13))  
returns int  
Begin  
  
    declare @cuitMIO  varchar(11)  
    SET @cuitMIO = ltrim(rtrim(replace(@cuit, '-', '')))  
    SET @cuitMIO = ltrim(rtrim(replace(@cuitMio, '_', '')))  
    SET @cuitMio = ltrim(rtrim(@cuitmio))  
    declare @suma  int  
        ,@contador int  
        ,@retorno  int  
    SET @retorno= -1  
    if @cuitMio <> ''  
    begin  
        select @contador= 1  
        select @suma= 0
```



```
while @contador < 12
begin
    SET @suma = @suma + ((cast(SUBSTRING(@cuitMIO,@contador,1)
        as Int) + 48) *
    case @contador
        when '1' then 5
        when '2' then 4
        when '3' then 3
        when '4' then 2
        when '5' then 7
        when '6' then 6
        when '7' then 5
        when '8' then 4
        when '9' then 3
        when '10' then 2
        when '11' then 1
    End )
    set @contador = @contador + 1
End --fin del while
set @suma = abs(@suma % 11)
if @suma = 3
    SET @retorno = 1

End -- fin del IF
return @retorno
end
```

---

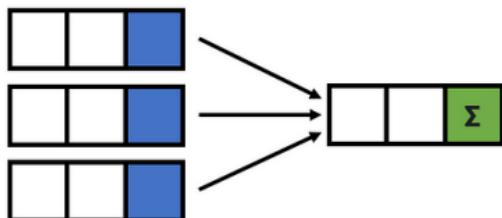
### Window Functions

Estándar a partir de SQL:1999. De la documentación de PostgreSQL:

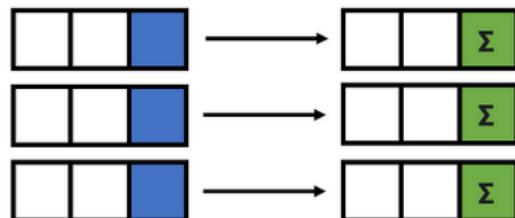
"Realizan cálculos en un conjunto de filas de una tabla que están relacionadas con la fila actual. Detrás de escena la window function puede acceder a más filas que solo la actual."

- Permiten trabajar con columnas agregadas y no agregadas, producen querys más simples.
- Se clasifican en agregado, ranking y valor.
- Devuelven un valor por cada fila de la consulta (en contraste con las de agregado).

### Aggregate Functions (SUM, AVG, etc.)



### Window Functions



- La ventana se define con la cláusula OVER(), pudiendo indicarse una columna específica.
  - No se pueden usar en el FROM, WHERE, GROUP BY o HAVING. Solo en SELECT y ORDER BY
- Ejemplo: Dada una tabla contenido registros de ventas por ciudad, se desea ver además el promedio del monto de ventas para esa ciudad y ese día.

```

select id, fecha, ciudad, monto,
       AVG(monto) OVER (PARTITION BY fecha,ciudad) as PromedioDiario
  From  ddbba.venta
 order by ciudad,fecha

```

Ejemplos de aplicación:

- Ver el sueldo promedio de distintos puestos y la diferencia entre el sueldo de cada empleado y el promedio.
- Ver el saldo acumulado de una cuenta.
- Determinar los N productos más vendidos (cualquier cantidad N de ítems rankeables).

Compare la simpleza de la sintaxis respecto a la misma consulta sin usar window functions.

#### *Common Table Expressions*

Resultados temporales con nombre.

- Facilitan la comprensión de consultas complejas (especialmente si incluyen subconsultas).
- Su ámbito está limitado a una única consulta SELECT, INSERT, UPDATE, DELETE o MERGE.
- Admiten recursividad. Solo pueden referenciar otra CTE definida antes (no después, por eso no hay problema de recursividad cruzada).
- Se define entre paréntesis con la palabra WITH y se designa un alias.
- Pueden definirse varios CTE para luego usar en una consulta.
- Pueden usarse en vistas.

Ejemplo: Ver cantidad de medallas de oro.

```

WITH oro_maraton AS
(
    SELECT      city,      year,      country
    FROM  olympic_games
    WHERE medal_type = 'Gold' AND sport like 'Marathon%' )
SELECT country,
       count(*) AS Medallas_doradas_maraton
  FROM oro_maraton
 GROUP BY country
 ORDER BY gold_medals_in_marathon DESC;

```



Ejemplo: Obtener la serie de Fibonacci

```
WITH Fibonacci (PrevN, N) AS
(
    SELECT 0, 1
    UNION ALL
    SELECT N, PrevN + N
    FROM Fibonacci
    WHERE N < 100
)
SELECT PrevN as Fibo
FROM Fibonacci
OPTION (MAXRECURSION 0);
```

Recomendación: Hágalo con un bucle. *Recursividad = mala idea.*

### SQL Dinámico

Se utiliza en escenarios donde no es posible contar con la consulta requerida hasta el momento de ser ejecutada.

- Se debe generar una cadena de texto con la sentencia SQL a ejecutar.
- La sentencia se prepara en tiempo de ejecución y se ejecuta en un ámbito separado.
- El código dinámico accederá a la DB pero no a variables del llamador.
- Es importante tomar en consideración el riesgo de ataques de SQL Injection.

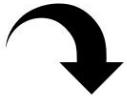
```
DECLARE @CadenaSQL NVARCHAR(MAX) ;
SELECT @CadenaSQL = COALESCE(@CadenaSQL + ' UNION ALL ', '') +
                    + 'SELECT ' +
                    + '...' +
QUOTENAME(SCHEMA_NAME(sOBJ.schema_id)) +
                    + '.' + QUOTENAME(sOBJ.name) + '...' +
AS [Tabla]
                    , COUNT(1) AS [CuentaDeFilas] FROM '
                    + QUOTENAME(SCHEMA_NAME(sOBJ.schema_id)) +
                    + '.' + QUOTENAME(sOBJ.name)
FROM sys.objects AS sOBJ
WHERE
    sOBJ.type = 'U'
ORDER BY SCHEMA_NAME(sOBJ.schema_id), sOBJ.name ;
EXEC sp_executesql @CadenaSQL
```

**PIVOT ENTRA DE CABEZA AL PARCIAAAAAAAAAAAAAAAAAL**

### PIVOT

Útil para presentar resultados más legibles.

También conocido como transposición.



	Total	Ciudad	Mes
1	36799.00	Buenos Aires	1-2023
2	14656.00	Carlos Paz	1-2023
3	15571.00	Claromeco	1-2023
4	43007.00	Iguazu	1-2023
5	31380.00	Rosario	1-2023
6	34226.00	Buenos Aires	2-2023
7	23095.00	Carlos Paz	2-2023
8	17943.00	Claromeco	2-2023
9	46113.00	Iguazu	2-2023
10	26425.00	Rosario	2-2023
11	40616.00	Buenos Aires	3-2023
12	16909.00	Carlos Paz	3-2023
13	11369.00	Claromeco	3-2023
14	46673.00	Iguazu	3-2023

	Ciudad	1-2023	2-2023	3-2023	4-2023	5-2023	6-2023	7-2023
1	Buenos Aires	36799.00	34226.00	40616.00	34988.00	42648.00	26889.00	40552.00
2	Carlos Paz	14656.00	23095.00	16909.00	23808.00	18328.00	18285.00	23159.00
3	Claromeco	15571.00	17943.00	11369.00	15589.00	11578.00	17338.00	15113.00
4	Iguazu	43007.00	46113.00	46673.00	49320.00	54410.00	41578.00	46646.00
5	Rosario	31380.00	26425.00	21720.00	34882.00	19345.00	35252.00	32703.00

Observe que se realiza el pivot por mes y ciudad.

Requiere siempre una función de agregado y un nombre.

```
with VentasResumidas (Total, Ciudad, Mes) as (
    select monto,
        Ciudad,
        cast(month(fecha) as varchar) + '-' + cast(year(fecha) as varchar)
    Mes
        from ddbba.Venta) -- aquí cierra el CTE
select *
from VentasResumidas
pivot(sum(Total))
    for Mes in ([1-2023],[2-2023],[3-2023],[4-2023],[5-2023],[6-2023],[7-2023]) ) Cruzado
```

### Notas de clases

- No puedo ejecutar “crear la base de datos” y “usar viernesTardes” al mismo tiempo.  
Tengo que hacer algo, y cuando termine el otro. Por eso es que se usa el **GO**

```
create database viernesTarde
go
use viernesTarde
go

create schema ddbba
go

create table ddbba.Venta
(
    id      int      identity(1,1) primary key ,
    fecha   smalldatetime,
    ciudad  char(20),
    monto   decimal(10,2)
)
go
```

- ¿Inicializar variables con Set o con Select?

Con set no se puede inicializar varias variables al mismo tiempo, pero con select si



```
-- Inicializo valores y limites
select @FechaInicio = '20230101',
       @FechaFin   = '20230731',
       @DiasIntervalo = (1+DATEDIFF(DAY, @FechaInicio, @FechaFin))
, @contador = 0
-- Las ciudades las hardcodé. Genero otro random y lo convierto

-- Inicializo valores y limites
set @FechaInicio = '20230101',
   @FechaFin   = '20230731',
   @DiasIntervalo = (1+DATEDIFF(DAY, @FechaInicio, @FechaFin))
, @contador = 0
```

- *Set nocount on*

Elimina el mensaje de X registros insertados. Sirve para evitar esos mensajes cuando insertamos muchos registros y viajan por una red o si nadie los va a ver a esos mensajes

```
-- suprimo los mensajes de "registro insertado"
set nocount on
```

- *Top / Limit (depende del motor si es uno o el otro)*

Me sirve para ver una cantidad de registros nomas

```
-- Podemos dar una mirada a los primeros registros
select top 20 * from ddbba.venta
```

- *Truncate*

Elimina los datos de la tabla, y deja la tabla limpia

- *Tiempo de vida de las variables*

Van a “vivir” mientras se ejecuta la consulta. Una vez que se ejecutó se elimina, y si la quiero volver a utilizar la tengo que volver a compilar.

- *Generar valores aleatorios*

```
-- Las ciudades las hardcodé. Genero otro random y lo convierto
while @contador < 1000
begin
    insert ddbba.venta (fecha,ciudad,monto)
        select DATEADD(DAY, RAND(CHECKSUM(NEWID()))*@DiasIntervalo,@FechaInicio),
               case Cast(RAND()*(51)+1 as int)
                   when 1 then 'Buenos Aires'
                   when 2 then 'Rosario'
                   when 3 then 'Bariloche'
                   when 4 then 'Claromeco'
                   else 'Iguazu'
               end ,
               Cast(RAND()*(2000-100)+100 as int)
    set @contador = @contador + 1
    print 'Generado el registro nro ' + cast(@contador as varchar)
end
```

- *Percentil*

Nos permite saber cómo está situado un valor en función a una muestra. Se toma como medida estadística, la cual divide una serie de datos ordenados de menor a mayor en cien partes iguales.



```
select id, fecha, ciudad, monto, ntile(4) OVER (order by monto) as EscalaVentas
from ddbba.venta
order by EscalaVentas,monto
```

- Fibonacci con recursividad

```
WITH Fibonacci (PrevN, N) AS
(
    SELECT 0, 1
    UNION ALL
    SELECT N, PrevN + N
    FROM Fibonacci
    WHERE N < 100
)
SELECT PrevN as Fibo
    FROM Fibonacci
    OPTION (MAXRECURSION 0);
```

- Función

```
-- Generemos registros de notas de examen de un par de alumnos

IF OBJECT_ID(N'ddbba.Nota', N'U') IS NOT NULL
    DROP TABLE ddbba.Nota;
GO

-- o mas simple
--DROP TABLE IF EXISTS ddbba.nota;

create table ddbba.Calificacion
(
    id int identity(1,1) primary key ,
    materia char(20),
    alumno char(20),
    nota tinyint
)
go
```

- CTE

Tiene un ámbito muy limitado. Va existir en la consulta, si tiro una después y quiero usar los datos de esa CTE NOO lo voy a poder hacer.

- ROW\_NUMBER: eliminar duplicados

Devuelve un número de filas, sobre un determinado criterio.

Cuantas veces aparece ese valor: (es pa' eliminar los duplicados)



```
set nocount off
WITH CTE(alumno, nota, materia,
          Ocurrencias)
AS (SELECT alumno,
           nota,
           materia,
           ROW_NUMBER() OVER(
               PARTITION BY alumno, nota, materia
               ORDER BY alumno, nota, materia
           ) AS Ocurrencias
    FROM ddbba.Calificacion
)
delete
from CTE
where Ocurrencias > 1;
```

- *CONSTRAIN*

Cada vez que creamos una Primary key se crea una constrain (depende de que método usemos para crear la tabla es si vamos o no a conocer a esta constrain). Bueno, entonces si queremos eliminar un campo que es Primary key nos va a tirar error porque esa tabla está vinculada con la constrain.

Tonces pa' eliminar un campo que es Primary key primero debo eliminar la constrain que está vinculado (si use el método rápido para crear la tabla, y no conozco la constrain NO voy a poder eliminar el campo de la Primary key).



## UNIDAD N°2: BASES DE DATOS RELACIONALES

### Bases de datos transaccionales

BBDD que tienen como fin el envío y recepción de datos a gran velocidad.

Están destinadas generalmente al entorno de análisis de calidad, datos de producción e industrial.

Objetivo principal: asegurar la consistencia de las transacciones dentro de una base de datos relacional.

En caso de que no se puedan confirmar, *deben ser capaz de revertirlas*. Evitar que las transacciones queden incompletas, es decir, o se realiza la transacción o no pasa nada (vuelve al estado original).

### Características

- Permiten llevar a cabo un gran número de transacciones cortas en línea.
- Manejan datos operativos que provienen de sistemas OLTP (on-line transactional processing)
- Capturar datos sobre el contexto histórico de la transacción para su posterior análisis.
- Están optimizadas para añadir actualizaciones cortas y rápidas en tiempo real.
- Gran velocidad de procesamiento permitiendo realizar consultar y obtener resultados muy rápidamente

### Ventajas

- Permiten asegurar la integridad de los datos (puesto que están diseñadas con propiedades ACID).
- Se puede modificar la información sin poner en riesgo dicha integridad.
- Rápidas y operan con muy baja latencia.
- Permiten replicar datos o recuperarlos de los almacenes en muy poco tiempo.

### Desventajas

- Limitación que tienen para generar informes.
- El historial de datos que facilitan a través de su consulta es limitado a datos actuales o recientes

### ¿Cómo funcionan y para qué sirven?

- Utilizan lenguaje SQL para acceder y modificar datos dentro de la BBDDs.
- Resultan útiles cuando la integridad de los datos es importante, puesto que no permiten que la transacción se complete si uno de los pasos de la misma falla.
- Cada transacción generará un proceso atómico que puede conllevar bien operaciones de inserción, modificación o borrado de datos. Este proceso debe ser validado con un commit o invalidado con rollback.

### Uso y aplicaciones

- Nos permiten obtener datos almacenados de manera muy rápida, y ofrecen una imagen actual de los procesos de la empresa.
- Obtener información para su posterior análisis y toma de decisiones.

- Organizar los datos de las empresas, puesto que permiten dotarles de un esquema común y optimizarlos para el procesamiento de consultas complejas.
- Contextualizar las transacciones llevadas a cabo por aplicaciones operativas.
- Cuando se integran con BBDDs analíticas en una sola plataforma se consigue una mayor consistencia del procesamiento de transacciones

### *Base de datos relacional vs transaccional*

Comprenden dos aspectos de la gestión de datos que van de la mano.

La base de datos transaccional funciona de manera asociada a una base de datos relacional.

La función de la base de datos transaccional es asegurar que las transacciones dentro de la base de datos relacional se cumplen de manera completa o, en caso de haber un error o fallo en el proceso, se reviertan y por tanto no se completen.

### *Ejemplos*

**Transferencias bancarias:** las BBDDs transaccionales se emplean aquí para validar o deshacer la transferencia. Puesto que las transferencias bancarias se desarrollan en dos operaciones distintas, primero se descuenta el dinero en la cuenta origen y luego se suma en la cuenta de destino, si la segunda operación falla, el dinero se devuelve a la cuenta de origen cancelando la operación. El sistema gestor de la base de datos transaccional garantiza que las dos operaciones se cumplen o ninguna de ellas lo haga.

**Sistemas de venta online:** registra una transacción, una venta y un ingreso de dinero para que esta se produzca. Si algún error durante el proceso, la transacción no se completa, y no se produce la venta ni el descuento del dinero.

### Arquitectura de Bases de datos Distribuidas

A nivel arquitectura un sistema distribuido se puede ver como un clúster<sup>1</sup> de servidores

Cuando hablamos de «unir» nos referimos a que comparten recursos de hardware y software; funcionando, así como si fueran un solo sistema unificado.

Es un conjunto de protocolos que permite que varios sistemas de base de datos (clientes, dispositivos, lenguajes) funcionen conjuntamente.

### *Clúster*

#### *Clúster en la nube*

Se trata de la configuración y despliegue de clústeres de servidores virtuales o físicos dentro de un entorno de computación en la nube

#### *Clúster de servidores dedicados*

Se refiere a un entorno de hosting en el cual un servidor físico es asignado exclusivamente a un solo cliente, organización o para un propósito específico

Los clústeres en la nube pueden desplegarse en nubes públicas, privadas o entornos híbridos, dependiendo de las necesidades de seguridad, regulación y desempeño de la organización.

---

<sup>1</sup> CLUSTER: grupo de servidores independientes que trabajan juntos como un solo sistema

## Base de datos - Data Warehouse – Data Lake

Son tres conceptos muy relacionados entre sí, pero no son lo mismo ni sirven para lo mismo, sirviendo cada uno a un propósito distinto.

### *Base de Datos*

Los datos se organizan en forma de tablas, filas y columnas. Este tipo de almacenamiento se caracteriza por ser muy estructurado y estar optimizado para realizar operaciones transaccionales (inserción, borrado, actualización), cuya arquitectura de datos sigue un diseño OLTP (Online Transaction Processing). Es el sistema de almacenamiento de datos que se encuentra debajo de los programas informáticos que usamos y que está diseñado para la introducción y modificación de datos en el sistema.

### *Data Warehouse (DWH)*

Es un sistema de base de datos que se construye encima de todas estas bases de datos OLAP (Online Analytical Processing) y que se usa típicamente para ser la fuente de datos de sistemas de Business Intelligence. El DWH obtiene la información de todos estos sistemas, limpia los datos, los unifica, les aplica reglas de negocio y finalmente los almacena con una estructura OLAP creando así una capa de datos optimizada para ejecutar análisis de datos sobre ella.

**Un DWH es una base de datos con estructura OLAP, que se encuentra por encima de las bases de datos transaccionales y que está diseñada para el análisis de los datos que hay en el sistema.**

### *Data Lake (DL)*

Es un repositorio de datos centralizado, para el almacenamiento de datos estructurados y no estructurados, en tiempo real o no, donde los datos se almacenan tal cual están en el sistema origen, es decir, sin aplicar ningún tipo de transformación sobre los mismos.



Similitudes y diferencias entre un DWH y DL	
DIFERENCIAS	SIMILITUDES
<b>Un DL almacena datos no estructurados.</b>	Son repositorios centralizados de datos.
<b>Un DL no transforma los datos, sino que la almacena tal cual le llegan.</b>	Pueden almacenar datos estructurados.



Un DL no almacena los datos con una estructura óptima para su consulta (OLAP).	Pueden gestionar datos estructurados en Tiempo real.
Un DL tiene capacidad (de procesamiento y de almacenamiento) para gestionar datos no estructurados en tiempo real.	

### *¿Data Lake o Data Warehouse?*

- DWH sirve para el almacenamiento y análisis de datos estructurados.
- DL sirve para el almacenamiento de datos estructurados y no estructurados, aunque no tanto para su análisis.



De esto puede deducirse que podemos tener un DWH sin un DL, pero rara vez tendremos un DL sin un DWH.

Si disponemos solamente de datos estructurados entonces usaremos un DWH.

Si disponemos tanto de datos estructurados como no estructurados, entonces:

- Usaremos un DL para la captura de todo tipo de datos.
- Los datos estructurados los procesaremos y organizaremos adecuadamente para su análisis en un DWH, siendo el DL la fuente de datos del DWH.
- Los datos no estructurados se quedarán en el DL, donde serán accedidos por los procesos que deban analizarlos.

### Motor de base de datos (RDBMS o SGBD)

Proporciona un acceso controlado y un procesamiento de transacciones rápido para cumplir los requisitos de las aplicaciones consumidoras de datos.

### *Características opcionales en MS SQL Server*

**Menos es más. Agregar cosas que NO vamos a usar hace que nuestro sistema sea más vulnerable**

- Replicación de SQL Server
- Machine Learning Services (R y Python) y extensiones de lenguaje (Java)  
**R es un lenguaje pensado para estadística**
- Búsqueda de texto completo
- Data Quality Services



- Servicio de consultas de PolyBase para datos externos: es un componente opcional. A partir de SQL Server 2019, también está disponible el conector de Java para orígenes de datos HDFS.
- Cliente de calidad de datos
- Integration Services  
**Tiene soluciones para automatizar**
- Componentes de conectividad
- Modelos de programación
- Herramientas de administración
- SQL Server Management Studio (SSMS)
- Componentes de documentación

#### *Análisis previo a la instalación*

- ¿Qué hardware se dispondrá para el sistema?  
**Hay motores que tienen restricciones (cantidad de núcleos que va a soportar)**
- ¿Se trata de una VM o bare metal? ¿Se alojará on premises o en la nube?  
**VM: máquina virtual (máquina virtual: cuando hablamos de un sistema operativo que le da soporte a otros sistemas).**

**Bare metal:** es como metal desnudo; instalar el sistema operativo directamente en el hardware / fierro en vez de usarlo en una máquina virtual (**bare metal:** cuando el sistema operativo es el único que reside en esa computadora)

**Esto es importante saberlo porque cuando trabajamos con VM tenemos que saber con qué va a competir mi máquina virtual por los recursos.**

**Nube: si no lo tengo en las instalaciones, y me conecto vía internet, está en la nube**

- ¿Cuánta memoria principal dispone el sistema?
- ¿Cuántos núcleos de CPU tiene el sistema?
- ¿Qué otros servicios proporcionará el mismo sistema? ¿Será también servidor web, fileserver, etc.?
- ¿Se utilizará el sistema para aplicación de ciencia de datos?
- Se harán réplicas de alguna DB?
- ¿Qué limitaciones tiene la versión de motor que se está instalando?  
**Si trabajo con sql express voy a tener una limitación con la memoria**
- ¿Se harán respaldos en el mismo sistema?  
**No es una buena práctica**
- ¿Cómo se conectarán las aplicaciones cliente (si las hay)?  
**¿Necesariamente las aplicaciones cliente se van a conectar vía red? Tal vez no, tal vez el sistema reside y se conecta en el servidor y no se conecta por fuera del servidor.**  
**Ej: app web e instalo en el mismo servidor la base de datos un servidor web, no necesito que la base de datos se conecte a otro lugar**

#### *Aspectos clave de la instalación*

- Ubicación de las bases de datos de usuario.  
**Son las que creamos**
- Ubicación de las bases de datos de sistema.  
**Son las que necesita el motor para funcionar. No tienen que estar en el mismo lugar**



- Ubicación y parámetro de crecimiento de la base temporal.  
Todos los motores tienen una base temporal. Donde la voy a ubicar y como va a crecer afecta mucho a la performance.  
La base temporal es la que va a usar cuando se quede sin memoria, por eso es TAN importante.
- Ubicación de los respaldos.  
Al pedo voy a tener el respaldo de mi base de datos en la misma base de datosss. Si se me rompe, y ahí tenía el respaldo ¿Cómo accedo?  
Si lleno la partición donde tengo la base de datos se me rompe todo. Si tengo el backup aca, en la misma partición, no voy a poder acceder al respaldo de mi base de datos.

Planifique cuidadosamente esta configuración considerando:

- Almacenamiento rápido para DBs.
- Almacenamiento rapidísimo para la DB temporal.
- Capacidad mayor para respaldos.
- Uso de RAID donde sea posible.  
**RAID NO ES EL INSECTICIDA** (matriz redundante de discos independientes, por sus siglas en inglés) es un tipo de almacenamiento en el que los datos se escriben en varios discos dentro de un mismo sistema.  
Si nos llega a fallar un disco, al utilizar RAID el sistema no va a perder continuidad ni datos.
- Evitar el riesgo de llenar la partición de sistema (default).
- Instancia predeterminada o con nombre  
Podemos instalar varios motores de bases de datos en la misma máquina. Si instalo una sola vez va a tener un nombre por defecto, si instalo más de una vez voy a tener que asignar un nombre.
  - Un mismo servidor puede alojar más de una instancia.  
Puedo descargar varias veces el motor (descargo “muchos motores”). CUIDADO CON ESTO porque al descargar varias instancias estas van a competir por los recursos del sistema. Tengo que definir muy bien los recursos de cada una
  - Cada una será un servicio independiente compitiendo por recursos.
- Servicios  
Si no sé qué hace no lo instalo.
  - Active solo los necesarios.
- Intercalación.
  - Seleccione cuidadosamente el collate que servirá para las bases de sistema y default de las nuevas DB.
- Seguridad (contraseñas y métodos de autenticación).
  - Tome en consideración los métodos disponibles en los cliente
- Conectividad
  - Cada motor utiliza un puerto TCP por default.
    - ⊕ No puede haber dos servicios utilizando el mismo puerto.  
El puerto TCP está reservado para ese servicio
    - Asegúrese de permitir el tráfico en el firewall local.
- Memoria
  - Por default utiliza la totalidad. Esto puede impactar en el resto del sistema.



El motor se come toda la memoria por defecto, si no quiero que haga esto lo tengo que configurar (se la limito). Si no la limito la bdd me puede consumir toda la memoria, y no voy a poder levantar otra aplicación.

- Opciones avanzadas.
  - Mantenga el sistema actualizado.
  - Filestream y características opcionales (Réplica, R, Python, Extracción de texto, etc.): instale solo lo necesario. Menos es más.

## *Conexión a un motor de base de datos*

### Paso1: Conexión al Motor de base de datos

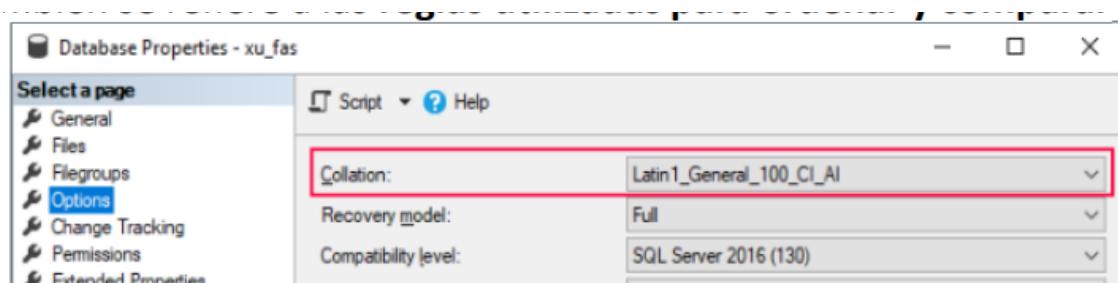
- ❖ Herramientas de introducción
- ❖ Conectarse a Management Studio
- ❖ Para conectarse al motor de base de datos
- ❖ Autorización de conexiones adicionales
- ❖ Crear un inicio de sesión con autenticación de Windows

### Paso 2: Conexión desde otro equipo

- ❖ Habilitar protocolos
- ❖ Cómo habilitar conexiones TCP/IP desde otro equipo
- ❖ Configurar un puerto fijo
- ❖ Configurar SQL Server para escuchar en un puerto específico
- ❖ Abrir puertos del firewall
- ❖ Conectarse al motor de base de datos desde otro equipo
- ❖ Para conectarse al motor de base de datos desde otro equipo
- ❖ Conectarse mediante el Servicio SQL Server Browser

## *Collate – Intercalación*

Intercalación (Collation) hace referencia al patrón de bits utilizado para representar / almacenar cada carácter, y en consecuencia también se refiere a las reglas utilizadas para ordenar y comparar caracteres (Campos de texto).





- Es la propiedad de la BBDDs que proporciona la distinción entre caracteres como acentos, mayúsculas, minúsculas, otros caracteres soportados y reglas de ordenación para los datos que la base de datos es capaz de manejar.
- CS/CI: Case Sensitive / Case Insensitive
- AS/AI: Accent Sensitive / Accent Insensitive.
- Condicionado por la región donde se vaya a instalar y explotar la base de datos o el aplicativo y la configuración de este parámetro es un punto crítico del despliegue

## Consideraciones particulares

- ❖ La Intercalación de las tablas y objetos de sistema.
- ❖ La Intercalación de las nuevas tablas y objetos que se creen en la base de datos. Si no se especifica de forma explícita la Intercalación deseada en la sentencia CREATE correspondiente, se tomará por defecto la Intercalación de la base de datos

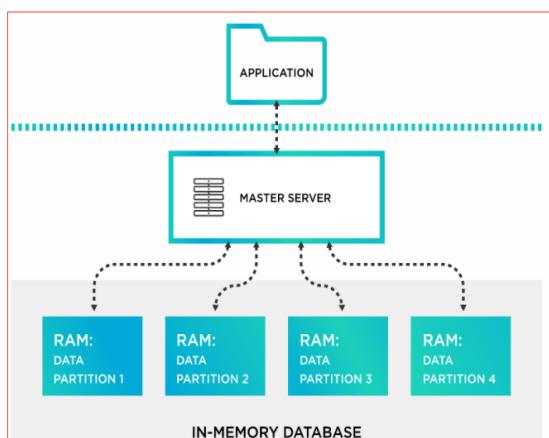
No tener correctamente configurada esta propiedad puede producir comportamientos indeseados. La comparación de cadenas de caracteres puede funcionar de forma distinta a como se pensó o incluso fallar.

Cuando realice comparaciones entre campos de texto de distintas DB conviene hacer explícita la intercalación a aplicar, indicándolo como sufijo de la misma.

```
WHERE campo1 = campo2 COLLATE Modern_Spanish_CS_AI
```

## Bases de datos en memoria

Almacena todos los datos en la memoria principal logrando que el análisis de datos en la BBDDs sea extremadamente rápido en comparación con las bases de datos tradicionales que utilizan almacenamiento secundario.



## ¿Por qué se necesitan BBDDs en memoria?

Con la "internet de las cosas" (IoT) y el crecimiento de las soluciones basadas en la nube, las organizaciones necesitan procesar datos en tiempo real. Millones de dispositivos como monitores de salud y seguridad generan datos cada segundo. Los casos más comunes son los siguientes:

- ❖ Necesidad de aprovechar las ventajas en tiempo real de Big Data.



- ❖ Necesidad de recopilar datos periódicamente y accederlos rápidamente.
- ❖ La persistencia de datos no es un gran problema.

BBDD en memoria ≠ BBDD en disco	
BBDDs en Memoria	BBDDs en Disco
Cada operación de lectura/escriturase realiza directamente en memoria (rápida).	Cada operación requiere lectura/escritura desde almacenamiento masivo, lo que implica una operación de Entrada / Salida (lento).
Los Datos se pueden perder en una falla de energía (volátiles), y deben usar técnicas adicionales para evitar la pérdida de datos	Los Datos no se pierden con una falla de energía (persistentes).
Estructura de almacenamiento simple ya que el acceso aleatorio es altamente eficiente	Estructuras de almacenamientos complejas, logrando acceso eficiente al dato.
Pueden ejecutarse de manera eficiente en la memoria de los dispositivos integrados	Solo pueden ejecutarse en sistemas con dispositivos de almacenamiento secundario.
Ideales para Aplicaciones con pocos requerimientos de datos.	Son usadas en Aplicaciones de grandes volúmenes de datos
Usadas fuertemente en Aplicaciones basadas en el acceso a internet	Usadas fuertemente en aplicaciones de gestión de clientes y usuarios.

### *Implementación de SQL Server*

Debemos incluir un FILEGROUP en la base de datos que sea para tal fin.

- Podemos hacerlo al crear la DB o modificarla.

Las tablas en memoria se generan de la misma forma que las comunes pero se agrega una cláusula específica

- Puede además configurarse para que sean persistentes o no.
- Existen algunas consideraciones especiales (p/e no tienen índice clúster).
- Se desempeñan mejor en ambientes de más escrituras que lecturas.

No deben confundirse con las tablas variables ni con las tablas temporales.

Tabla variable	Tabla temporal	Tabla en memoria
Alcance limitado al módulo (script, SP, trigger) en ejecución.	Alcance a la sesión, excepto temporales globales (mientras se usen).	Alcance idéntico a otras tablas.
Intentará mantenerla en memoria (luego va a tempdb).	Se almacenan en tempdb.	Se mantiene en memoria. Respetan el esquema.
Se indica en DECLARE con @ como otras variables.	Se indica en CREATE con prefijo # o con doble ## las globales.	Se indica en CREATE con una cláusula, pero el nombre no lo distingue.
Su contenido se pierde al quedar fuera de alcance.	Su contenido se pierde al cerrarse la sesión que las creó (o que la usó).	Su contenido no se pierde (salvo en reinicio si no es persistente).
No tienen índices (*) ni estadísticas.	Admiten índices y estadísticas.	Admiten índices no clúster.

## ODBC y JDBC

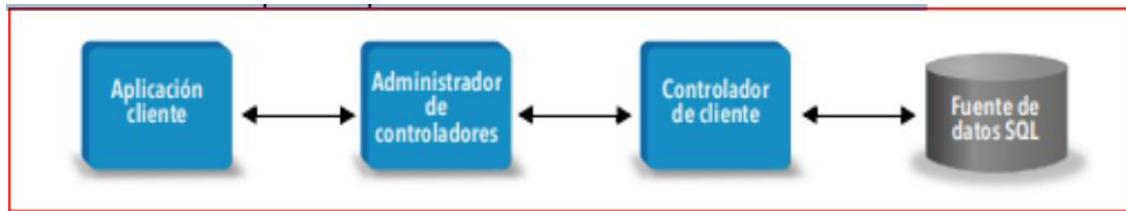
ODBC: Open Data Base Connectivity / JDBC: Java Data Base Connectivity

Normas o interfaces de programación de aplicaciones (API).

*ODBC* es una API para aplicaciones escritas en el lenguaje C.

*JDBC* es una API similar para el lenguaje Java.

Permiten la comunicación con el motor de base de datos. Implementan los protocolos de comunicación necesarios para ejecutar operaciones sobre la base de datos. Exponen una API, es decir una serie de métodos, objetos y funciones con los que nos podemos comunicar con la base de datos.



Estas API ofrecen a las aplicaciones cliente un lenguaje común para interactuar con diversas fuentes de datos y servicios de base de datos. Todas las aplicaciones compatibles con ODBC y JDBC reconocen un subconjunto básico de secuencias SQL (Lenguaje de consulta estructurado).

Si trabaja con SQL, puede utilizar otras aplicaciones (como hojas de cálculo, procesadores de texto y herramientas de generación de informes) para ver, analizar y modificar datos. Mediante las API de ODBC o JDBC, una aplicación cliente se comunica con un administrador de controladores que identifica el controlador de cliente que se va a comunicar con una fuente de datos.

## Formatos de intercambio de datos

Estándares ampliamente difundidos para almacenar y compartir información entre sistemas.

Adherir a un estándar implica que no necesitamos saber de antemano quién utilizará los datos. Reduce además la información adicional requerida para interpretarlo.

Siempre de texto simple.



Poseen estructuras y reglas (sintaxis) tal como los lenguajes de programación.

Suelen existir funciones de biblioteca para generarlos o interpretarlos.

Ejemplos: XML, JSON, YAML, CSV, EDI, etc.

### EDI (Electronic Data Interchange)

Formato de texto simple (ASCII, Unicode, etc.).

Orientado a funcionar computer-to-computer.

Existen varios estándares en uso, cada uno con múltiples versiones.

Debe informarse el estándar adherido en el intercambio.

Se detalla cómo interpretar números, fechas, etc.

Ejemplos: ANSI, ASC, X12, EANCOM, HIPAA, ODETTE, SWIFT, etc.

Usos comunes: facturas, órdenes de compra.

Más información en:

<https://www.edibasics.com/what-is-edi/>



```
ST*810*0001~BIG*20000513*SG427254*20000506*508517*1001~N1*ST*ABC AEROSPACE
CORPORATION*9*123456789-0101~N3*1000 BOARDWALK
DRIVE~N4*SOMEWHERE*CA*98898~ITD*05*3*****30*****E~IT1*1*48*EA*
3**MG*R5656-2~TDS*14400~CTT*1~SE*10*0001~
```

Tabla 1 (de 3).

ST\*810\*0001~

Indica inicio de una factura y asigna un número de control.

BIG\*20000513\*SG427254\*20000506\*508517\*1001~

La factura fue emitida el 13/05/2000 y se le asignó el número SG427254. Corresponde a la orden de compra número 508517, envoi 1001 enviada el 06/05/2000.

N1\*ST\*ABC AEROSPACE CORPORATION\*9\*123456789-0101~

El domicilio de entrega es "ABC Aerospace Corporation" y un identificador fiscal.

N3\*1000 BOARDWALK DRIVE~

Domicilio de entrega Boardwalk drive 1000.

*CSV: Comma separated values*

Formato de texto simple (ASCII, Unicode, etc.).

Interpretación tabular separando registros por filas.

Las columnas se separan con un carácter delimitador (en general coma o punto y coma).

También existe el TSV: separado por carácter tabulador.

Cada registro (fila) debe tener la misma cantidad de campos.

Si no hay valor para un campo se presentan dos delimitadores contiguos.

Pueden contener nombres de campo en la primera fila.

El delimitador se debe escapar o indicar entrecerrillado si es parte de un dato.

```
anio,mes,zona,municipio_id,municipio_nombre,delegacion,genero,cantidad
2019,1,17,06427,La Matanza,San Justo,masculino,74
2019,2,17,06427,La Matanza,San Justo,masculino,73
2019,3,17,06427,La Matanza,San Justo,masculino,66
2019,4,17,06427,La Matanza,San Justo,masculino,59
2019,5,17,06427,La Matanza,San Justo,masculino,91
2019,6,17,06427,La Matanza,San Justo,masculino,49
2019,7,17,06427,La Matanza,San Justo,masculino,69
2019,8,17,06427,La Matanza,San Justo,masculino,58
2019,9,17,06427,La Matanza,San Justo,masculino,52
2019,10,17,06427,La Matanza,San Justo,masculino,50
2019,11,17,06427,La Matanza,San Justo,masculino,41
2019,12,17,06427,La Matanza,San Justo,masculino,44
```

*Ancho fijo, etc.*

Formato de texto simple (ASCII, Unicode, etc.).

Existen variantes con una premisa similar.

Ancho fijo

- Cada registro en una fila.
- Cada columna tiene una cantidad determinada de caracteres.
- Debe determinarse carácter de relleno (típicamente ceros en números y espacios en texto).
- Cada registro tiene la misma cantidad de caracteres.

Tabulados

- Símil CSV pero utilizando el carácter tabulador

*XML: extended Markup Language*

Formato de texto simple (ASCII, Unicode, etc.).

Similar a HTML (evolución de este).

Auto descriptivo. Cada dato se delimita por etiquetas.

<Etiqueta>dato</etiqueta>

No hay etiquetas predefinidas ni una estructura de documento.

Se puede usar sangría (indentación) para facilitar la interpretación humana.

Admite el anidado de tablas.

Ampliamente difundido para aplicaciones de formatos de archivo.

Admite la generación de un archivo de plantilla

```
<response>
  <row _id="row-v4f5~xz3v-vr86">
    <brth_yr>2011</brth_yr>
      <gndr>FEMALE</gndr>
      <ethcty>HISPANIC</ethcty>
      <nm>GERALDINE</nm>
      <cnt>13</cnt>
      <rnk>75</rnk>
  </row>
  <row _id="row-gdep~mr7x-dj3u">
    <brth_yr>2011</brth_yr>
    <gndr>FEMALE</gndr>
    <ethcty>HISPANIC</ethcty>
    <nm>GIA</nm>
    <cnt>21</cnt>
    <rnk>67</rnk>
  </row>
```

[JSON: JavaScript Object Notation](#)

Formato de texto simple (ASCII, Unicode, etc.).

Utiliza una estructura jerárquica y admite valores anidados.

Emplea llaves {} para almacenar objetos y corchetes [] para almacenar vectores.

- Utiliza la lógica de pares clave/valor (key/valué).
  - Los objetos consisten en uno o más pares de claves/valores (key/valué) contenidos dentro de llaves {}.
  - Las claves deben ser cadenas de caracteres contenidas dentro de comillas " ".
  - Los valores deben ser un tipo de información válida (cadena de caracteres, números, vectores, valores booleanos, caracteres nulos u otro objeto).
- Las claves y los valores son separados por dos puntos (:).

Múltiples pares de claves /valores dentro de un objeto se separan mediante comas.

El espacio en blanco no es significativo.

Para representar vectores (de variables escalares, objetos, etc.):

- La clave debe estar seguida por dos puntos (:) y una lista de valores contenidos dentro de corchetes [].

- Un vector en JSON es una lista de un tipo de valor de entre los existentes (caracteres, números, valores booleanos, objetos u otro vector).

Cada valor dentro del vector es separado por una coma.

```
[
  {
    "Posición": "1",
    "Obra": "Poema del Cid.",
    "Enlace_BDH": "http://bdh.bne.es/bnesearch/detalle/bdh0000036451"
  },
  {
    "Posición": "2",
    "Obra": "Beato de Liébana: códice de Fernando I y Dña. Sancha.",
    "Enlace_BDH": "http://bdh.bne.es/bnesearch/detalle/bdh0000051522"
  },
  {
    "Posición": "3",
    "Obra": "Índices de varias obras genealógicas, S. XVIII [Manuscrito]. Signatura: MSS/11490.",
    "Enlace_BDH": "http://bdh.bne.es/bnesearch/detalle/bdh0000192085"
  }
]
```

### *YAML: YAML Ain't a Markup Language*

- Se considera una versión mejorada de JSON.
- Minimalista, más fácil de leer y escribir.
- Utiliza la sangría para definir la estructura (como Python).
  - Pero no admite tabuladores. Siempre espacios (uno o más).
- Objeto YAML se compone de uno o más pares clave-valor.
- El par clave-valor se separa por dos puntos, sin comillas (excepto cadenas).
- Se utiliza un guion para separar elementos en una lista.
- Un documento (un archivo puede contener varios) comienza con tres guiones.

#### %YAML 1.2

```
---
```

**YAML: YAML Ain't Markup Language™**

#### What It Is:

YAML is a human-friendly data serialization language for all programming languages.

#### YAML Resources:

##### YAML Specifications:

- [YAML 1.2:](#)
  - [Revision 1.2.2](#) # Oct 1, 2021 \*New\*
  - [Revision 1.2.1](#) # Oct 1, 2009
  - [Revision 1.2.0](#) # Jul 21, 2009
- [YAML 1.1](#)
- [YAML 1.0](#)

```
# mongod.conf

# Where and how to store data.
storage:
  dbPath: C:\Program Files\MongoDB\Server\4.4\data
  journal:
    enabled: true
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: C:\Program Files\MongoDB\Server\4.4\log\mongod.log

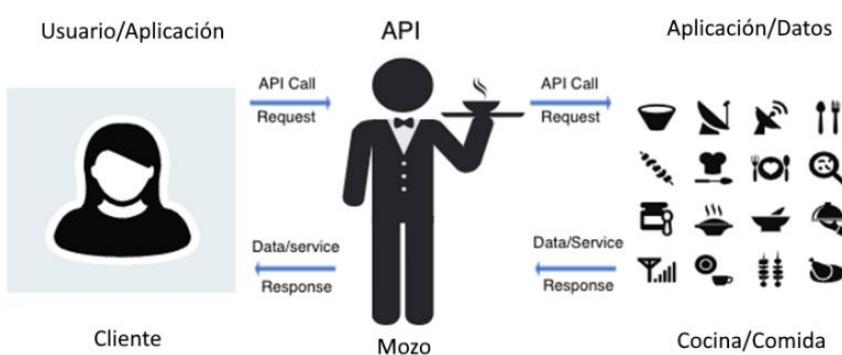
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1
```

*Práctica sugerida*

- Conversión entre formatos (excepto EDI).
  - Por ejemplo: <https://www.json2yaml.com/>
- Operaciones de inserción en SQL por lote (bulk).
- Consulta directa desde SQL de campos XML, JSON.
- Generación de XML, JSON desde SQL.
- Utilización de datasets públicos:
  - <https://catalog.data.gov/>
  - <https://catalogo.datos.gba.gob.ar/>
  - <https://datos.gob.es/>

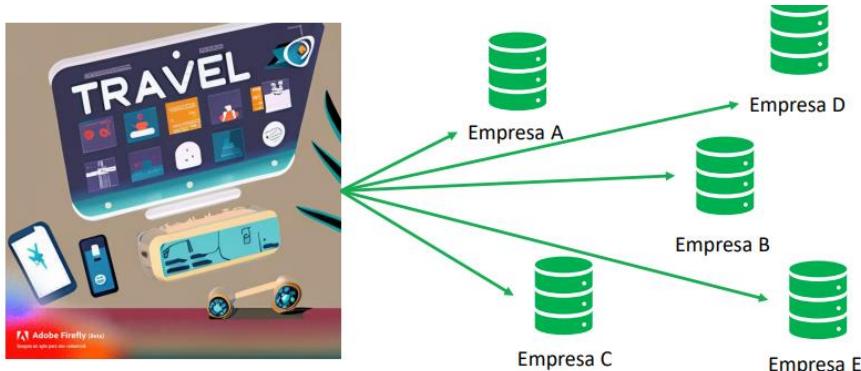
API: Application Programming Interface

Contrato o acuerdo implementado en software que permite el acceso a datos o servicios de un sistema informático desde otro sistema informático.



## Ejemplo

Un sistema de una agencia de turismo se comunica con distintos sistemas de empresas de transporte terrestre, aéreo, marítimo, y permite reservar y comprar pasajes.



## APIs

### APIs abiertas o públicas:

Disponibles para su uso Gral. Algunas requieren una clave (o un token) para controlar las solicitudes que atiende por cliente. También pueden limitar las peticiones según el plan contratado. Ejemplo: Google Maps.

### APIs internas o privadas:

Se utilizan dentro de una organización.

### APIs de socios:

Permiten que una organización intercambie información con proveedores, clientes o socios comerciales.

Tipos de comunicación: SOAP, XML-RPC, JSON-RPC, REST.

Las API REST o API HTTP son las más difundidas.

### APIs: Postman

Postman es una plataforma de APIs para crear y usar APIs. Simplifica cada paso del ciclo de

Vida de las API y facilita la colaboración para crear mejores APIs y hacerlo más rápido

Recursos adicionales en: Postman Academy

<https://www.postman.com/>

## Notas de clases

### Perlitas de Jair:

**Control + R:** es para bajar y subir la pantallita de la salida de la ejecución del SQL

**Tecla ALT:** es para abrir el cursor en varias partes del texto en SQL server

*Shutdown*: es para apagar el motor de base de datos desde adentro del motor

*Sp\_help*: muestra mucha info de la base de datos

*@@*: Son variables del servidor / sistema

*@@rowcnt*: guarda la cantidad de filas que fueron afectadas

#*Tablas temporales*: se crean en la sesión que está activa. No importa que me conecte desde el mismo usuario, si es otra sesión NO voy a poder acceder a esa tabla. Solo desde la sesión que la creo.

##*Tablas temporales globales*: las veo en todas las sesiones, pero cuando la cierro en la sesión que la creo esta tabla desaparece.



## UNIDAD N°3: ASPECTOS AVANZADOS

### El plan de ejecución

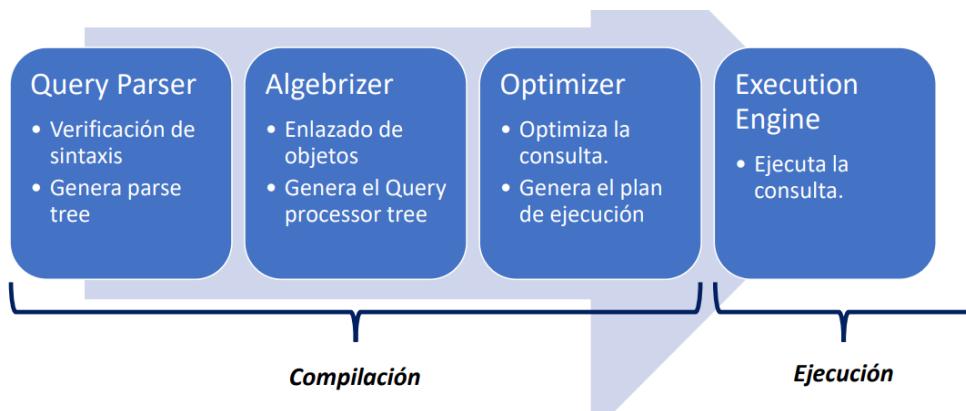
Describe el **conjunto de operaciones** que el motor de ejecución debe realizar para obtener los datos requeridos en una consulta.

Detalla las tablas e índices **que** se accederán, **en qué orden**, de qué **manera** (algoritmo), en qué punto se calculan las funciones de agregado, ordenamiento, estimaciones tenidas en cuenta, reservas de memoria, etc.

Son un excelente punto de partida para la resolución de problemas de performance.

### ¿Qué ocurre cuando se ejecuta una consulta?

Hay dos fases: compilación y ejecución.



El plan de ejecución obtenido se almacena en una caché del motor.

La próxima vez que se ejecute la misma consulta, primero se verifica si existe el plan en caché, en ese caso se reutiliza.

**Query parser:** verifica sintaxis y produce un parse tree (estructura que representa los pasos lógicos necesarios para ejecutar la consulta).

**Query binding:** se resuelven los nombres de objeto, identifica los tipos de datos, determina la ubicación de funciones de agregado.

Los SP postergan (deferr) la resolución de nombres.

¿Qué errores se pueden generar en cada paso?

### ¿Qué ocurre cuando se ejecuta una consulta?

El **Algebrizer** produce un query processor tree (binario) que se pasa al **Query Optimizer**. Se incluye un hash del mismo.

El **hash** se utiliza para verificar si la consulta está en caché.

- De ser así se utiliza la versión compilada en caché.
  - Salvo que se haya modificado alguna estructura (p/e índice) o se hayan actualizado las estadísticas desde la creación del plan.

- Caso contrario el **Query Optimizer** analiza las distintas alternativas para resolver la consulta buscando la de menor costo
  - Utiliza **heurística**: busca la mejor alternativa hasta un punto.

¿Qué ocurre cuando se ejecuta una consulta?

El Optimizador de Consultas (no el desarrollador) decide cómo debe ejecutarse una consulta.

Para obtener buena performance... debemos escribir SQL eficiente

- Utilizar el enfoque de conjuntos para resolver problemas.
- La menor cantidad de sentencias posible.
- Con solo los datos necesarios.
- Evitar consultas ad hoc (parámetros concatenados) en favor de SP y consultas preparadas.

También podemos lograrlo manteniendo estadísticas exactas y actualizadas y promoviendo la reutilización de planes

Plan estimado y plan real (actual)

Solo hay un plan de ejecución para una query en un momento dado.

- El plan estimado es el que genera el Optimizador o el que se encuentra en caché para la query enviada.
- El plan real incluye la información de tiempo de ejecución.
- La carga del servidor en el momento de la ejecución puede modificar el plan empleado.
- Si se crean y usan tablas temporales en la misma query el plan estimado no se podrá generar.

¿Qué mirar en un plan de ejecución?

Se lee de arriba a la derecha... hacia la izquierda y abajo.

Aprenda a identificar los Operadores.

¿Hay table scans?

¿Coinciden las estimaciones con lo real?

Observe que el plan estimado muestra los mismos costos que el real.

¿Dónde se producen los flujos de datos más grandes?

¿Se utiliza swap (page spills)?

¿Cuánta memoria se reserva para la consulta? (Primer op. a la izquierda)

¿Hay operadores en warning?

¿Se logra un nivel de optimización FULL?

Puede ser TRIVIAL, FULL, Timeout, Memory Limit Exceeded.

Índices

Recurso/estructura para acceder rápidamente a ciertos datos.

Se aplican a tablas, incluso a un subconjunto de campos.

En algunos RDMBS / versiones se pueden aplicar a vistas.

Hay bases de datos que tienen un orden dado:

Diccionarios

Directorio telefónico (**páginas blancas y amarillas**) ¿iguales?



Estructuras de datos redundantes: generan copias de campos (excepto el índice clúster).

**El RDBMS siempre utilizará el índice que le permita reducir al máximo el ámbito de la búsqueda.**

**El índice va a establecer un orden para que encuentre más rápido lo que busco**

## Índices clúster

El índice CLÚSTER (o de agrupamiento) es el que determina en qué orden se guardan los registros en la tabla.

- Solo puede haber uno por tabla.
- Si la tabla no tiene índice clúster se le denomina **heap** (montón).
- Puede ser compuesto por varios campos.
- Aunque no se indique la cláusula UNIQUE solo acepta valores únicos.
- Las búsquedas por rango se benefician especialmente de ellos.
- Se puede generar con la designación de un campo como primary key.
- Pero para indicar clave compuesta o sentido de ordenamiento se hace en forma explícita.

**INDICE CLASTER:** si el índice clúster coincide con la clave primaria de la tabla clúster scans (en la vida real es muy difícil encontrar un índice / clave primaria)

Puedo tener un criterio para el índice clúster: si guardo por DNI la búsqueda va a ser muy rápida si esta ordenada por este. Si busco por nombre, esta ordenada por DNI tengo que leer TODA la tabla para encontrar el dato.

**Si es un índice compuesto el orden de los datos es importante.**

## Índices no clúster

Generan una copia de los campos indexados (y los incluidos) manteniendo el orden según el criterio de indexación.

En cada inserción/borrado/modificación se afectará también.

Se generan con la sentencia

**CREATE NONCLUSTERED INDEX nombreIndice ON**

esquema.tabla (campos criterio);

Si una consulta hace referencia a campos del índice puede que no se requiera acceder a la tabla para resolverla.

Verifíquelo haciendo un SELECT solo de los campos del índice (incluidos también) con un ORDER BY de los campos indexados.

Siempre incluirá una referencia al índice clúster (aunque no lo indiquemos).

**INDICE NO CLUSTER:** Si voy a estar buscando siempre el “DNI” me conviene ordenarlo por ese campo, para poder acceder más rápido (indexar por DNI). Copia ordenada por otro criterio, y esto va a ser útil cuando la búsqueda la haga por este criterio

Genero una copia de la tabla, y guarda los datos ordenada de otra forma. Es una paja, porque primero busca en la tabla del índice no clúster, y después para tener todos los datos accede a la tabla original.

Campos incluidos: están incluidos en el índice pero NO implica que el índice esté ordenado en base a ellos. Ejemplo: el índice está creado en base al campo apellido, está ordenado por el campo apellido. Incluye el campo fechaNac pero este campo NO influye en el ordenamiento del índice no clúster.

indice no-cluster		
Apellido,	Apellido	FechaNac
54544 B		1-ene
132132132 C		1-sep
423423423 Centurion		9-ene
432423456 M		5-agosto
345433 V		6-jul
78132312 X		8-jul

create nonclustered index idx\_alumno\_apellido on dbo.alumno (apellido) include (FechaNac)

Esto de añadir campos al índice mejora la optimización porque no hace falta acceder a la tabla para obtener los datos que quiero, me alcanza con el índice. Si accedo a la tabla la bloqueo para los demás que quieren acceder.

## Índices de cobertura

En los índices NO CLÚSTER podemos agregar **campos incluidos**.

- No se utilizan para indexar (no se ordena en base a ellos).
- Pueden mejorar mucho la performance si todos los campos requeridos están incluidos.
- ¡No se necesitaría acceder a la tabla!
- Cuando un índice contiene todos los campos referenciados por una columna se dice que cubre la consulta (índice de cobertura).
- El orden de los campos incluidos (non-key) no afecta la performance.
- No siempre es la mejor solución (p/e si siempre lee todos los campos)

- ¿Ve alguna relación con la recomendación “SELECT \*”?

Con estos índices puedo resolver una consulta SIN tener que acceder a la tabla, me alcanza con el índice.

Índices: Fill factor

Índice de factor de relleno: lo que le dice a la bbdd es decirle cuánto lugar tiene que dejar en blanco. El problema acá está en decidir si dejar espacio en una página para después agregar datos si lo necesito, o llenar toda la página y no dejar espacio para agregar datos.

Fill factor: factor de relleno. Atributo de cada índice que se define al crearlo.

Cuando se llena una estructura de índice se genera una nueva y se enlaza a la anterior, eso se denomina “Page Split”.

“Page Split” implica: (es MUYY pesado)

- Agregar una nueva página.
- Mover la mitad de los datos a la nueva página.
- Marcar como inválidos los datos movidos en la página anterior.
- Actualizar las referencias de páginas existentes para que apunten a la nueva.

Dejar un espacio libre en cada página (hoja) del índice reduce los page Split.

- Si se insertan entradas adicionales (o un campo ocupa más lugar en un Update) se usa ese espacio disponible.

Para evitar un page Split se deja un espacio en blanco para llenar si es que necesito agregar datos.

Reducir los page Split mejora la performance.

¡Pero mantener las páginas incompletas la empeora!

- Tendrá bases de datos más grandes
- Mayor uso de memoria (menos datos entrarán en ella)
- Las consultas demorarán más (más spills).
- Backups y verificaciones DBCC demorarán más.
- Regeneración de índices demorará más.



¿Cuál es el mejor valor?

No aplique el mismo default a todos los índices.

- Si la clave del índice es siempre creciente: **100% (default) es bueno**
- Si la clave no es siempre creciente: evalúe la fragmentación de los índices (más fragmentado es peor) y **reduzca el Fill factor gradualmente**

Las páginas de datos en sql server tienen un ancho pre fijado. Cada motor tiene su tamaño.

El Fill factor es un atributo que le indica al índice cuando debe ocupar de la página, o sea le dice cuánto espacio en blanco va a dejar para poder agregar datos después.

## Optimización de índices

El desarrollador de DB genera índices de acuerdo a las consultas que realizará.

- Mantener actualizado cada índice supone un costo de I/O.
  - ¿Qué pasa si el campo clave de varios índices no clúster se actualiza frecuentemente?  
Y hay que actualizar el la tabla del índice no clúster tambiennnn
- ¿Se utilizan todos los índices?
- ¿Se han generado de forma óptima?
- ¿Se necesitan índices adicionales? ¿Valen la pena?
  - Determine no solo las consultas lentas sino también su frecuencia.
- ¿Hay índices CLÚSTER sobre campos NO ÚNICOS?
  - Para salvarlos el DBMS agrega un uniqueifier de 4 bytes.
- ¿Se podrían agregar campos INCLUDE para mejorar performance?
- ¿Hay campos INCLUDE que rara vez se acceden?
- ¿Se podrían mover al INCLUDE campos indexados que no se usan en las búsquedas?
- ¿Se podría generar un índice no clúster con la cláusula UNIQUE (en campos no clave)?
- ¿Es apropiado el FILL FACTOR?
- Orden lógico y físico dejan de estar alineados con updates, deletes...
- En algunos DBMS regenerar los índices mejora su desempeño.

Determine cuánto tiempo puede estar offline una tabla.

### Rebuild vs reorganize

Diferentes operaciones que reducen la fragmentación de los índices.

#### Rebuild

Impacto alto (bloquea la tabla). Genera de nuevo el índice, puede modificar el FILL FACTOR. Aconsejada para fragmentación alta (>30%).

#### Reorganize

Impacto bajo. Aconsejada para fragmentación baja (<30%).

- Si hay una ventana considerable use rebuild (cuidado con mirroring).

- No tiene sentido ejecutar ambas.
- Siempre mantenga un plan de mantenimiento.

## Estadísticas

Información generada y mantenida automáticamente (\*) por el DMBS acerca de la distribución de los valores de las tablas (histogramas). “Data about the data”.

El optimizador de consultas las utiliza en la generación de planes de ejecución.

Obtiene de ellas la **cardinalidad**: una estimación de la cantidad de filas.

Se crean **automáticamente** en base a campos indexados.

- Por defecto no se muestran todas las filas sino un subconjunto.
- Se puede forzar el muestreo de todas las filas de la tabla.

Constan de:

- **Encabezado**

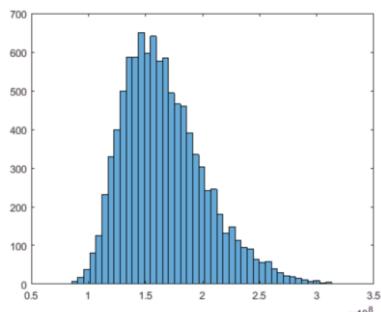
Información general de un conjunto de estadísticas dado.

- **Grafo de densidad**

Selectividad, singularidad (uniqueness) de los datos.

- **Histograma**

Cuenta tabulada de las ocurrencias de valores particulares de hasta 200 puntos elegidos por representar toda la tabla.<sup>2</sup>



## ESTADISTICAS

El motor de bdd hace una estimación de cuantos registros va a leer por cada consulta. Se saca un promedio pa' saber esto con un muestreo. Pero también se puede hacer en base a una estadística (lee todo para sacar esto)

El motor para cada índice no clúster crea una estadística asociada.

No tengo que crear a mano las estadísticas (muy pocas veces lo hago) lo hace el motor

---

<sup>2</sup> Histograma: Representación gráfica de una distribución de frecuencias por medio de barras, cuyas alturas representan las correspondientes frecuencias. (RAE)

Leer las estadísticas tiene un costo, por eso por defecto se genera sobre pocas filas y no sobre todos los registros de la tabla

Las estadísticas solo guardan 200 puntos de muestreo

Genero la estadística al momento de hacer la consulta

Si solicitan un valor muy atípico, en una tabla muy grande, seguramente me dé un valor muy errado. Puedo aumentar el valor del muestreo para que coincida mejor. Si sigue mal la estadística puedo hacer una estadística filtrada por los datos que me interesan.

Puede mejorarse la performance creando estadísticas manualmente si...

- Lo sugiere el Database Engine Tuning Advisor.
- El predicado de la consulta (ver plan) contiene múltiples columnas correlacionadas que no están en el mismo índice.
- La consulta aplica a un subconjunto de datos (estadísticas filtradas).

Existe un SP para actualizar todas las estadísticas de una DB:

**EXEC sp\_updatestats;**

También podemos actualizar para una tabla, por ejemplo con muestreo de toda la tabla:

```
UPDATE STATISTICS ddbba.Venta  
WITH FULLSCAN
```

FULLSCAN lee toda la tabla para regenerar la estadística

*¡Cuidado! Esta sentencia consume muchos recursos*

Opción (remcompile): obliga a recompilar el plan de ejecución; cuando lo obliga a recompilar lo hace con esa nueva estadística; OJO PORQUE SACRIFICO PERFORMANCE EN LA EJECUCION

Recompile: vuelve a ejecutar la consulta; vuelve a ejecutar el plan de ejecución ¿?????????????

Actualización de estadísticas **sincrónica** (default)

- Recomendada para operaciones que cambian la distribución de los datos.
- Truncado, inserción en lote (bulk), actualización de muchas filas (bulk).
- Garantiza uso de estadísticas actualizadas.
- Si no están actualizadas primero se actualizan y luego se compila y ejecuta la query.

Actualización de estadísticas **asincrónica**

- El query optimizar utiliza las estadísticas incluso si no están actualizadas.
- Se ejecutan frecuentemente las mismas consultas (o similares).
- El tiempo de respuesta puede ser más predecible.



- Se producen timeout por demora en compilación por demora en actualización de estadísticas

## Optimización de consultas

Cuento más compleja, más esfuerzo para optimizarlas.

- ¿Hay relaciones o tablas de sobra? Observe las Subconsultas. Minimice los SELECT.
- ¿Se usan los WHERE y JOIN aprovechando los índices?
- ¿Podemos generar CTEs para facilitar el mantenimiento?

Verifique el plan de ejecución.

- ¿Hay algún table scans? Evalúe creación de índices.
- ¿Alguna tabla es un heap? Cree índice clúster.
- ¿Hay mucha diferencia entre filas estimadas y reales? Estadísticas.
- Reduzca o elimine el uso de tablas temporales.
- Las tablas variables pueden tener un índice clúster (aprovechelo).
- ¡Cuidado con el NOLOCK! Obtendrán lecturas sucias.
- Mida las ejecuciones por SP para optimizar selectivamente.
- Verifique los tipos de datos empleados (abuso de float).
  - Realice el testing correspondiente antes de modificar en productivo.
  - No todos los lenguajes tienen tipos de datos 100% equivalentes.
- ¿Realmente necesita UNICODE?
  - Sino, puede usar char/varchar en lugar de nchar/nvarchar.
    - Reducirá espacio ocupado (disco, memoria, backups) a la mitad.
- ¿Realmente necesita VARchar/nVARchar?
  - Si el campo no es realmente tan variable use CHAR/NCHAR.
    - Mejora la performance (no necesita calcular largo del campo).
    - ¡Cuidado! Rellena con espacios al final del texto.
- Examine la calidad del plan de ejecución: ¿requiere partir la consulta en otras más simples?
- Las tablas variables NO tienen estadísticas.

Siempre se asume una fila (dependiendo la versión).

Las tablas temporales SÍ.

Realizar JOINs sobre tablas temporales (especialmente si son grandes) tendrá mejor performance.

- Cursores.

Elimínelos. Tecnología caduca. Bloquea el acceso a las tablas, y además de que no aplican lógica de conjuntos por lo que no tienen buena performances

Si no se pueden eliminar, optimícelos (solo de lectura, forward, ciérrelos).

- Utilice SPs o consultas parametrizadas en lugar de literales hardcodeados.
  - Facilita la reutilización de planes en caché.

Mi consulta tarda mucho...

Sospechosos de siempre: consulta pobremente generada, red, recursos del servidor, coincidencia con tareas de backup.

Habiendo descartado lo demás:

- Actualizar estadísticas
- Actualizar estadísticas con FULLSAMPLE.
- Regenerar índices.
- Crear índices o modificar índices existentes.
- ¿Se necesita una estadística filtrada? Campos con distribución MUY desigual son candidatos.
- Si en SSMS o entorno de desarrollo (VS) funciona rápido y en el cliente no, comparar configuración de entorno.
- ¿Demasiado normalizado?
- ¿Puedo mejorar el HW?

### Tablas temporales

1. ¡No abuse! No siempre son el mejor recurso.
  2. La base tempdb debe estar correctamente tuneada (ubicación, crecimiento).
  3. Solo inserte las columnas y filas necesarias.
  4. Compare la performance de SELECT INTO e INSERT INTO y use la mejor.
  5. Elimínelas tan pronto deje de usarlas. No deje esto al motor.
  6. tempdb es un recurso compartido. Compartir es de nene bueno.
  7. Genere índices si está seguro de que se requerirán.
  8. Las tablas variables sufren menos por bloqueos, son seguras y los rollback no las afectan.
- Ideales para pocos datos.

### Rowstore vs Columnstore

- Las tablas siempre se almacenaron por fila (rowstore)

- Pero al realizar operaciones de agregado normalmente nos enfocamos en determinadas columnas.
- Generar una tabla almacenada por columna (columnstore) puede mejorar muchísimo las consultas, especialmente las de agregado/ventana.
- Los índices para almacenado por columna son ideales en tablas grandes del datawarehouse.

➤ Se comportan mejor si son solo de consulta

## OPTIMIZACION

**++ Difícil la consulta == ++ me tengo que esforzar para mejorar performance**

¿Hay cosas de sobra?

Si la Subconsultas se repite más de una vez meter una CTE

Solo traer en el select lo que necesito

¿Aprovecho bien los índices? Antes de salir a crear índices sin pensar (porque nos los recomienda el motor). Hay veces que los join NO aprovechan los índices

- Join
- Where

Estimado != real = revisar estadísticas

**++ Tablas temporales meto == ++ fuerzo al motor (no toma decisiones el motor)**

Tablas variables tienen un GRAN problema con las estadísticas, esto NOOOO genera estadística, si crecen mucho es una caca la performance (porque solo hace una estimación). Lo que si permiten las tablas variables son los índices cluster, usarlosssssss

## PREGUNTAR DE PARCIAL

Lo primero que tengo que optimizar es la estructura de las tablas

- Float: casi todos lo que lo usan lo usan maaaaaaaaaaaa; no usar punto flotante para boludeces (lo usamos en C porque ese lenguaje NO tiene punto fijo, en los demás lenguajes si lo tiene así que usar eso)
- UNICODE: lo necesito sí o sí? Para guardar una ñ no lo necesito; solo para nombres chinos; la gente de china tiene un nombre en español, así que para guardar sus nombres no necesitan los caracteres chinos;
- Curosres: donde aparezcan hay que tratar de eliminarlos, y si no se puede hay que optimizarlos; por default los curosres vienen declarados para ir pa adelante y para atrás, para optimizarlos los pongo solo pa lectura, escritura y cerrándolos

Tablas temporales

- Eliminarlas cuando la termino de usar, no dejarle todo el trabajo al motor

Existe otro método pa guardar por columnas

Se usa cuando me interesas hacer análisis por campo y cuando las tablas solo son pa lecturas y no se van a modificar (ciencias de datos)

### Transaction Control Language (TCL)

- Conjunto de operaciones que se realizan como una sola unidad lógica de trabajo.
- Dicha unidad debe cumplir con cuatro propiedades básicas ACID para poder ser clasificada como una transacción



Fuente:gravitar.biz

Transacción: una porción de código que se ejecuta debe cumplir con tres características principales:

- ATOMICIDAD
- CONSISTENCIA
- AISLAMIENTO
- DURABILIDAD

Ejemplo: Se tiene dos tablas

- creditCard\_info → contiene nro de tarjeta y el límite de crédito disponible
- Transacción → contiene el nro. De tarjeta, la fecha, el monto de la transacción y el tipo de transacción (Venta o Reversión)

Por cada transacción de venta se decremento el límite de crédito

Por cada transacción de reversión se incrementa el límite de crédito

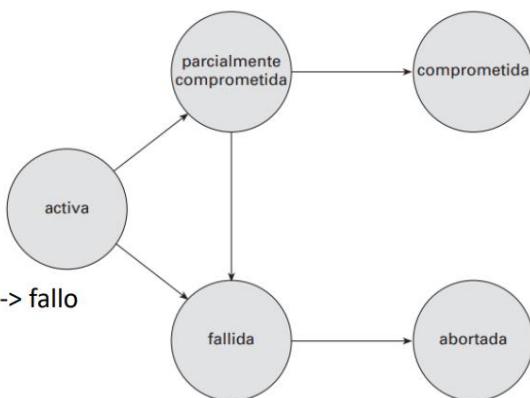
Tarjeta de crédito: tiene un límite, y ese límite es hasta donde podemos gastar. Cada vez que se pasa ese límite es una transacción

#### *Atomicidad*

Para que una transacción se dé por «completada», deben haberse realizado todas sus partes o ninguna de ellas. Si la transacción se aborta no cumple con este principio



```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50; --> fallo
escribir(creditCard_info).
```



### Consistencia

Se basa en la premisa que afirma que una transacción debe llevar al sistema de un estado válido a otro que también lo sea.

La responsabilidad de asegurar la consistencia de una transacción es del programador de la aplicación que codifica dicha transacción.

Tiene que ver con yo veo a la transacción con un estado

```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50;
escribir(creditCard_info)
```



La  
transacción  
se ve como  
**un todo**

### Aislamiento

Asegura que el resultado obtenido al ejecutar concurrentemente las transacciones es un estado del sistema equivalente a uno obtenido al ejecutar una tras otra en algún orden

Toda transacción debe ejecutarse como si fuera única o como si estuviera aislada del mundo.  
Nadie puede tocar esos recursos de la transacción

#### Venta 1

```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50;
escribir(creditCard_info)
```

#### Venta 2

```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50;
escribir(creditCard_info)
```

### Durabilidad

Implica que los datos y cambios en una transacción que ya se ha realizado deben ser permanentes y no puede ocurrir una pérdida de los mismos en el sistema.

Se puede garantizar la durabilidad si se asegura que:

1. Las modificaciones realizadas por la transacción se guardan en disco antes de que finalice la transacción.
2. La información de las modificaciones realizadas por la transacción guardada en disco es suficiente para permitir a la base de datos reconstruir dichas modificaciones cuando el sistema se reinicie después del fallo.

**Que los resultados permanezcan por mucho tiempo en la base de datos**

SQL Server

#### *Transacciones implícitas*

Se inicia implícitamente una nueva transacción cuando se ha completado la anterior, pero cada transacción se completa explícitamente con una instrucción **COMMIT** o **ROLLBACK**.

Para activar Modo implícito: **SET IMPLICIT\_TRANSACTIONS ON**

Para desactivar Modo implícito: **SET IMPLICIT\_TRANSACTIONS OFF**

No es aconsejable este modo dado que puede olvidar de cerrar las transacciones, y esto puede ser origen de bloqueos.

Para ver la configuración actual de **IMPLICIT\_TRANSACTIONS**

```
DECLARE @IMPLICIT_TRANSACTIONS VARCHAR (3) = 'OFF';
IF ((2 & @@OPTIONS) = 2)
    SET @IMPLICIT_TRANSACTIONS = 'ON';
SELECT @IMPLICIT_TRANSACTIONS AS ModoImplicitTranActual;
```

#### *Transacciones explícitas*

Cada transacción **se inicia explícitamente** con la instrucción BEGIN TRANSACTION y **se termina explícitamente** con una instrucción COMMIT o ROLLBACK.

Ejemplo de transacción  
TCL\_0.sql

Ejemplo 2- READ COMMITTED  
TCL\_3.sql y TCL\_4.sql

Ejemplo 1 - READ UNCOMMITTED  
TCL\_1.sql y TCL\_2.sql

Ejemplo 3- REPEATABLE READ  
TCL\_5.sql - TCL\_6.sql - TCL\_7.sql

Ejemplo 4- SERIALIZABLE  
TCL\_8.sql y TCL\_9.sql

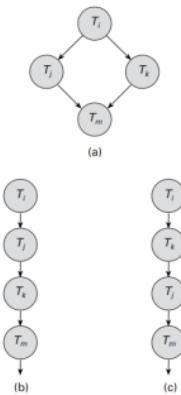


#### *Control de concurrencia*

Trata con los problemas de aislamiento y consistencia del procesamiento de transacciones

- Los esquemas de control de concurrencia que se describen a continuación se basan en la propiedad de secuencialidad.

- Todos los esquemas de ejecución que se presentan aseguran que las planificaciones son secuenciales si es equivalente a una planificación secuencial



Lo que trata de hacer es generar un plan secuencialización de concurrencia para que las transacciones se ejecuten de manera aislada

Sin que el usuario note que está sucediendo atrás en la base de datos

### Bloqueos

- Compartido. Si una transacción  $T_i$  obtiene un bloqueo en modo compartido (denotado por C) sobre el elemento Q, entonces  $T_i$  puede leer Q pero no lo puede escribir.
- Exclusivo. Si una transacción  $T_i$  obtiene un bloqueo en modo exclusivo (denotado por X) sobre el elemento Q, entonces  $T_i$  puede tanto leer como escribir Q.

BLOQUEO: transacción A toma la tabla para leer, y como el bloqueo es compartido, la transacción B puede leer esa tabla.

Si es un bloqueo exclusivo nadie puede tomar la tabla.

Esta decisión se toma dependiendo del contexto: Con el control de concurrencia se trata de evitar

Los niveles de aislamiento definen el grado en que una transacción debe aislarse de las modificaciones de datos realizadas por cualquier otra transacción en el sistema de base de datos.

Cuando se accede a un mismo dato en dos transacciones distintas se pueden dar las siguientes situaciones:

- Lectura sucia (Dirty Read). Una transacción lee datos que han sido escritos por otra transacción que aún no se ha confirmado.

- Lectura no repetible (Non-repeatable Read). Una transacción vuelve a leer los datos que ha leído anteriormente y descubre que otra transacción confirmada ha modificado o eliminado los

datos. Ej: comienzo a realizar una compra por internet, y al ratito comienzo otra comprar. Ambas tienen el mismo límite de crédito de comprar, como la primera transacción compra no termino el límite de crédito nunca fue modificado y cuando se confirma esa compra la segunda se entera que ese límite de crédito es más chico de lo que conocía.

- Lectura fantasma (Phantom Read). Una transacción vuelve a ejecutar una consulta que devuelve un conjunto de filas que satisface una condición de búsqueda y descubre que otra transacción confirmada ha insertado filas adicionales que satisfacen la condición.

Puedo leer datos de la tabla en diferentes momentos de la tabla, que no existen o que es nueva  
 ¿Cómo es el error? ¿Cómo se solucionan? Importante para el parcial

A afip no le gusta que modifique una venta. Realizo una reversión sobre la transacción (revierto lo que se vendió)

No hay una operación de modificación sobre una venta

Para el estándar SQL define cuatro niveles de aislamiento

Nivel	Dirty Read (Lectura sucia)	Non-Repeatable Read (Lectura No Repetible)	Phantom Read (Lectura fantasma)
<b>Read Uncommitted</b>	Es posible	Es posible	Es posible
<b>Read Committed</b>	-	Es posible	Es posible
<b>Repeatable Read</b>	-	-	Es posible
<b>Serializable</b>	-	-	-

#### *SET TRANSACTION ISOLATION LEVEL*

Controla el comportamiento del bloqueo y de las versiones de fila de las instrucciones Transact-SQL emitidas por una conexión a SQL Server

Tener en cuenta que la elección de un nivel de aislamiento de transacción no afecta a los bloqueos adquiridos para proteger la modificación de datos. Siempre se obtiene un bloqueo exclusivo en los datos modificados de una transacción, bloqueo que se mantiene hasta que se completa la transacción, independientemente del nivel de aislamiento seleccionado para la misma

#### *READ UNCOMMITTED*

Especifica que las instrucciones pueden leer filas que han sido modificadas por otras transacciones, pero todavía no se han confirmado. Se trata del nivel de aislamiento menos restrictivo

#### *READ COMMITTED*

Especifica que las instrucciones no pueden leer datos que hayan sido modificados, pero no confirmados, por otras transacciones. Esta opción es la predeterminada para SQL Server

Tiene un bloqueo exclusivo sobre la tabla; nadie lo puede consultar

El comportamiento de READ COMMITTED depende del valor de la opción de base de datos READ\_COMMITTED\_SNAPSHOT:

- Si se establece en OFF (el valor predeterminado de SQL Server), el motor de base de datos usa bloqueos compartidos para impedir que otras transacciones modifiquen las filas mientras la transacción actual esté ejecutando una operación de lectura.
- Si se establece en ON (el valor predeterminado de Azure SQL Database), el motor de base de datos usa versiones de fila para presentar a cada instrucción una instantánea coherente, desde el punto de vista transaccional, de los datos tal como se encontraban al comenzar la instrucción. No se emplean bloqueos para impedir que otras transacciones actualicen los datos.

#### *REPEATABLE READ*

Especifica que las instrucciones no pueden leer datos que han sido modificados, pero aún no confirmados por otras transacciones y que ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que ésta finalice.

#### *SERIALIZABLE*

Especifica lo siguiente:

- Las instrucciones no pueden leer datos que hayan sido modificados, pero aún no confirmados, por otras transacciones.
- Ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que la transacción actual finalice.
- Otras transacciones no pueden insertar filas nuevas con valores de clave que pudieran estar incluidos en el intervalo de claves leído por las instrucciones de la transacción actual hasta que ésta finalice.

#### *SNAPSHOT*

Especifica que los datos leídos por cualquier instrucción de una transacción serán la versión coherente, desde el punto de vista transaccional, de los datos existentes al inicio de la transacción.

La transacción únicamente puede reconocer las modificaciones de datos confirmadas antes del comienzo de la misma. Las instrucciones que se ejecuten en la transacción actual no verán las modificaciones de datos efectuadas por otras transacciones después del inicio de la transacción actual.

El efecto es como si las instrucciones de una transacción obtienen una instantánea de los datos confirmados tal como se encontraban al inicio de la transacción

#### *Observaciones*

SET TRANSACTION ISOLATION LEVEL se aplica en tiempo de ejecución, no en tiempo de análisis

Solo es posible establecer una de las opciones de nivel de aislamiento cada vez, y permanecerá activa para la conexión hasta que se cambie explícitamente. Todas las operaciones de lectura realizadas dentro de la transacción se rigen por las reglas del nivel de aislamiento especificado, a menos que se utilice una sugerencia de tabla en la cláusula FROM de una instrucción para especificar un comportamiento de bloqueo o versiones diferente para una tabla.

Cuando se cambia el nivel de aislamiento de una transacción por otro, los recursos leídos después del cambio se protegen de acuerdo con las reglas del nuevo nivel. Los recursos leídos antes del cambio siguen estando protegidos en función de las reglas del nivel anterior. Por ejemplo, si una transacción ha cambiado de READ COMMITTED a SERIALIZABLE, los bloqueos compartidos adquiridos después del cambio se mantienen hasta el final de la transacción.

Si se ejecuta SET TRANSACTION ISOLATION LEVEL en un procedimiento almacenado o un desencadenador, cuando el objeto devuelve el control, el nivel de aislamiento se restablece en el nivel en efecto cuando se invocó el objeto. Por ejemplo, si se establece REPEATABLE READ en un lote y, después, este lote llama a un procedimiento almacenado que establece el nivel de aislamiento en SERIALIZABLE, el valor del nivel de aislamiento vuelve a REPEATABLE READ cuando el procedimiento almacenado devuelve el control al lote

-- Syntax for SQL Server and Azure SQL Database

**SET TRANSACTION ISOLATION LEVEL**

```
{ READ UNCOMMITTED
  | READ COMMITTED
  | REPEATABLE READ
  | SNAPSHOT
  | SERIALIZABLE }
```

```
USE AdventureWorks2022;
GO
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
GO
BEGIN TRANSACTION;
GO
SELECT * FROM HumanResources.EmployeePayHistory;
GO
SELECT * FROM HumanResources.Department;
GO
COMMIT TRANSACTION;
GO
```

## Transacciones distribuidas

Es una transacción que afecta a varios recursos.

- Para que se confirme todos los participantes deben garantizar que los cambios en los datos serán permanentes. Los cambios deben conservarse a pesar de bloqueos del sistema u otros eventos imprevistos.
- Si alguno de los participantes no cumple esta garantía, toda la transacción da error y se revertirán los cambios en los datos en el ámbito de la transacción.

```
BEGIN DISTRIBUTED { TRAN | TRANSACTION }
  [ _transaction_name | @tran_name_variable ]
  [ ; ]
```

## Argumentos

**transaction\_name**

Nombre de transacción definida por el usuario que se utiliza para realizar el seguimiento de la transacción distribuida en las utilidades de MS DTC. transaction\_name debe seguir las reglas de los identificadores y ser <= 32 caracteres.

**@tran\_name\_variable** Nombre de una variable definida por el usuario que contiene el nombre de una transacción utilizada para realizar el seguimiento de la transacción distribuida en las utilidades de MS DTC. La variable debe declararse con un tipo de datos char, varchar, nchar o nvarchar.

La instancia del DBMS SQL Server que ejecuta la instrucción **BEGIN DISTRIBUTED TRANSACTION** es el **originador** de la transacción y **controla su realización**. Posteriormente, cuando en la sesión se ejecuta una instrucción **COMMIT TRANSACTION** o **ROLLBACK TRANSACTION**, la instancia que controla la transacción solicita a MS DTC que administre la realización de la transacción distribuida entre todas las instancias participantes.

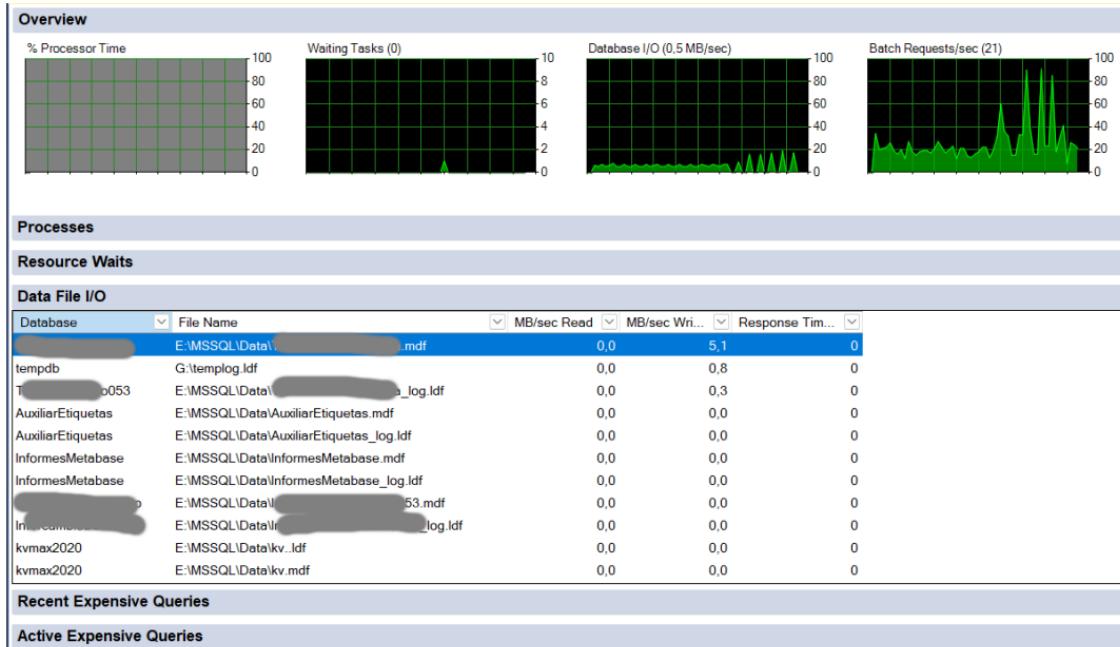
El aislamiento de instantáneas de nivel de instantánea no admite transacciones distribuidas.

La principal manera en que las instancias remotas del Motor de base de datos se dan de alta en una transacción distribuida es cuando una sesión ya dada de alta en la transacción distribuida ejecuta una consulta distribuida que hace referencia a un servidor vinculado.

Este ejemplo elimina un candidato de la base de datos AdventureWorks2022 tanto en la instancia local del Motor de base de datos como en la instancia de un servidor remoto. Ambas bases de datos, local y remota, confirmarán o revertirán la transacción.

```
USE AdventureWorks2022;
GO
BEGIN DISTRIBUTED TRANSACTION;
-- Borramos un registro local
DELETE AdventureWorks2022.HumanResources.JobCandidate
WHERE JobCandidateID = 13;
-- Borramos el correspondiente en un srv remoto
DELETE
RemoteServer.AdventureWorks2022.HumanResources.JobCandidate
WHERE JobCandidateID = 13;
COMMIT TRANSACTION;
```

## Monitoreo



## Métricas de performance

Nos ayudan a entender la utilización de la DB y el consumo de recursos. Pueden incluir:

- Estadísticas de HW (consumo de CPU, memoria, lecturas, escrituras).
- Cuenta de filas.
- Esperas (waits).
- Deadlocks.

Son de gran utilidad para determinar si un servidor tiene pocos recursos, en el desarrollo, resolución de problemas y detección de bugs.

Concepto importante: *baseline*

Necesitamos una referencia para determinar el comportamiento anómalo.

Debemos conocer el sistema y su uso por hora/día/etc.

- ¿Cuántas transacciones se realizan?
- ¿Cuántas filas se agregan en X tabla?
- ¿Qué ritmo de crecimiento tiene la DB?
- ¿Cuál es la duración estándar de algunas operaciones críticas?

### Recuento de filas (row counts)

- Fácil de medir y efectiva para aplicaciones data-driven.
- Puede controlarse por valores absolutos o en términos de porcentajes, o ambos.
- Limitación: no provee información sobre actualizaciones o lecturas.



Supongamos que normalmente se registran 5 mil registros en una tabla de ventas en un sistema dado. Si un día solo se cargan 20 y otro día hay 200 mil, tal vez haya que investigar la causa. (Tal vez feriado o hot sale).

### *Entrada/salida de archivos (database file IO)*

- Permite conocer cuántos datos se leen y escriben en un archivo dado.
- Puede medirse en datos y log, incluso en bases con varias particiones.
- Limitación: Sirven para detectar un problema pero no para determinar una tabla u objeto particularmente afectado.
- Permiten predecir si el crecimiento de la DB será acompañado por los recursos de almacenamiento, red, backup a lo largo del tiempo.

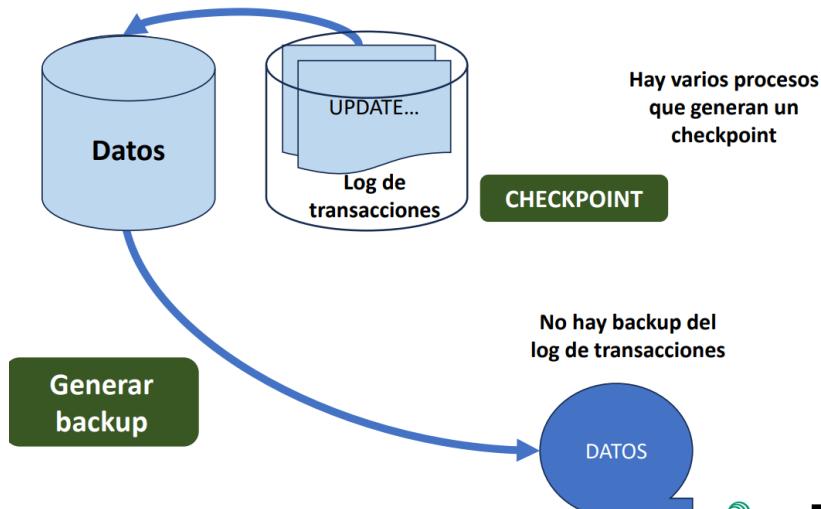
*El modelo de recuperación es una propiedad de la DB que controla cómo se mantiene el log de transacciones.*

- Cuándo se registran
- Si se requiere que se respalden (backup)
- Qué tipos de operaciones de restauración están disponibles.

Son tres: Simple, full, and bulk-logged.

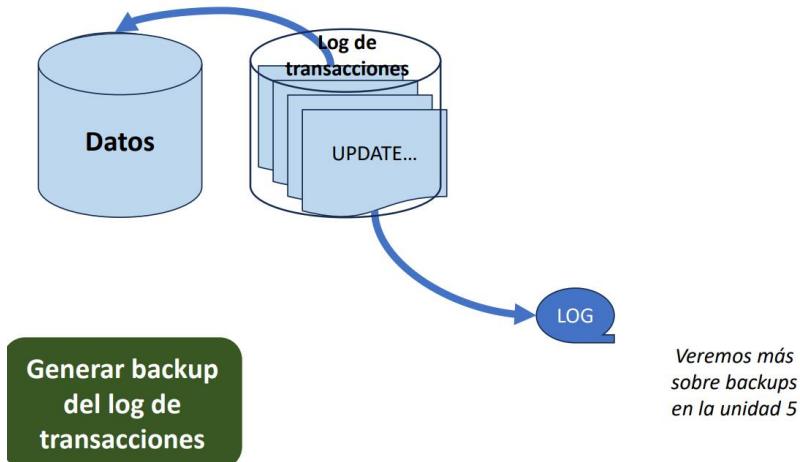
Puede cambiarse en cualquier momento.

### *Modelo de recuperación SIMPLE*





## Modelo de recuperación FULL



Simple	Full
El log se libera al completarse las transacciones.	El log se libera cuando se respalda. Debe realizarse para que no crezca en forma desmedida.
No se respalda el log.	Requiere backups del log de transacciones.
Solo se puede recuperar a estados respaldados en un backup.	Puede recuperarse a un punto específico en el tiempo (por ejemplo antes de un error de una App).
No admite funcionalidades tales como AlwaysOn, Mirroring, log shipping, restauración a un punto en el tiempo.	

## Tamaño del backup del log de transacciones (*transaction log backup size*)

- En una DB en modo de recuperación completo (full) o bulk-logged los cambios sobre ella se reflejarán en el tamaño del log.
  - Si los backups del log de transacciones se toman cada hora y tienen un tamaño dado, un crecimiento anormal de ese tamaño indica una situación a investigar.
  - Ventaja: puede determinarse sin acceder a la DB, sino solo a los backups.

## Bloqueos/esperas

- Una consulta que demora mucho por lentitud o por esperar que otros procesos liberen recursos pueden apuntar a una App a depurar.
  - Las esperas pueden detectarse haciendo muestreo repetidamente.
  - Las wait stats son un recurso de mucha utilidad cuando se requiere investigación adicional, porque pueden asociarse a SPID, texto de la consulta, plan de ejecución en XML, etc.

## Obtención y análisis de las métricas

- Ejecución manual de scripts.
- Trabajo de ejecución automática.
- Envío de email, dashboard, etc.

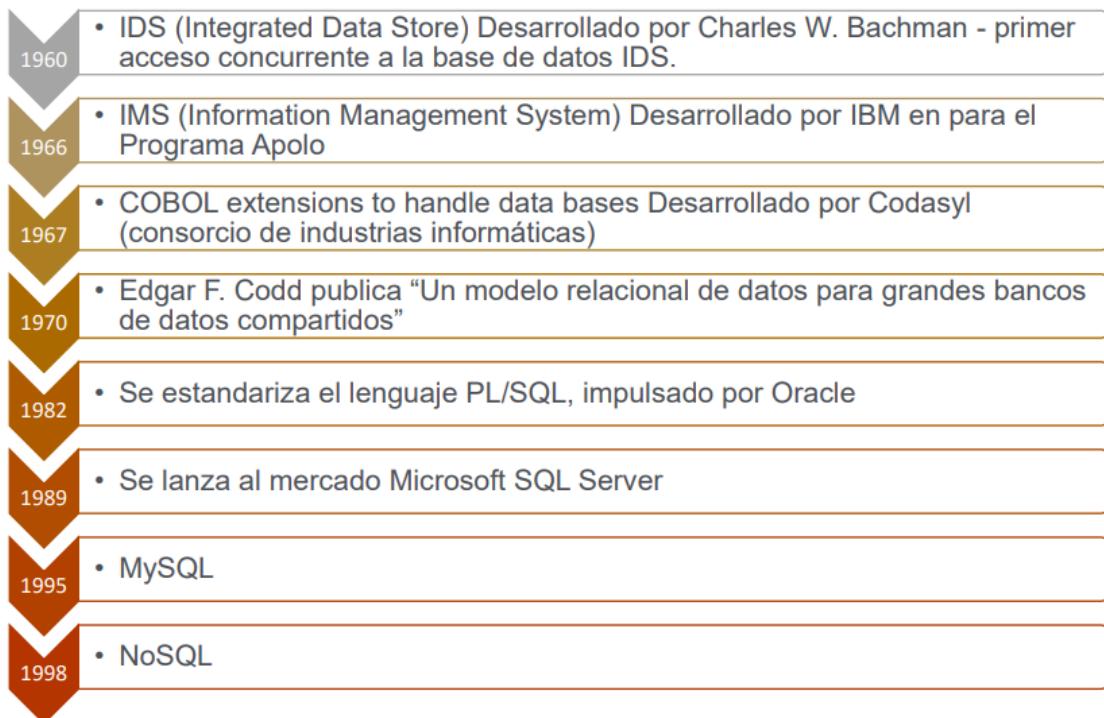
- Software especializado (3rd party).

Los valores fuera de lo normal pudieran indicar problemas a resolver.

#### Notas de clases

- TRANSACCION: tomo el recurso cuando hago una Query/consulta; si solo defino variables NO tome nada todavía, no tome ningún recurso.
- Ctrl + I  
Me muestra el plan estimado de la consulta
- Table scan  
Lee toda la tabla
- Sp\_help
- Index scan  
Lee el índice por completo
-

## UNIDAD N°4: BASES DE DATOS NO RELACIONALES



### Surgimiento de base de datos NoSQL

Problemáticas a las que se enfrenta los RDBMS (sistema de gestión de bases de datos relacionales)

- Costo (Hardware y Gestión)
- Volumen de datos
- Tiempos de respuesta de baja latencia
- Alta disponibilidad
- Problemas de rendimiento

1960: hablábamos de bbdd no re4lacuonales; se crea una bdd para el apolo (nave espacial)

Escalabilidad: Ofrecen servicios en la nube

Flexibilidad: puedo meter cualquier cosa en la bdd

Surgen por nuevas necesidades que hay en el negocio, y que no pueden ser atendidas por las bases de datos no relacionales.

Replicas: NO son copias, son NODOS servidores o clúster, no es igual que es bdd sql

No estructurado: me permite diferentes tipos de datos

¿Dónde guardamos estos datos?

- O lo guardamos en un datalake  
“gran depósito” donde guardo muchas cosas

## PROPIEDADES DE LOS MODELOS DE BBDD

Atomicidad - Basically

Consistencia - Available

Aislamiento - Soft State (datos parcialmente confirmada ¿operaciones bancarias parcialmente confirmadas?)

Durabilidad - Eventually consistent

### Base de datos NoSQL

El auge de este tipo de base de datos se produjo por la necesidad de cubrir las carencias de las RDBMS

#### *Escalabilidad:*

Servicios en la nube - Entornos distribuidos.

#### *Flexibilidad:*

Modelo de datos flexible - Datos semiestructurados y no estructurados.

#### *Alta disponibilidad:*

Garantiza la continuidad del servicio

#### *Rendimiento:*

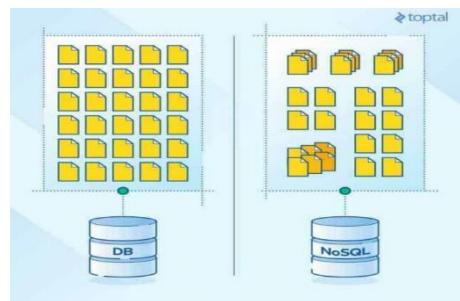
Tiempo de respuesta de las consultas muy superior con respecto a las RDBMS tradicionales

### Características de bases de datos NoSQL

- Los modelos NoSQL no tienen una estructura definida
- La información no se organiza en registros o campos
- Sistemas distribuidos de datos

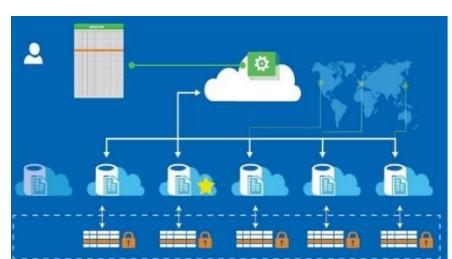
#### *Arquitectura*

- Clúster en la nube
- Clúster de servidores dedicados



#### *La decisión sobre cual arquitectura implementar se basa en:*

- Costo (Capex y Opex)
- Complejidad (infraestructura y el personal técnico calificado)
- Tiempo de implementación (configurar y puesta a punto)



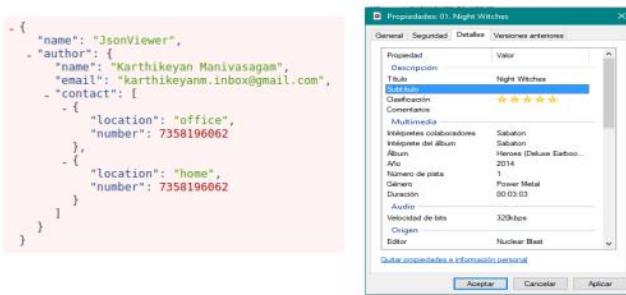
*Datos estructurados, semi estructurados y no estructurados*

Datos estructurados

TIPO DE DATO	TAMAÑO	WRAPPER
Boolean	1 bit	Boolean
Char	16 bits	Character
Byte	8 bits	Byte
Short	16 bits	Short
Int	32 bits	Integer
Long	64 bits	Long
Float	32 bits	Float
Double	64 bits	Double

Datos semi estructurados

Ej: Mapas; metadatos; imágenes



Datos no estructurados

Ej: Contenido de redes sociales (imagen + videos);



*Almacenamiento de datos - Procesamiento de datos*

- Reconocimiento Óptico de Carácteres (OCR)
- Procesamiento del Lenguaje Natural (NLP)



*Datos estructurados, semi estructurados y no estructurados*

Ejemplo:

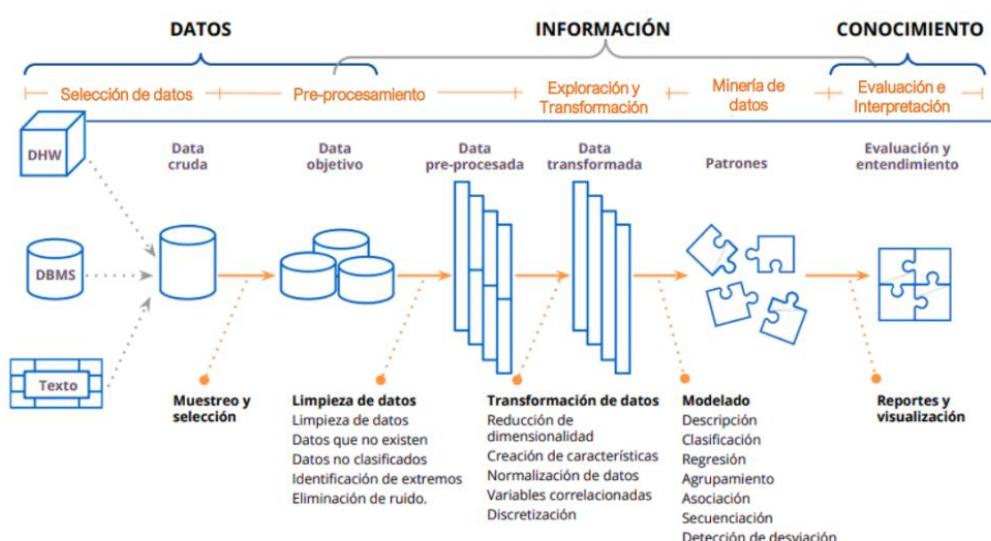


Una empresa de servicios de internet trata de evitar que sus clientes emigren hacia la competencia. Para ello detectan cuales son las actitudes/señales que tiene un cliente antes de cambiarse de empresa.

La empresa cuenta con:

- Información histórica: Facturación, consumos, uso de los datos, plan, costo del plan (datos estructurados)
  - Información del cliente: Atención de soporte recibida (llamadas al call center), navegabilidad en las páginas de otra empresa (ticket de consumo 4G), zonas en las que habitualmente se desplaza (antenas propias y de otras empresas), días en los que consume más datos u otros servicios
- (datos no estructurados.)

### Proceso de Minería de datos



### Propiedades de los modelos de BBDD

#### ACID

- Atomicidad
- Consistencia
- Aislamiento
- Durabilidad

#### BASE

- Basically Available **alta disponibilidad**
- Soft State **puede llegar a ser inconsistente la bdd**
- Eventually Consistent



La consistencia es eventual

*Teorema CAP*

Eric Brewer año 2000

**C - Consistencia (Consistency)**

: Se refiere a la lectura coherente del valor actual del dato desde cualquier instancia, es decir que los datos se encuentran sincronizados y replicados en todos los nodos a la vez.

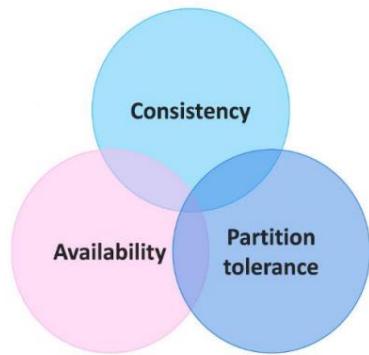
**A - Disponibilidad (Availability)**

: Se refiere a obtener una respuesta válida y rápida para todas las solicitudes, aunque existan nodos inactivos, es decir el acceso a los datos sin interrupciones

**P - Tolerancia a particiones (Partition tolerance):**

Se refiere a la capacidad del sistema para permanecer estable y continuar procesando solicitudes a pesar de ocurrir una partición (interrupción) entre la comunicación de los nodos.

Tolerancia a fallas: se me cae un servidor o replica y el sistema tiene que seguir funcionando



NO todos los modelos de bases de datos cumplen con estas tres características

Cada modelo de base de datos no relacionales tiene características diferentes, y tenemos que elegir el modelo que cumpla con las características de negocio que estemos desarrollando

Modelos de base de datos NoSQL

- Base de datos de pares clave-valor
- Bases de datos de documentos
- Bases de datos de grafos
- Bases de datos Columnas y familias de columnas

*Base de datos de pares clave -valor*

Claves: identificadores asociados con valores

Valores: cadena de caracteres, número, imágenes u objetos binarios

Ejemplos de aplicación

Almacenamiento en caché de datos

Administración de sesiones

Administración de perfiles y preferencias de usuario

Recomendación de producto y servicio

Base de datos relacional			Base de datos Clave-Valor	
ID (Int)	Name (Varchar)	Age(int)	Key	Value
1	Sergio	22	1	Sergio, Andres, 22,19/09/1994
2	Ana	48	2	Ana, 12/08/1969
3	Pablo	49	3	Pablo
4	Juan	12	4	Juan, 12

Conjunto de documentos agrupados en una colección.  
 La clave para lo único que sirve es para encontrar al documento. Puedo tener datos repetidos.  
 Dentro del documento puedo tener lo que quiera.

Acá tengo algo que se llama valor, y este puede ser lo que quiera, por eso son datos semiestructurados.

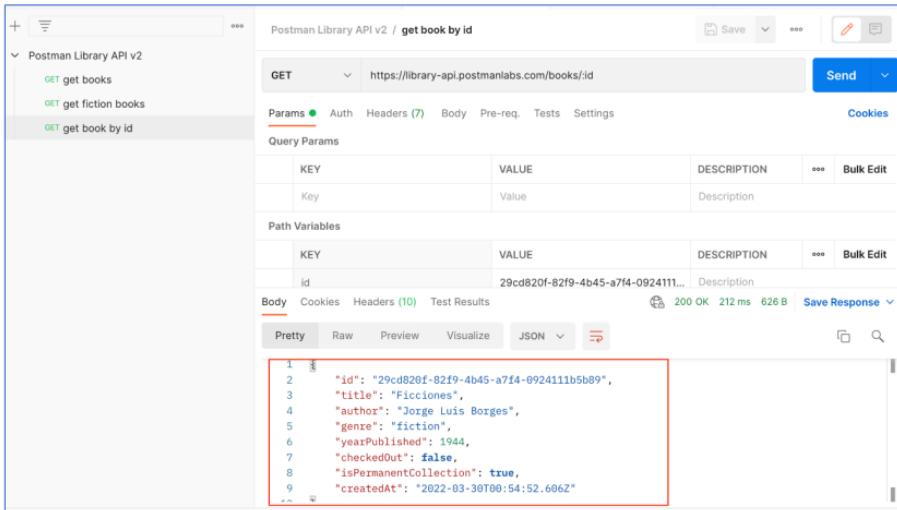
EJEMPLO: recomendaciones de google. Acá no todas las recomendaciones van a tener comentarios, ni la likes, ni compartido

#### *Bases de datos de documentos*

Los documentos son entidades semiestructuradas JSON o XML

#### Ejemplos de aplicación

- Catálogo de productos
- Administración de contenido
- Administración de inventario



The screenshot shows the Postman Library API v2 collection. A GET request is selected with the URL `https://library-api.postmanlabs.com/books/:id`. The 'Pretty' tab of the response body is highlighted, displaying a JSON document:

```

1 {
2   "id": "29cd820f-82f9-4b45-a7f4-092411b5b89",
3   "title": "Ficciones",
4   "author": "Jorge Luis Borges",
5   "genre": "fiction",
6   "yearPublished": 1944,
7   "checkedOut": false,
8   "isPermanentCollection": true,
9   "createdAt": "2022-03-30T00:54:52.686Z"
}

```

Que tipos de documentos almacenos: json o xml (datos semiestructurados)

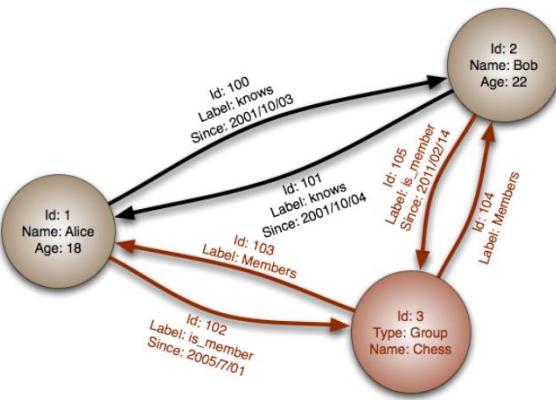
## Bases de datos de grafos

Un nodo es un objeto que tiene un identificador y un conjunto de atributos. (Vértice)

Una relación es un enlace entre dos nodos que contienen atributos sobre esa relación (aristas)

Ejemplos de aplicación

- Organigramas
- Gráficos sociales
- Detección de fraudes
- Motores de recomendaciones



Parecido a puntero de puntero

La relación es dependiendo como necesite moverme dentro del grafo

El id es a modo de etiqueta, porque la relación esta descripta en base al grafo.

Twitter está basada en una red de grafos, por eso puedo ver el encadenamiento de comentarios.

## Bases de datos Columnas y familias de columnas

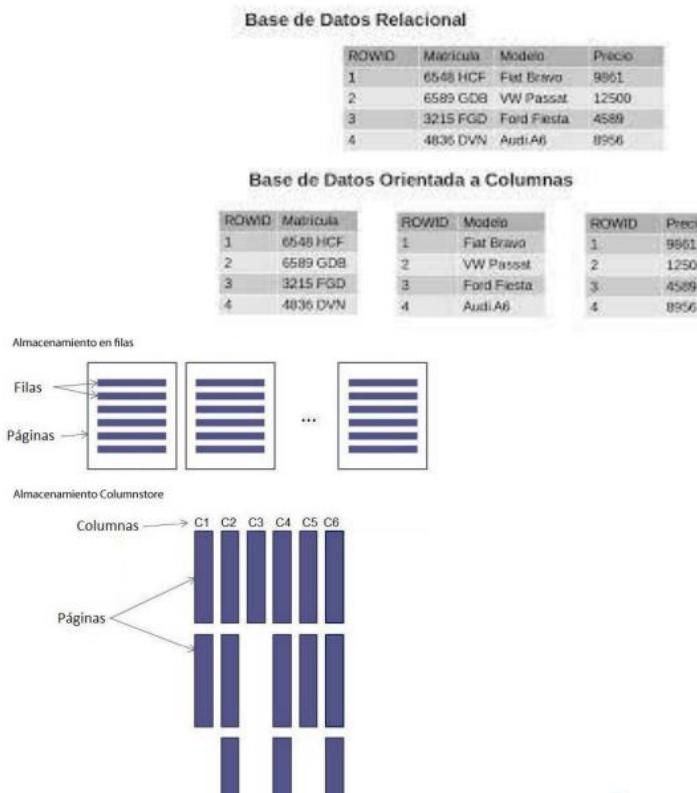
Una columna es una unidad básica de almacenamiento en una base de datos de familia de columnas

Ejemplos

- Recomendaciones
- Personalización
- Datos del sensor
- Telemetría
- Mensajería
- Análisis de redes sociales
- Análisis web

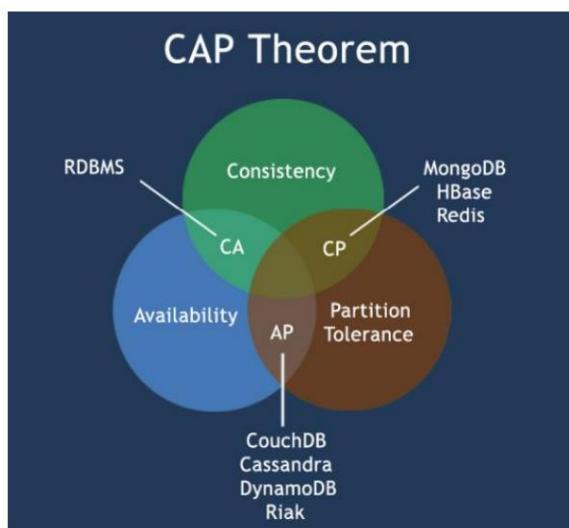


- Supervisión de la actividad
- El tiempo y otros datos de serie temporal



Aplica el efecto contrario a la normalización, des normalizo una relación

Información tabulada.



#### Columnar:

1	Things	A   foo	B   bar	C   baz	
2	Things	C   bam	E   coh	People	A   Emmanuel
3	Languages	A   C	B   Java	C   Ceylon	

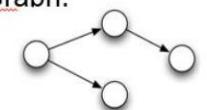
#### Documental:

```
{
  "user": {
    "id": "124",
    "name": "Emmanuel",
    "addresses": [
      {"city": "Paris", "country": "France"},
      {"city": "Atlanta", "country": "USA"}
    ]
  }
}
```

#### Key-value:

key	value
123	Address@23
126	"Booya"

#### Graph:



Notas de clases

[Crear base de datos en mongo](#)

- Instancia el objeto db, pero todavía no creo nada:



- ```
> use Astronomia_2
< switched to db Astronomia_2
➤ Crear una colección para que se instancie la base de datos
> db.createCollection('Estrella');
➤ Cuando me muestra este mensaje significa que ya creo la colección y la base de datos
< { ok: 1 }
```

*Ver desde la línea de comando lo que acabo de crear*

```
> db.getCollectionNames();
< [ 'Estrella' ]
```

*Insertar un documento*

```
db.Estrella.insertOne({
  "Nombre": "-1,47",
  "Magnitud": 1,
  "Distancia_(años_luz)": "8,6",
  "Tamaño": "Enana blanca",
  "Constelacion": "Can Mayor"
})
```

**Acknowledged:** Señal que da el sistema; significa que se pudo llevar a cabo la operación

**InsertId:** me dice el id del registro / documento insertado

```
acknowledged: true,
insertedId: ObjectId('66f35eaa6ab81050857018bd')
```

Los documentos se crean en base a los datos que voy insertando. Es por eso que el mongo db nos va a aparecer “insertar documento” y no “crear documento”.

Por eso mongo es FLEXIBLE. Es por eso que no exige que yo determine los atributos / campos del documento / registro.

Por línea de comando se pueden insertar varios documentos a la vez

*Insertar varios documentos: insertMany*

```
Astronomia > db.Estrella.insertMany([
  {
    "Nombre": "Canopus",
    "Magnitud": "0,72",
    "Distancia_(años_luz)": "309,8",
    "Tamaño": "Supergigante blanco-amarilla",
    "Constelacion": "Carina"
  },
  {
    "Nombre": "Arturo",
    "Magnitud": "0,04",
    "Distancia_(años_luz)": "36,7",
    "Tamaño": "Gigante naranja",
    "Constelacion": "Boyero"
  },
  {
    "Nombre": "Alfa Centauri A",
    "Magnitud": "-0,01",
    "Distancia_(años_luz)": "4,37",
    "Tamaño": "Enana amarilla",
    "Constelacion": "Centauro"
  }
])
```

### *Tipos de datos básicos de mongo*

String

Integer

Enteros

Doublé

Con coma

Booleano

Null

Array

No es el Array que estamos acostumbrados en c; acá en mongo es para poder agregar subconjuntos de datos. Listado de datos dentro de un campo, puedo poner cualquier cosa

**Amenities:** se puso muy de moda la palabra, muy “palermisiana”; es como un agregado de valor

Objeto

Estructura de datos dentro de otra estructura; un json dentro de un json

Date (fecha)

```
/**  
 * Paste one or more documents here  
 */  
{  
  "_id": {  
    "$oid": "66f367aa1815709381b5a4df"  
  },  
  "Nombre": "Facundo",  
  "Magnitud": 1,  
  "Distancia_(años_luz)": "8,6",  
  "Tamaño": "Enana blanca",  
  "Constelacion": "Can Mayor",  
  "Fecha_ISO": {"$date": "2024-09-25"}  
}
```

*Update un solo campo*

Para usarlo primero tengo que decir que quiero actualizar

```
> db.Estrella.updateOne({Nombre:"Canopus"},{$set:{"Fecha Descubrimiento":null}})  
< {  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}
```

**Acknowledged:** el comando lo pudo ejecutar

**InsertId:** dato insertado

**Matchedcount:** cantidad encontrada de campos

**Modifiedcount:** cantidad de campos modificados

**Upsertedcount:**

```
db.collection.updateOne (  
  {query}, --> que documento queremos modificar  
  {$set: {campo:Valor} -->campo y valor  
 } )
```

```
db.Estrella.updateOne({Nombre:"Canopus"},{$set:{"Fecha Descubrimiento":null}})
```



Update un varios campos

```
db.Estrella.updateMany(  
  {},  
  { $set: {"Fecha Descubrimiento":null} }  
)
```

```
> db.Estrella.updateMany(  
  {},  
  { $set: {"Fecha Descubrimiento":null} }  
)  
< {  
    acknowledged: true,  
    insertedId: null,  
    matchedCount: 10,  
    modifiedCount: 9,  
    upsertedCount: 0  
}
```

Borrar un registro

```
db. Estrella.deleteOne( { Nombre: " Canopus" })
```

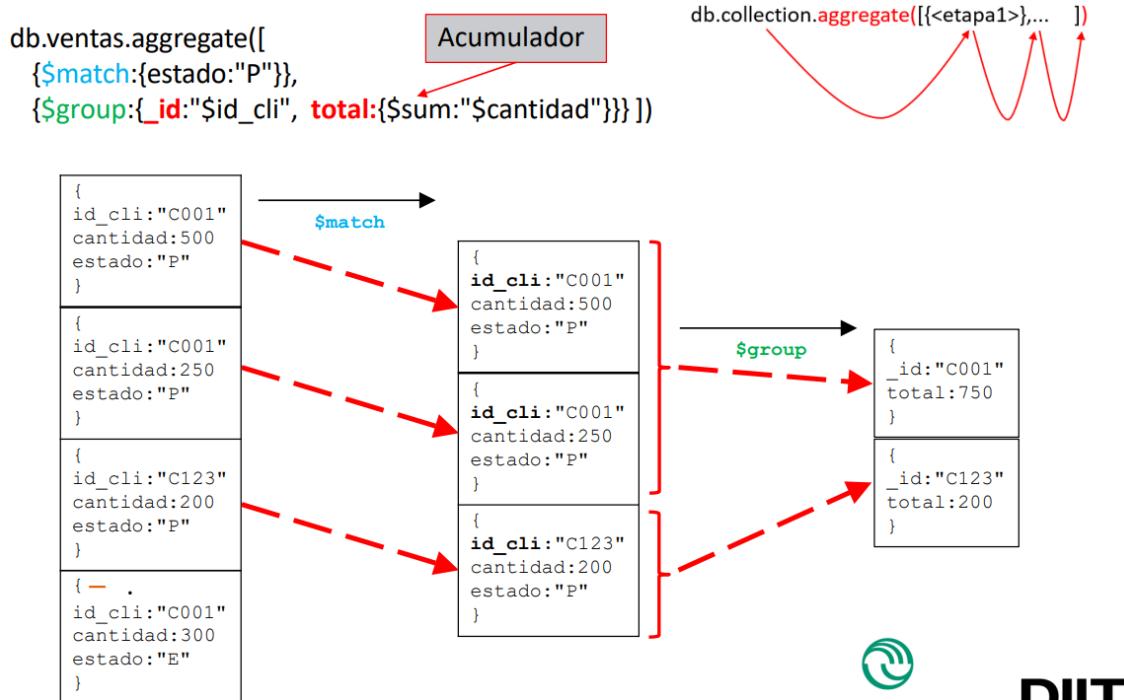
```
> db. Estrella.deleteOne( { Nombre: "Canopus" })  
< {  
    acknowledged: true,  
    deletedCount: 1  
}
```

Borrar varios registros

## MongoDB Compass Aggregation Framework

- ❑ En MongoDB Compass, el Aggregation Framework se refiere a un conjunto de herramientas y operadores que permiten realizar consultas avanzadas y procesar datos de manera eficiente dentro de una base de datos MongoDB.
- ❑ Este framework utiliza una estructura de pipeline (flujo de etapas o canalizaciones) que permite transformar, filtrar, agrupar y analizar datos paso a paso.
- ❑ Se maneja a través de una interfaz gráfica que facilita la creación de las diferentes etapas de agregación.
- ❑ ¿Qué permite ?:

- ✓ Crear y modificar etapas.
- ✓ Visualizar resultados: Ver el resultado de cada etapa en tiempo real, lo que facilita el ajuste y la depuración de consultas.



### *Crear agregaciones con MongoDB Compass*

- ❑ En cada etapa se pueden filtrar, ordenar, agrupar, remodelar y modificar los documentos que pasan por el proceso.



- ✓ \$match etapa - filtra los documentos con los que necesitamos trabajar.
- ✓ \$group etapa - realiza el trabajo de agregación
- ✓ \$sort etapa - ordena los documentos resultantes de la forma que deseemos (ascendente o descendente)
- ❑ Las etapas pueden repetirse.
- ❑ La entrada de la cadena puede ser una sola colección, en la que se pueden fusionar otras más adelante.
- ❑ De este modo, podemos dividir una consulta compleja en etapas más sencillas, en cada una de las cuales completamos una operación diferente sobre los datos.
- ❑ La salida de cada etapa será la entrada del siguiente.

□ Algunas de las etapas más comunes incluyen:

- \$match: Filtra los documentos para que solo pasen aquellos que cumplan con ciertos criterios.
- \$project: Proyecta solo los campos deseados de los documentos, permitiendo renombrar, agregar o eliminar campos.
- \$group: Agrupa los documentos basándose en un campo o expresión y realiza cálculos de agregación, como sumas, promedios o contar documentos en cada grupo.
- \$sort: Ordena los documentos en el resultado según un campo específico.
- \$limit: Limita el número de documentos en el resultado.
- \$skip: Omite un número específico de documentos en el resultado.
- \$lookup: Realiza una operación de unión (join) con otra colección, agregando información de documentos de otra colección al resultado.
- \$unwind: Descompone un campo de matriz en varios documentos, generando un documento separado para cada elemento de la matriz.
- \$replaceRoot: Se usa \$mergeObjects para combinar el documento original (\$\$ROOT) con un subdocumento. Los campos de este pasan a la raíz del documento.
- \$addFields: Agrega nuevos campos calculados al documento.

## \$lookup

- Permite hacer joins.
- Añade un array a cada documento para aquellos elementos que coinciden con la colección enlazada.

### {\$lookup:

```
{
  from: <colección2 a enlazar>,
  localField:<campo de colección origen>,
  foreignField: <campo de colección "from" >,
  as: <array de salida>
}
```

Array de documentos combinados.

Campos por los que hacemos el left outer join.

Permite hacer joins entre dos colecciones. Añade un array a cada documento que contiene los elementos coincidentes de la colección enlazada..



Stage1 \$lookup

```

1 * /**
2  * from: The target collection.
3  * localField: The local join field.
4  * foreignField: The target join field.
5  * as: The name for the results.
6  * pipeline: Optional pipeline to run on the target collection.
7  * let: Optional variables to use in the pipeline.
8 */
9 {
10   from: "datosvendedores",
11   localField: "id_vendedor",
12   foreignField: "Id",
13   as: "vendedor"
14 }
    
```

Output after \$lookup stage (Sample of 10 documents)

```

_id: ObjectId('66d19b677b671292102217f')
id_venta : 1
fecha : "2024-05-16"
cantidad : 1
importe : 1500
cliente : Object
  id_vendedor : 1
  producto : Object
  vendedor : Array (1)
    
```

```

_id: ObjectId('66d19b677b6712922102217f')
id_venta : 2
fecha : "2024-05-15"
cantidad : 2
importe : 3000
cliente : Object
  id_vendedor : 2
  producto : Object
  vendedor : Array (1)
    
```

## \$Unwind

- Descompone un documento en tantos documentos como elementos tenga el Array.
- Opción 1: {\$unwind: }

- Si no es un array, lo trata como un array de 1.
- Si está vacío o no existe, lo ignora.

```
Ejemplo: {"_id": 1, "item": "ABC1", "tallas: ["S","M","L"]}
db.inventory.aggregate([{$unwind: "$tallas"}])
  { "_id" : 1, "item" : "ABC1", "tallas" : "S" }
  { "_id" : 1, "item" : "ABC1", "tallas" : "M" }
  { "_id" : 1, "item" : "ABC1", "tallas" : "L" }
```

- Opción 2:{\$unwind:}

path: <campo>,

includeArrayIndex: <string>,

preserveNullAndEmptyArrays: <boolean> } }

Donde:

- campo: cadena que identifica un array (p.e. "\$Datos")
- includeArrayIndex: nombre del campo que contendrá el índice del valor que desagrega. Si no es un array, tendrá un null.

- `preserveNullAndEmptyArrays`: Por defecto, es falso, y no incluye los valores de la colección en el caso de que no contenga nada. Si vale true, los incluye.

En este ejemplo, separa los documentos de la colección original que contienen un array en documentos individuales para cada vendedor.

```

_id: ObjectId('66fe6cd74c9d897125f40589')
nombre : "Ana García"
dni : 98765432
fecha_nacimiento : 1990-07-12T00:00:00.000+00:00
activo : true
saldo : 2500.50
> direccion : Object
  > documentos : Array (2)
    0: "DNI"
    1: "Licencia"
  fecha_registro : Timestamp({ t: 1693104000, i: 1 })
  > misc : Array (5)

```

**\$unwind:**

```

{ path:"$Array",
  includeArrayIndex:"arrayIndex",
  preserveNullAndEmptyArrays:true
}

```

▼ Stage 1 \$unwind

```

1 ▶ /**
2  * path: Path to the array field.
3  * includeArrayIndex: Optional name for the new array index field.
4  * preserveNullAndEmptyArrays: Optional
5  *   toggle to unwind null and empty value
6  */
7 ▶ {
8   path: "$documentos",
9   includeArrayIndex: "Indice",
10  preserveNullAndEmptyArrays:true
11 }

```

Output after \$unwind stage (Sample of 5 documents)

|                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> dni : 98765432 fecha_nacimiento : 1990-07-12T00:00:00.000+00:00 activo : true saldo : 2500.50 &gt; direccion : Object   documentos : "DNI"   fecha_registro : Timestamp({ t: 1693104000, i: 1 })   &gt; misc : Array (5)     Indice : 0 </pre> | <pre> dni : 98765432 fecha_nacimiento : 1990-07-12T00:00:00.000+00:00 activo : true saldo : 2500.50 &gt; direccion : Object   documentos : "Licencia"   fecha_registro : Timestamp({ t: 1693104000, i: 1 })   &gt; misc : Array (5)     Indice : 1 </pre> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### \$replaceRoot:

Se utiliza para reemplazar el documento raíz actual por el contenido de un subdocumento especificado

```

_id: ObjectId('66fe95ce49e8f44140e129e8')
idPelicula : 1
nombrePelicula : "Inception"
duracion : 148
anio : 2010
> director : Object
  idDirector : 1
  apellido : "Nolan" Object
  .
  .
  .

```

▼ Stage 1 \$replaceRoot

```

1 ▶ {
2   newRoot: {
3     $mergeObjects: [
4       "$$ROOT",
5       "$director"
6     ]
7   }
8 }

```

Mantiene todos los campos originales

Promueve los campos del subdocumento 'director' al nivel raíz

```

1 ▶ /**
2  * specifications: The fields to
3  *   include or exclude.
4  */
5 ▶ {
6   director:0
7 }

```

Output after \$project stage (Sample of 10 documents)

|                                                                                                                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> idPelicula : 1 nombrePelicula : "Inception" duracion : 148 anio : 2010 &gt; actores : Array (3)   idDirector : 1   apellido : "Nolan"   nombre : "Christopher"   nacionalidad : "Británico" </pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### \$addFields

➢ Añade nuevos campos a documentos, manteniendo los que existen.

➢ También permite modificar los campos existentes.

➤ Es equivalente a un \$project que mantenga todos los campos y añada otros nuevos.

```
{ $addFields:
  {
    campo: <expresión>, ...
  }
}
```

Si el campo ya existe (incluyendo \_id), se sobrescribe.

- mes: Añade un campo nuevo que contiene el nombre del mes extraído de la fecha de la venta.

✓ \$let: Define variables locales (meses y mesNúmero).

- meses: Array que contiene los nombres de los meses en español.
- mesNúmero: Convierte el mes de la fecha a un número entero utilizando \$dateToString y \$dateFromString.

✓ in: Utiliza las variables locales para obtener el nombre del mes correspondiente al número del mes.

Esta etapa agrega un campo "mes" al documento, que representa el nombre del mes en el que se realizó la venta.



```

2   * newField: The new field name.
3   * expression: The new field expression.
4   */
5   {
6     mes: {
7       $let: {
8         vars: {
9           meses: [
10             "",
11             "Enero",
12             "Febrero",
13             "Marzo",
14             "Abril",
15             "Mayo",
16             "Junio",
17             "Julio",
18             "Agosto",
19             "Septiembre",
20             "Octubre",
21             "Noviembre",
22             "Diciembre"
23           ],
24         mesNúmero: {
25           $toInt: {
26             $dateToString: {
27               format: "%m",
28               date: {
29                 $dateFromString: {
30                   dateString: "$fecha"
31                 }
32               }
33             }
34           }
35         }
36       }
37     }
38   }
39 }
```

```

_id: 13
fecha: "2024-05-04"
cantidad: 1
importe: 2500
cliente_id: null
cliente_nombre: "Juan"
cliente_apellido: "García López"
cliente_direccion: "Calle 123"
cliente_localidad: "Ramos Mejía"
vendedor_id: null
vendedor_nombre: "Juan"
vendedor_apellidos: "García López"
vendedor_edad: 30
vendedor_sexo: "masculino"
vendedor_pais: "Argentina"
vendedor_país: "Ramos Mejía"
vendedor_país: "Argentina"
producto_id: 101
producto_nombre: "Laptop"
producto_marca: "Dell"
mes: "Mayo"
```

```

_id: 5
fecha: "2024-05-12"
cantidad: 2
importe: 5000
cliente_id: 5
cliente_nombre: "Carlos"
cliente_apellido: "López"
cliente_direccion: "Avenida 789"
cliente_localidad: "Ciudad Evita"
vendedor_id: null
vendedor_nombre: "Juan"
vendedor_apellidos: "Sánchez"
vendedor_edad: 40
vendedor_sexo: "masculino"
vendedor_pais: "Argentina"
vendedor_país: "Gregorio de La"
vendedor_país: "Argentina"
producto_id: 5
producto_nombre: "Disco duro ext"
producto_marca: "Seagate"
mes: "Mayo"
```

- Otro ejemplo: dada una fecha, determinar la cantidad de días transcurridos hasta hoy.



Stage 1 \$addFields

```
1 ▼ { // Convertir la fecha en
2   // formato de texto a una fecha válida
3   fechaActual: "$fecha",
4   fechaConvertida:
5   { $dateFromString:
6     { // Campo fecha en texto
7       dateString: "$fecha",
8       // El formato de tu fecha
9       format: "%Y-%m-%d", },
10    },
11   fechaActualFormatada:
12   {
13     $dateToString:
14     { // Formato: dia/mes/año
15       format: "%d/%m/%Y",
16       // Fecha actual
17       date: "$$NOW", },
18   },
19 }
```

Output after \$addFields stage (Sample of 10 documents)

```
_id: ObjectId('66f4108e6fcf0ceaf4c33c23')
id_venta: 1
fecha: "2024-05-16"
cantidad: 1
importe: 1500
cliente: Object
id_vendedor: 1
producto: Object
fechaActual: "2024-05-16T00:00:00.000+00:00"
fechaConvertida: "2024-05-16T00:00:00.000+00:00"
fechaActualFormatada: "03/10/2024"
```

Stage 2 \$addFields

```
1 ▼ { // Cantidad de días desde una fecha dada y
2   // el día de hoy
3   diasPrestamo:
4   {
5     // Devuelve la diferencia entre dos
6     // fechas.
7     $dateDiff: {
8       // Fecha de dada (convertida)
9       startDate: "$fechaConvertida",
10      // Fecha de fin (fecha actual)
11      endDate: "$$NOW",
12      // Unidad de tiempo (días)
13      unit: "day" },
14   }
15 }
```

Output after \$addFields stage (Sample of 10 documents)

```
fecha: "2024-05-16"
cantidad: 1
importe: 1500
cliente: Object
id_vendedor: 1
producto: Object
fechaActual: "2024-05-16"
fechaConvertida: "2024-05-16T00:00:00.000+00:00"
fechaActualFormatada: "03/10/2024"
diasPrestamo: 140
```

- Añade otro campo calculado llamado "Situacion" que depende de la edad del vendedor.
- Situacion: Define una clasificación de los vendedores basada en su edad.
  - ✓ \$cond: Operador condicional que evalúa si la edad del vendedor es menor, igual o mayor a 30 años.
    - Si la edad es menor de 30, la situación es "Menor de 30 años".
    - Si la edad es exactamente 30, la situación es "Con 30 años".
  - Si la edad es mayor de 30, la situación es "Mayor de 30 años".
- Agrega una descripción de la situación del vendedor en función de su edad.

Stage 5 \$addFields

```
1 ▼ {
2   $cond: {
3     if: {
4       $lt: ["$vendedor_edad", 30]
5     },
6     then: "Menor de 30 años",
7     else: {
8       $cond: {
9         if: {
10           $eq: ["$vendedor_edad", 30]
11         },
12         then: "Con 30 años",
13         else: "Mayor de 30 años"
14       }
15     }
16   }
17 }
18 }
```

Output after \$addFields stage (Sample of 10 documents)

```
cliente_direccion: "Avenida 789"
cliente_localidad: "Ciudad Evita"
vendedor_id: null
vendedor_nombre: "Juan"
vendedor_apellidos: "Sánchez"
vendedor_edad: 40
vendedor_sexo: "masculino"
vendedor_ciudad: "Gregorio de Laferrere"
vendedor_pais: "Argentina"
producto_id: 5
producto_nombre: "Disco duro externo"
producto_marca: "Seagate"
mes: "Mayo"
Situacion: "Mayor de 30 años"
```

```
cliente_direccion: "Calle 123"
cliente_localidad: "Ramos Mejía"
vendedor_id: null
vendedor_nombre: "Juan"
vendedor_apellidos: "García López"
vendedor_edad: 30
vendedor_sexo: "masculino"
vendedor_ciudad: "Ramos Mejía"
vendedor_pais: "Argentina"
producto_id: 101
producto_nombre: "Laptop"
producto_marca: "Dell"
mes: "Mayo"
Situacion: "Con 30 años"
```

\$project

- Se utiliza para incluir o excluir campos en el resultado final.
    - ✓ \_id: 0: Excluye el campo \_id del resultado.
    - ✓ fecha: 0: Se Incluye el campo fecha del resultado.
    - ✓ Incluye todos los campos necesarios para el resultado final, renombrando algunos si es necesario. Por ejemplo: "id\_venta": "\$id"

```
1
2 ▾ { cliente: 0,
3       producto: 0,
4       VentasxVendedor: 0,
5
6
7
8   }
```

Output after \$project stage (Sample of 10 documents)

```
importe : 1500
id_vendedor : 1
Subindice : 0
id : 101
nombre : "Juan"
apellido : "García López"
direccion : "Calle 123"
localidad : "Ramos Mejía"
marca : "Dell"
Id : 1
```

---

```
▼ Stage 5 $project
```

Output after \$project stage (Sample of 10 documents)

```
_id:0,
_id_venta:1,
fecha:1,
"Id_Vendedor": "$id",
"nombre_Vendedor": "$nombre",
"apellido_Vendedor": "$apellido",
"direccion_Vendedor": "$direccion",
"localidad_Vendedor": "$localidad"
```

\$match

- La operación \$match filtra los documentos que pasan a la siguiente etapa de la tubería según una condición.
    - ✓ vendedor\_ciudad: "Ramos Mejía": Filtra los documentos para que solo aquellos donde el vendedor esté en la ciudad de "Ramos Mejía" sean incluidos en el resultado final.
  - Esta etapa reduce los resultados a solo aquellos vendedores que residen en "Ramos Mejía".

```
Stage 7 $match  [ ] ...  
1 * /**  
2 * query: The query in MQL.  
3 */  
4 {  
5   vendedor_ciudad: "Ramos Mejía"  
6 }  
  
Output after $match stage (Sample of 7 documents)  


vendedor_sexo : "masculino"	vendedor_sexo : "masculino"
vendedor_ciudad : "Ramos Mejía"	vendedor_ciudad : "Ramos Mejía"
vendedor_país : "Argentina"	vendedor_país : "Argentina"
producto_id : 101	producto_id : 101
producto_nombre : "Laptop"	producto_nombre : "Laptop"
producto_marca : "Dell"	producto_marca : "Dell"
mes : "Mayo"	mes : "Abril"
Situacion : "Con 30 años"	Situacion : "Con 30 años"
id_venta : 13	id_venta : 37


```

## \$sort

- Ordena los documentos
- {\$sort: { <campo1>: <ordenación>, <campo2>: <ordenación> ... } }

## \$sortByCount

- Agrupa los documentos y los cuenta por grupos, devolviendo en el resultado el \_id correspondiente a la agrupación y un valor con el número de documentos.
- {\$sortByCount: <expression> }

## \$count

- Devuelve el número de documentos que entran en la etapa.
- { \$count: <string> }

## \$limit

- Limita el número de documentos que pasan a la siguiente etapa, sin afectar al contenido.
- { \$limit: <entero positivo> }

## \$skip

- Ignora los primeros n documentos.

{ \$skip: <entero positivo> }

## \$out

- Escribe la salida a una colección. Debe ser la última etapa del pipeline.
- { \$out: "<colección de salida>" }

## \$group

➤ Agrupa los documentos de acuerdo a alguna expresión.

➤ El documento de salida contiene un \_id que contiene el identificador de los distintos grupos. Puede contener campos calculados mediante algún acumulador.

### Sintaxis

{ \$group: { \_id: <expresión>,

<campo1>: { <acumulador1> : <expresión1> },

... } }

✓ El campo \_id es obligatorio: Si se pone null, agrupa todos los documentos

(\_id : null)

✓ Acumuladores: \$sum, \$avg, \$last, \$max, \$min, \$push, \$addToSet...

- ✓ \$first: Operador que selecciona el primer valor encontrado en el grupo para el campo especificado.

- Agrupa los documentos de acuerdo a alguna expresión.
- El documento de salida contiene un \_id que contiene el identificador de los distintos grupos. Puede contener campos calculados mediante algún acumulador.

#### Sintaxis

```
{ $group: { _id: <expresión>,
<campo1>: { <acumulador1> : <expresión1> },
... } }
```

- ✓ El campo \_id es obligatorio: Si se pone null, agrupa todos los documentos (\_id : null)
- ✓ Acumuladores: \$sum, \$avg, \$last, \$max, \$min, \$push, \$addToSet...
- ✓ \$first: Operador que selecciona el primer valor encontrado en el grupo para el campo especificado.

- Ejemplo: Total de ventas por localidad de clientes. Este ejemplo muestra cómo agrupar las ventas por la localidad de los clientes y sumar el importe total para cada localidad.



The screenshot shows the MongoDB Compass interface. On the left, under 'Stage 1 \$group', there is a code editor with the following aggregation stage:

```
1  {
2    // Agrupa por localidad de cliente
3    _id: "$cliente.localidad",
4    // Suma el importe de las ventas
5    totalVentas: { $sum: "$importe" },
6    // Cuenta el número de ventas por localidad,
7    cantidadVentas: { $sum: 1 }
8  }
```

To the right, under 'Output after \$group stage (Sample of 6 documents)', there is a preview of the resulting documents:

```
_id: "San Justo"
totalVentas : 26400
cantidadVentas : 10
```

Below the interface, a terminal window shows the MongoDB shell command:

```
Grupo00> db.datosventas.aggregate([
...   {
...     $group: {
...       _id: "$cliente.localidad",
...       totalVentas: { $sum: "$importe" },
...       cantidadVentas: { $sum: 1 }
...     }
...   }
... ])
```

- Ejemplo: Ventas por producto y marcas ordenado por producto y marca



```
Grup00> db.datosventas.aggregate([
...   {
...     $group: {
...       _id: { idProducto: "$producto.id", marca: "$producto.marca" },
...       totalIngresos: { $sum: "$importe" },
...       ventasTotales: { $sum: 1 }
...     }
...   },
...   {
...     $sort: { "_id.idProducto": 1, "_id.marca": 1 }
...   }
... ])
[ {
  _id: { idProducto: 4, marca: 'HP' },
  totalIngresos: 1200,
  ventasTotales: 1
},
{
  _id: { idProducto: 5, marca: 'Seagate' },
  totalIngresos: 5000,
  ventasTotales: 1
},
```

```
Stage 1 [$group]
1 { _id: { idProducto: "$producto.id", marca: "$producto.marca" }, totalIngresos: { $sum: "$importe" }, ventasTotales: { $sum: 1 } }
2
Stage 2 [$sort]
1 /**
2 * Provide any number of field/order pair
3 */
4 {
5   "_id.idProducto": 1,
6   "_id.marca": 1
7 }
```

- Ejemplo: Ventas por producto y marcas ordenado por producto y marca

```
Grup00> db.datosventas.aggregate([
...   {
...     $group: {
...       _id: { idProducto: "$producto.id", marca: "$producto.marca" },
...       totalIngresos: { $sum: "$importe" },
...       ventasTotales: { $sum: 1 }
...     }
...   },
...   {
...     $sort: { "_id.idProducto": 1, "_id.marca": 1 }
...   }
... ])
[ {
  _id: { idProducto: 4, marca: 'HP' },
  totalIngresos: 1200,
  ventasTotales: 1
},
{
  _id: { idProducto: 5, marca: 'Seagate' },
  totalIngresos: 5000,
  ventasTotales: 1
},
```

```
Stage 1 [$group]
1 { _id: { idProducto: "$producto.id", marca: "$producto.marca" }, totalIngresos: { $sum: "$importe" }, ventasTotales: { $sum: 1 } }
2
Stage 2 [$sort]
1 /**
2 * Provide any number of field/order pair
3 */
4 {
5   "_id.idProducto": 1,
6   "_id.marca": 1
7 }
```

Se agrupan todos los documentos por el campo "id\_venta" y se selecciona los primeros valores encontrados para cada campo especificado, creando un solo documento por cada venta con todos los detalles de la venta y del vendedor..



```
Grupo00> db.datosventas.aggregate([
...   {
...     $group: {
...       _id: "$producto.id",
...       productoNombre: { $first: "$producto.nombre" },
...       minImporte: { $min: "$importe" },
...       maxImporte: { $max: "$importe" }
...     }
...   }
... ])
[ {
  _id: 103,
  productoNombre: 'Teclado',
  minImporte: 2700,
  maxImporte: 4500
},
```

```
Grupo00> db.datosventas.aggregate([
...   {
...     $group: {
...       _id: "$producto.id",
...       productoNombre: { $first: "$producto.nombre" },
...       importes: { $push: "$importe" }
...     }
...   }
... ])
[ { _id: 104, productoNombre: 'Mouse', importes: [ 2200, 2200, 2200 ] },
{ _id: 102,
  productoNombre: 'Monitor',
  importes: [ 3000, 4000, 4000, 4000 ] }]
```

\$push: Agrupa todos los valores del campo importe en un array para cada grupo.

\$addToSet: permite agrupar valores únicos en un array cuando utilizas la etapa \$group.

Esta opción asegura que los elementos en el array resultante sean únicos, eliminando duplicados automáticamente.

➤ \$last: se utiliza en el contexto de la etapa \$group para seleccionar el último valor de un grupo de documentos, basado en el orden en que esos documentos aparecen en la entrada de datos.

➤ Este operador devuelve el último valor de un campo dentro de cada grupo que se está creando.

## □ Ejemplo:

The screenshot shows a MongoDB pipeline being run against a sample of 10 documents. The stages are as follows:

- Input:** A sample of 10 documents with fields like `_id`, `vendedor`, `cliente`, and `producto`.
- \$group:** Groups documents by `vendedor`, `cliente`, and `producto`. The output stage shows grouped documents with fields `vendedorNombre`, `vendedorApellido`, `clienteNombre`, `clienteApellido`, `productoNombre`, and `productoMarca`.
- \$unwind:** Unwinds the grouped documents. The output stage shows individual documents with fields `_id`, `vendedorNombre`, `vendedorApellido`, `clienteNombre`, `clienteApellido`, `productoNombre`, and `productoMarca`.
- \$addFields:** Adds new fields `direccion`, `localidad`, `marca`, and `clienteNombre` to each document. The output stage shows the full document structure including these new fields.
- \$project:** Projects specific fields: `_id`, `fecha`, `cantidad`, `importe`, `vendedor`, `cliente`, and `producto`. The output stage shows the final projected document structure.



□ Una vez creada la canalización se puede:

Save  
Save as  
Create view

Save Pipeline

Name: Buscar\_por\_ciudad

Cancel Save

+ CREATE NEW

Crear una nueva

Puedo exportar el código a distintos lenguajes

DDBBA – Unidad 4 – parte 2 – V2.0

EXPORT TO LANGUAGE

My Pipeline

```
1 {  
2   $match:  
3     /*  
4      * query: The query in MQL.  
5      */  
6     {  
7       ciudad: "Ramos Mejía",  
8     },  
9   }  
10 }
```

Exported Pipeline

Java

```
< Document("{$match",  
"ciudad": "Ramos Mejia"})>
```

Node.js

C#

Python

Ruby

Go

Rust

PHP

Include Import Statements  
Include Driver Syntax

PREVIEW | PREVIEW Excluyo que se muestren los datos en la consulta. Esto es importante para grandes cantidades de filas

Stage 2 (\$project

```
1 • {  
2   _id: 0,  
3   nombre: 1,  
4   apellidos: 1,  
5   ciudad: 1,  
6   edad: 1,  
7   Situación: {  
8     $cond: {  
9       if: {  
10         $gte: ["$edad", 30],  
11       },  
12       then: "Mayor o igual a 30",  
13       else: "Menor a 30",  
14     },  
15   },  
16 }
```

{ STAGES </> TEXT Nos muestra la sintaxis en modo Etapa o texto

```
1 • {  
2   _id: 0,  
3   nombre: 1,  
4   apellidos: 1,  
5   ciudad: 1,  
6   edad: 1,  
7   Situación: {  
8     $cond: {  
9       if: {  
10         $gte: ["$edad", 30],  
11       },  
12       then: "Mayor o igual a 30",  
13       else: "Menor a 30",  
14     },  
15   },  
16 }
```

DDBBA – Unidad 4 – parte 2 – V2.0

29

```
1 • [  
2   {  
3     $match:  
4       /*  
5          * query: The query in MQL.  
6          */  
7       {  
8         ciudad: "Ramos Mejía",  
9       },  
10    },  
11    {  
12      $project:  
13        _id: 0,  
14        nombre: 1,  
15        apellidos: 1,  
16        ciudad: 1,  
17        edad: 1,  
18        Situación: {  
19          $cond: {  
20            if: {  
21              $gte: ["$edad", 30],  
22            },  
23            then: "Mayor o igual a 30",  
24            else: "Menor a 30",  
25          },  
26        },  
27      }  
28    }  
29  ]
```



- Una vez creada la agregación se puede guardar como una vista

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with collections like 'ejemplos', 'datosvendedores', 'datosventas', and 'personas'. A context menu is open over a document in the 'vDetalleVentas' collection, with options 'Save', 'Save as', and 'Create view'. A modal window titled 'Create a View' has the name 'vDetalleVentas' entered. Below these, the document details are shown in two sections:

```
_id: ObjectId('664f467afab13c0580262a97')
id_venta: 1
fecha: "2024-05-16"
cantidad: 1
importe: 500
cliente_id: 1
cliente_nombre: "Juan"
cliente_apellido: "García López"
producto_id: 101
producto_nombre: "Laptop"
producto_marca: "Dell"
vendedor_Id: 1
vendedor_nombre: "Juan"
vendedor_apellidos: "García López"
vendedor_ciudad: "Ramos Mejía"
vendedor_pais: "Argentina"

_id: ObjectId('664f467afab13c0580262aa2')
id_venta: 12
fecha: "2024-05-05"
cantidad: 3
importe: 6000
cliente_id: 12
cliente_nombre: "Raúl"
```

- Se puede crear una vista usando una canalización.

```
Grupo00> db.createView(
...   "ventasxvendedores",
...   "datosventas",
...   [ { $lookup: {
...     from: "datosvendedores",
...     localField: "id_vendedor",
...     foreignField: "Id",
...     as: "VentasxVendedor"
...   } },
...   { $unwind: {
...     path: "$VentasxVendedor",
...     includeArrayIndex: "Subindice",
...     preserveNullAndEmptyArrays: true
...   } },
...   { $replaceRoot:
...     { newRoot: {
...       $mergeObjects: [
...         "$$ROOT",
...         "$cliente",
...         "$producto",
...         "$VentasxVendedor" ] } } },
...   { $project:
...     {
...       cliente: 0,
...       producto: 0,
...       VentasxVendedor: 0
...     } },
...   { $sort:
...     { id_venta: 1 } }
... ]
{ ok: 1 }
```

```
_id: ObjectId('66ffd1d176ae41c94bf4b0b1')
id_venta: 1
fecha: "2024-05-16"
cantidad: 1
importe: 1500
id_vendedor: 1
Subindice: 0
id: 101
nombre: "Juan"
apellido: "García López"
direccion: "Calle 123"
localidad: "Ramos Mejía"
marca: "Dell"
Id: 1
apellidos: "García López"
edad: 30
sexo: "masculino"
ciudad: "Ramos Mejía"
pais: "Argentina"
```

## UNIDAD N°5: SEGURIDAD DE DATOS

Seguridad vs Protección

Puedo aplicar muchos métodos de protección, y seguir teniendo inseguridad.

Es más correcto hablar de protección que de seguridad. La seguridad es un concepto nada más.

*Seguridad:*

Ausencia de un riesgo. Aplicando esta definición a al tema correspondiente, se hace referencia al riesgo de accesos no autorizados, de manipulación de información, manipulación de las configuraciones, entre otros

*Protección:*

Mecanismos empleados para proteger datos de ser alterados o borrados, de acceso no autorizado, etc.

Protección: un policía por cuadra. Aumenta la seguridad... pero no nos previene de un ataque ransomware!

*Principio del mínimo privilegio (POLP)*

A los usuarios y las aplicaciones se les debe otorgar acceso solo a los datos y operaciones que requieren para realizar su trabajo.



Otorgarle a aplicaciones y personas los permisos mínimos para su trabajo.

“darle muchos permisos a un programa choto es como darle la llave del edificio al repartidor para que entre la pizza”

*Reducción del área expuesta*

implica detener o deshabilitar componentes que no se utilizan. Reduce los puntos de acceso para potenciales ataques. Los servicios deben ejecutarse con privilegios mínimos.



Todo aquello que no se usa hay que suprimirlo para minimizar el riesgo (dejar ventanas sin rejas)

¿Qué medidas de protección podemos implementar en un DBMS?

- Restricciones y permisos.
- Respaldos.
- Encriptación en tránsito y at rest.
- Replicación.
- Buenas prácticas de programación (p/e prevenir inyección SQL).

No termina ahí... debemos securizar el sistema operativo, el software del DBMS, firewall, el hardware, etc, etc.

Permisos, usuarios y roles

*Entidades de seguridad (principals)*

Individuos, grupos y procesos que tienen acceso (pueden solicitar recursos) al sistema. Pueden definirse a nivel de sistema operativo, de servidor SQL o de base de datos.

*El usuario sa (sysadmin)*

Entidad de seguridad a nivel servidor SQL.

Puede deshabilitarse

**Administrador; usuario todo poderoso; super usuario**

*Elementos protegibles (securables)*

Servidor, base de datos y objetos incluidos en ella. Cada uno de estos posee un conjunto de permisos que pueden configurarse para reducir el área expuesta.

**Elementos protegibles: doy permisos para que se usen ciertas cosas: permisos al servidor; con estos permisos puedo reducir el área expuesta**

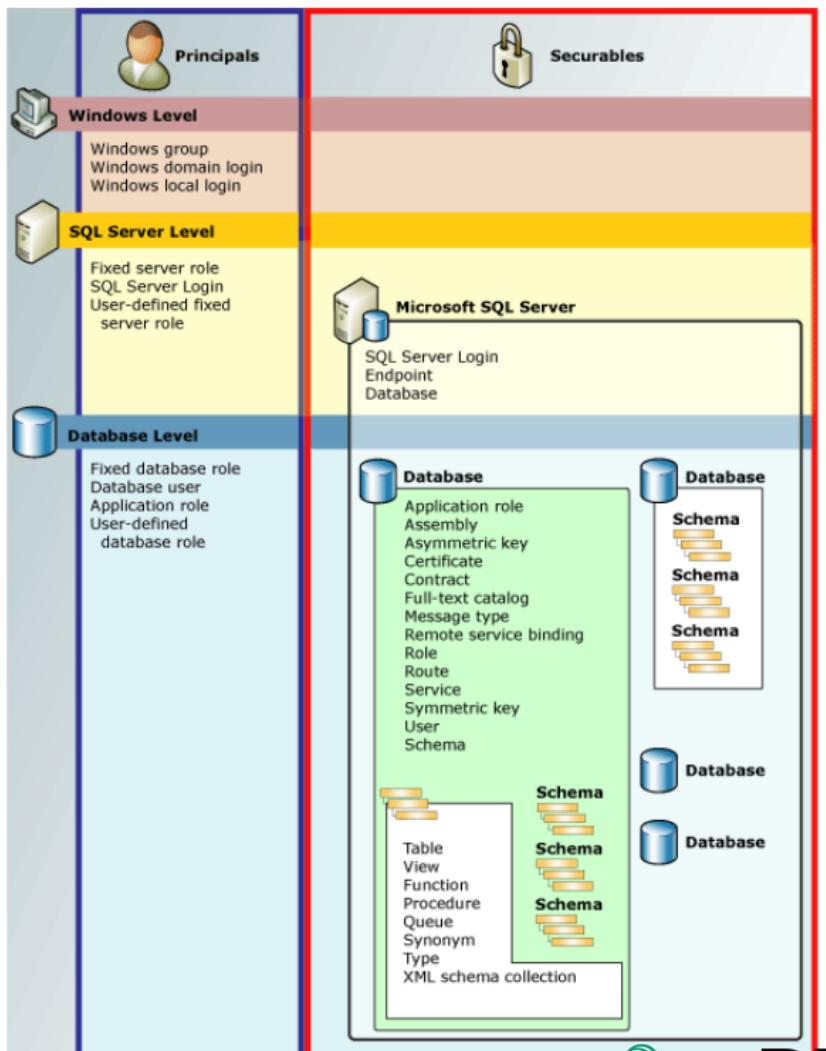
*Directivas de contraseñas*

- Sensibles a mayúsculas/minúsculas.
- Complejidad: largo mínimo, tres de cuatro: mayúsculas, minúsculas, números, símbolos no alfanuméricos (CHECK\_POLICY).
  - Si se apoya en el sistema operativo, aplican las del mismo.
  - Vencimiento (CHECK\_EXPIRATION).
  - Se puede forzar el cambio en siguiente LOGIN.
  - Si usamos ODBC cuidado con los caracteres empleados.

**Contraseñas de acceso al motor; los usuarios que usan nuestro sistema no tienen que tener permisos a las bases de datos.**



## Principals & Securables



```
CREATE LOGIN <login_name>
```

```
WITH PASSWORD = '<StrongPasswordHere>;'
```

```
CREATE LOGIN <login_name> WITH PASSWORD =
```

```
'<StrongPasswordHere>'
```

```
MUST_CHANGE, CHECK_EXPIRATION = ON;
```

```
CREATE LOGIN [<domainName>\<login_name>] FROM WINDOWS;
```

```
CREATE LOGIN [MyUser]
```

```
WITH PASSWORD = 'MyPassword', DEFAULT_DATABASE = MyDatabase,  
CHECK_POLICY = OFF, CHECK_EXPIRATION = OFF ;
```

Un LOGIN permite acceso al servidor, pero por sí mismo no otorga permisos para ninguna DB

Lo login se almacenan en master; los usuarios que son locales en la base de datos se almacenan en la base de datos, pero ¿qué pasa si hago un backup y lo resto en otro servidor? Por eso es que puedo tener usuarios en la base de datos

Hacer copia de master al momento de hacer el backup

*Usuario de base de datos*

- El LOGIN permite acceder al servidor pero por defecto no otorga permisos de acceso a ninguna DB.
- Debemos crear un USUARIO a nivel DB y darle permisos.
- El nombre de USUARIO puede ser distinto al nombre de LOGIN.

Pero solo podemos crear UN usuario para un login.

- El usuario dbo o propietario es una cuenta de usuario con permisos implícitos para realizar todas las actividades en la DB.

```
USE [BasesDatosAplicada]
```

```
GO
```

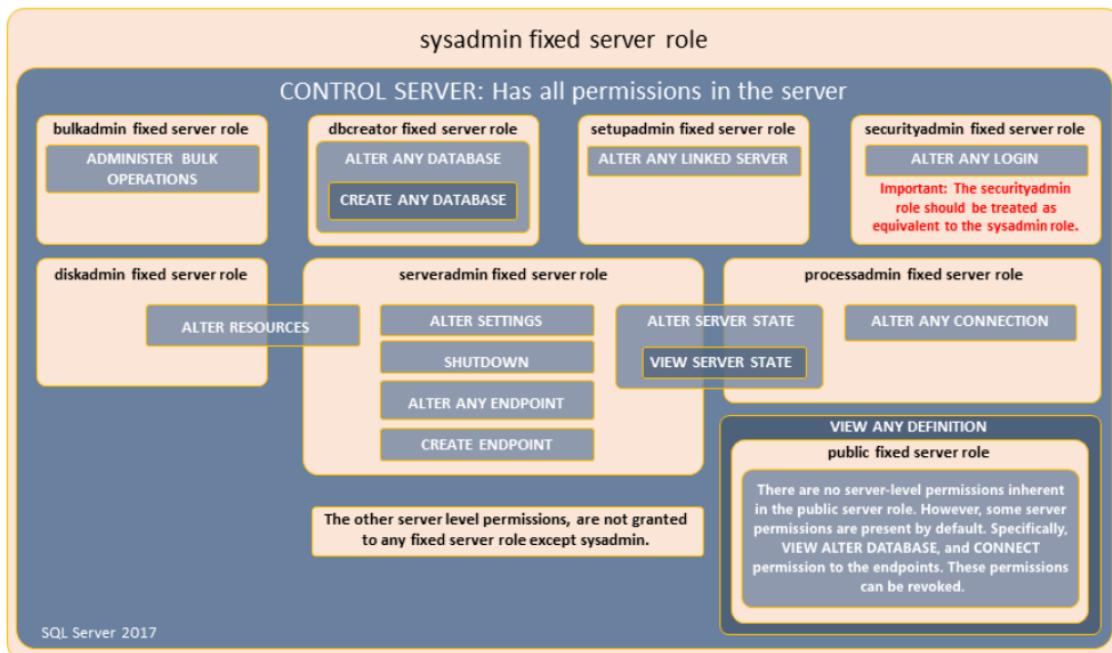
```
CREATE USER [pruebaUser] FOR LOGIN [testuser]
```

*Roles fijos de servidor*

- Son entidades de seguridad que agrupan otras entidades de seguridad. Se aplican a todo el servidor en lo que respecta a su ámbito de permisos.
- No se pueden modificar (en la última versión se pueden crear roles definidos por el usuario).



- Se heredan a las DB si el usuario tiene permisos de conexión a la DB.



- Se puede agregar un LOGIN a un rol para darle permisos.

**ALTER SERVER ROLE Production ADD MEMBER**

**[servidorFulano\usuarioMengano] ;**

**ALTER SERVER ROLE diskadmin ADD MEMBER Ted ;**

**GO**

**ALTER SERVER ROLE Production DROP MEMBER Ted ;**

**GO**

#### Roles de nivel base de datos

- Es mejor práctica otorgar permisos a roles en lugar de a usuarios.
- Todos los miembros del rol heredan los permisos.
- Los roles se pueden anidar (cuidado).
- Los usuarios se asignan a los roles con `ADD MEMBER` y `DROP MEMBER`.
- Puede conceder permisos en el nivel de esquema. Los usuarios heredan los permisos en todos los objetos NUEVOS creados en el esquema.

#### Propietarios

- Los propietarios de los objetos disponen de permisos irrevocables para administrarlos. No se pueden eliminar usuarios si existen objetos que les pertenezcan. No se pueden quitar los privilegios del propietario.



- Los esquemas pueden pertenecer a cualquier entidad de seguridad. Una entidad puede poseer varios esquemas.
- La propiedad de un objeto se puede cambiar con ALTER AUTHORIZATION

El propietario siempre tiene permiso sobre el objeto

## DCL Data Control Language:

### GRANT

- Concede permiso. Puede además dar permiso para conceder permisos (WITH GRANT OPTION).

Le doy un permiso

### REVOKE

- Revoca un permiso. Un permiso revocado se puede heredar de OTRO GRUPO o ROL.

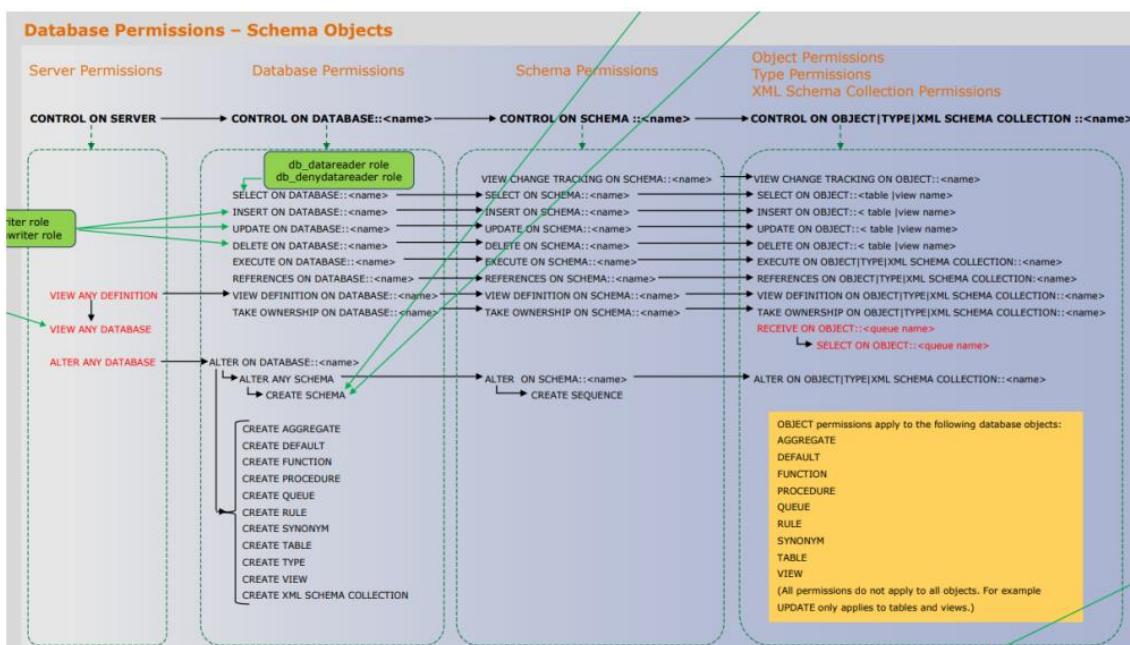
Quito el permiso

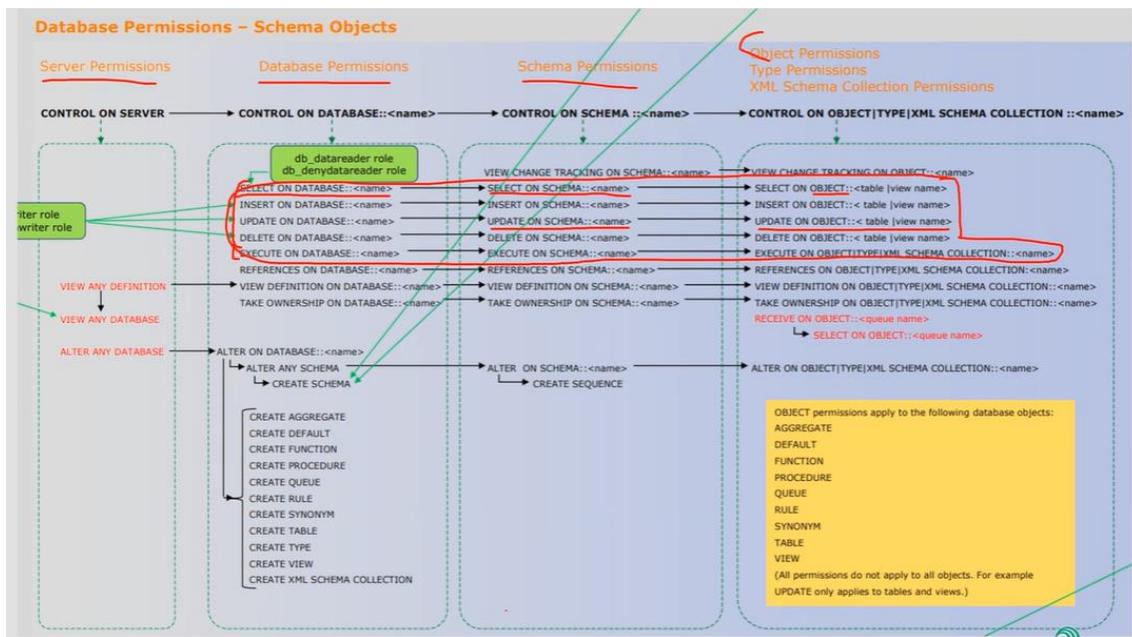
### DENY

- Revoca un permiso de manera que no pueda ser heredado.

Le niego a un usuario hacer algo

¿Cómo anulo un DENY? Con un revoke le puedo quitar esta revocación de permiso





## ❑ Jerarquía de permisos

- Si se concede el permiso SELECT en una DB, incluirá todos los esquemas.
- Si se concede el permiso SELECT en un esquema, incluirá todas las tablas y vistas del esquema.
  - Si los objetos que requieren mismos permisos se encuentran en el mismo esquema se simplifica drásticamente.
- El permiso CONTROL en un objeto normalmente concede todos los otros permisos del objeto.  
**Este no va con el concepto de mínimo de permisos**

• Supongamos la tabla Región en el esquema Clientes de la base de datos Ventas. El usuario Aza obtendría permiso SELECT en la tabla Región mediante cualquiera de estas instrucciones:

**GRANT SELECT ON OBJECT::Region TO Aza;**

**GRANT CONTROL ON OBJECT::Region TO Aza;**

**GRANT SELECT ON SCHEMA::Customers TO Aza;**

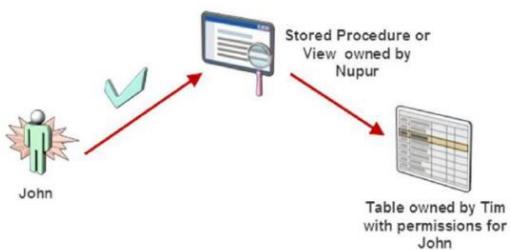
**GRANT CONTROL ON SCHEMA::Customers TO Aza;**

**GRANT SELECT ON DATABASE::SalesDB TO Aza;**

**GRANT CONTROL ON DATABASE::SalesDB TO Aza;**

## ❑ Permisos mediante código basado en procedimiento

- Se puede evitar que los usuarios interactúen directamente con los objetos de la DB otorgando permiso solo a SP o funciones y denegando permisos a objetos subyacentes.
- Encadenamiento de propiedad.



### ❑ ¿Qué es una transacción?

- La unidad de trabajo más pequeña que se ejecuta en la DB.
- Cumplen con las propiedades ACID.

Es lo que el motor va a tomar como usa sola operación: varias operaciones que se ejecuta todo o nada, lo que ocurre dentro de una transacción se considera como una sola cosa

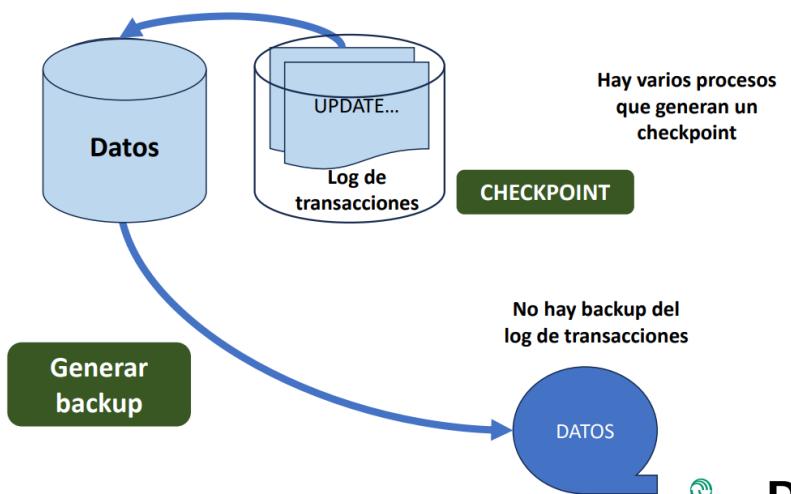
Aunque todas las DB admiten el manejo implícito o explícito de las transacciones, hay distintos **modelos de recuperación**.

El modelo **SIMPLE** no contempla respaldo del log de transacciones.

El modelo **FULL** y **BULK-LOGGED** admiten y requieren respaldo del log de transacciones.

#### Modelo de recuperación SIMPLE

No manejamos backup del log de transacciones, no lo contempla



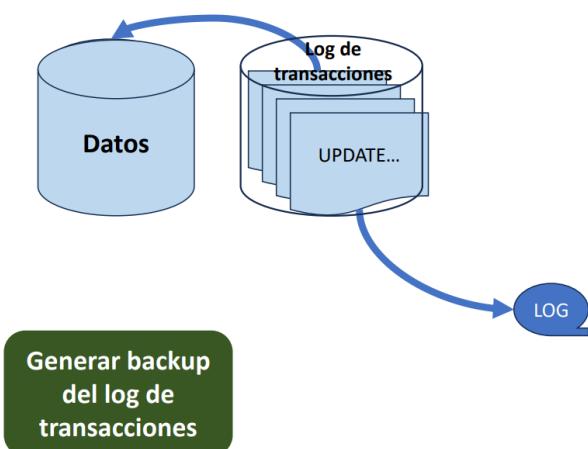
Checkpoint: los cambios que están en el log de transacciones se van a impactar los cambios en la base de datos cuando se de el Checkpoint; hasta que no se ejecute el Checkpoint

Log: su traducción es bitácora

#### Modelo de recuperación FULL

Contempla el respaldo del backup del log de transacciones

Viene por defecto ete cuando instalo



Acá se pueden hacer backup sobre el log de transacciones: estas transacciones del log van a impactar en la base de datos

Este permite, en un esquema de backup bien hecho, restaurar las transacciones hasta un punto específico (ejemplo: hasta un número específico de transacción, o hasta una hora determinada del sistema)

Log de transacciones: info de transacciones desde el último backup que se hizo

## Registro de Transacciones (RT)

❑ *¿Qué es el registro de transacciones?*

- Es donde se registran todas las transacciones y las modificaciones que cada transacción realiza en la DB.
- Es un componente esencial de la base de datos.
- Si hay un error del sistema, ese registro será necesario para devolver la base de datos a un estado coherente.

❑ *¿Cómo se implementa?*

- En un archivo o grupo de archivos separado de la DB.
- Al crear la DB se establece su ubicación y configuración de crecimiento.

❑ *¿Qué almacena un registro de transacciones SQL Server?*

- Almacena cada transacción hecha en una base de datos SQL Server, excepto algunas que son mínimamente registradas como BULK IMPORT o SELECT INTO.
- Internamente está dividido en partes más pequeñas llamadas Archivos de Registros Virtuales (Virtual Log Files, VLFs).
- Cuando un VLF se llena, el registro continúa en el siguiente registro de transacciones disponible.

❑ *Archivos de registro virtuales (VLF).*

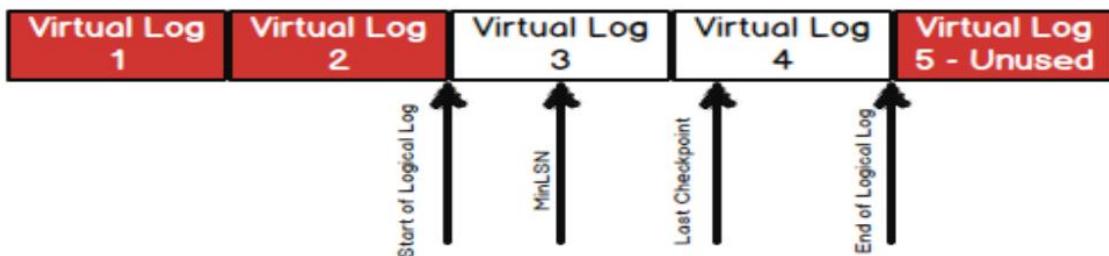
- El DBMS SQL Server divide cada archivo de registro físico internamente en varios archivos de registro virtuales (VLF).
  - Los VLF no tienen un tamaño fijo y no hay una cantidad fija de archivos de registro virtuales para un archivo de registro físico.
  - El motor de base de datos elige dinámicamente el tamaño de los archivos de registro virtuales mientras crea o extiende archivos de registro
  - La vista sys.dm\_db\_log\_info, devuelve información del archivo de registro virtual (VLF) del registro de transacciones.
  - Cada fila de la salida representa un VLF en el registro de transacciones y proporciona información relevante para ese VLF en el registro.

```
SELECT db.name, count(dbL.database_id) CuentaTotalVLF,
       convert(decimal (10,2), avg(dbL.vlf_size_mb))
TamanioPromedioVLFB
FROM sys.databases db  CROSS APPLY
    sys.dm_db_log_info(db.database_id) dbL
GROUP BY db.name
ORDER BY Total_VLF_count DESC
```

- Cuando un VLF se llena, el registro continúa en el siguiente registro de transacciones disponible.
- El archivo de registro de transacciones puede ser representado como un archivo circular que cuando el registro llega al final del archivo, inicia de nuevo desde el principio, pero sólo si todos los requerimientos han sido cumplidos y las partes inactivas han sido truncadas.
- El proceso de truncar es necesario para marcar todas las partes inactivas de modo que puedan ser usadas de nuevo y sobrescritas. (Truncar no libera el espacio en disco).
- Un registro ya no es necesario en el registro de transacciones si se cumplen todas estas premisas:
  - ✓ La transacción de la que es parte se ha confirmado.
  - ✓ Las páginas de la base de datos que cambió han sido todas escritas en un CHECKPOINT.
  - ✓ El registro no es necesario para una copia de seguridad (completa, diferencial o de log)
  - ✓ El registro no es necesario para ninguna característica que lee el registro (tales como mirroring o replicación)
- El registro lógico es una parte del registro de transacciones.
- El Log Sequence Number (LSN) identifica cada transacción en el registro de transacciones.



- EL MinLSN es el punto de partida de la transacción activa más antigua en el registro de transacciones en línea.



## ❑ Mantenimiento del log de transacciones

- Es una tarea importante en la administración de SQL Server.
- Se recomienda monitorear su crecimiento diariamente o incluso más frecuentemente si la base de datos SQL Server tiene una gran cantidad de tráfico.
- El espacio del registro de transacciones puede ser obtenido usando el comando DBCC SQLPERF:

The screenshot shows the SQL Server Management Studio interface with two tabs: 'SQLQuery2.sql - DE...PRESS.PEA (sa (59))\*' and 'SQLQuery1.sql - DE...PRE'. The query window contains the command: `DBCC SQLPERF (LOGSPACE); GO`. The results tab displays a table with the following data:

|   | Database Name      | Log Size (MB) | Log Space Used (%) | Status |
|---|--------------------|---------------|--------------------|--------|
| 1 | master             | 1,992188      | 57,15686           | 0      |
| 2 | tempdb             | 7,992188      | 8,284457           | 0      |
| 3 | model              | 7,992188      | 11,75464           | 0      |
| 4 | msdb               | 5,054688      | 19,51314           | 0      |
| 5 | AdventureWorks2019 | 71,99219      | 22,19615           | 0      |

## ❑ Backup

- Debe ser respaldado de forma regular para controlar la operación de crecimiento automática y evitar que se llene el archivo del registro de transacciones.
- No debe confundirse con el backup de los archivos de datos.
- No está disponible para el modelo de recuperación SIMPLE.
- Una vez realizado el backup el espacio del archivo del log de transacciones puede ser reutilizado

```
BACKUP LOG ACMEDB  
TO DISK = 'F:\ACMEDB.TRN'  
GO
```



## *Respaldo de la base de datos*

El objetivo de las copias de respaldo es **protegernos** ante la **pérdida de datos** por cualquier causa (falla de HW, error humano, falla de SW, malware, etc.).

Son un mecanismo para **asegurar** las propiedades ACID, ya que deben permitirnos mantener la DB **consistente**, incluso si ocurre un evento catastrófico durante una transacción.



### Clasificación

- **Completo:** base de datos completa.
- **Diferencial o incremental:** solo incluye cambios desde el último completo.
- **Registro de transacciones:** solo del log, desde el último completo o desde el último backup de log.

### Otros tipos de backup

- **Tail log:** respaldo de log de transacciones restante. **Tratar de generar un backup sobre una base de datos corrompida. Le decimos al servidor “por favor recuperáme algo”**
- **Copy only:** respaldo independiente de la secuencia de respaldos (ideal para exportar). **Generar un respaldo si alterar la secuencia de respaldo**

### Completo (FULL)

- Respala TODA la DB. Copia completa incluyendo todos los objetos de la DB. Permite restaurar la DB a su estado preciso al momento del respaldo. **Es básico este**
- Es la base de cualquier estrategia de respaldo. Antes de realizarse otro tipo de backup primero debe contarse con un FULL.
- Incluye una copia del log de transacciones.

### Diferencial

- Se respaldan todos los componentes de la DB modificados a partir del último backup completo.

- El tamaño del respaldo dependerá de los cambios efectuados.
  - Solo se respalda el estado actual de cada objeto modificado.
  - El tiempo transcurrido desde el último backup FULL y los cambios efectuados en la DB son los factores que determinarán el tamaño del respaldo diferencial.
  - Son acumulativos.

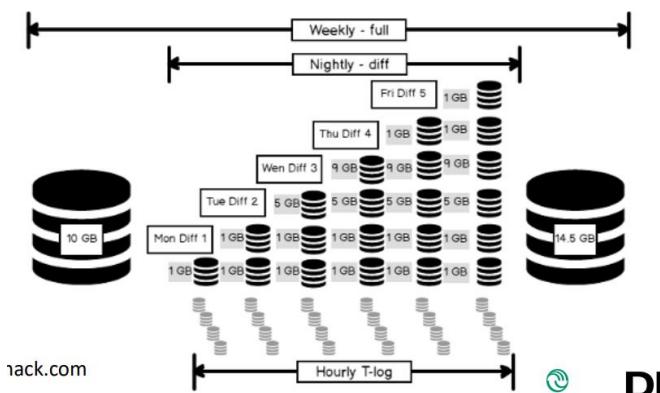
## Registro de transacciones

- Resguarda el log de transacciones, que contiene la historia de cada modificación realizada en una DB.
    - Cada respaldo del log de transacciones contiene los registros generados desde el respaldo del log de transacciones anterior.
    - Son incrementales. Para restaurar la DB a un momento específico en el tiempo debemos restaurar el último backup FULL, el último diferencial y todos los respaldos del log de transacciones desde este.

Es el que va a tener todo el historial de cambios

## Estrategia de backups

¿Cuánto tiempo lleva realizar cada backup y cuánto tiempo puede tomarle restaurar el sistema?

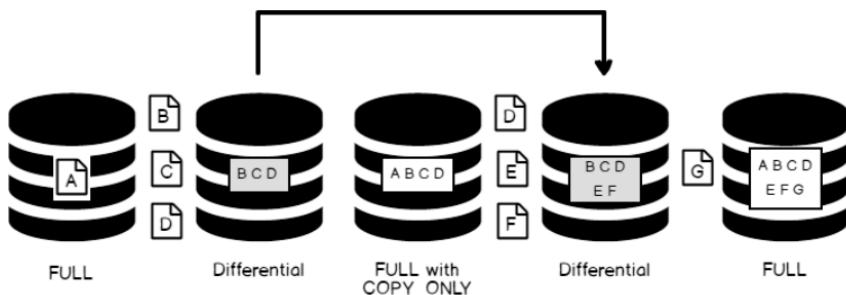


Tail log

- Ante una DB dañada u offline es posible resguardar el log de transacciones para recuperar los últimos cambios realizados (intentarlo al menos)
  - Luego de recuperar el sistema con los respaldos completos, diferenciales (si hubiera) y del log de transacciones, se puede usar el backup del tail log para minimizar la pérdida de datos.

Copy only

- No modifica el nro de secuencia como sí ocurre con los backups completos.
  - Observe que al modificarse D, E, F, no se toma como base el COPY ONLY sino el FULL anterior.



#### Otras opciones

- Se pueden generar backups comprimidos.
- Se pueden generar backups cifrados.



#### Buenas prácticas

- Estrategia 3-2-1.
- Tres copias
- En al menos dos lugares o medios.
- Uno de ellos siendo la nube.
- Verifique los backups. SIEMPRE.
- ¡No se conforme con realizar restauración! DBCC checks
- Operación que permite restaurar una base de datos a un estado anterior a través de la recuperación de una copia de seguridad previamente creada.
- Las operaciones que incluya dependerán de la estrategia empleada y del momento del evento catastrófico.
- Recomendación: documente TODO. Pruebe TODO

Al restaurar podrá indicar si desea:

- Sobrescribir la base de datos existente (WITH REPLACE)
- Conservar la configuración de replicación (WITH KEEP\_REPLICATION)
- Restringir el acceso a la base de datos restaurada (WITH RESTRICTED\_USER)
- Modificar la ruta de almacenamiento de los archivos de los FILEGROUP (WITH MOVE). Recuerde que MDF y LDF son los dos archivos mínimos a restaurar. Podemos verificar el encabezado para ver el contenido de un respaldo

```
RESTORE HEADERONLY FROM DISK = 'F:\AdventureWorks2019.bak'
```

Al restaurar una DB debemos indicar el estado en que deseamos dejar la DB al completar la operación:}

- RECOVERY (default): Deja la DB lista para ser utilizada. No se podrán restaurar respaldos adicionales.
- NORECOVERY: La DB aun no puede ser utilizada. Permite restaurar respaldos adicionales (diferenciales, de registro de transacciones).

Podemos restaurar un backup de log hasta un punto específico:

```
RESTORE LOG customer FROM DISK = 'f:\backup\customer.bak'
WITH STOPATMARK = 'lsn:12000000050000037'
```

#### *Alta disponibilidad (HA)*

**SLA = Service Level Agreement:** esta sigla significa que el proveedor; tenemos una réplica de la base de datos (copia exacta)

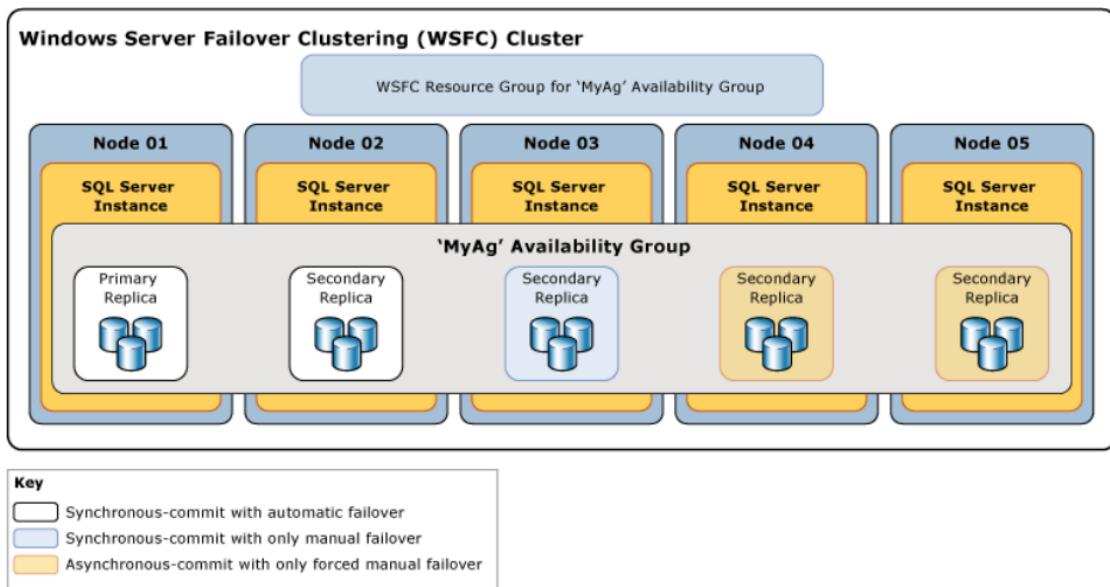
- Habilita una base de datos a mantener una réplica.
- La réplica se mantiene inactiva hasta que se le necesita para un failover.
  - La operación de failover invierte los roles de las DB de la réplica.
- Pueden operar en entornos híbridos.
- Las operaciones de lectura sobre la réplica están restringidas al licenciamiento.
- Podemos usar una réplica en modo read only (por default no permite leer) para operaciones ETL y de reporting.
  - Es posible redirigir las conexiones ReadOnly a una réplica secundaria incluso aunque intenten realizarse a la primaria.
  - Se puede utilizar la réplica para backups

Puede usar el modo **synchronous-commit** o **asynchronous-commit**.

- ASYNCHRONOUS-COMMIT (AC) en réplica primaria: no espera a la réplica secundaria para escribir el log de transacciones. Ídem si una réplica usa AC.
- SYNCHRONOUS-COMMIT (SC) en ambos (primario y secundario) la réplica primaria espera que la réplica secundaria confirme que escribió el log.
- Si se excede el timeout en una réplica secundaria la réplica primaria pasa a modo AC y tan pronto se restablece la comunicación retoman el modo SC.

El failover automático solo puede producirse entre nodos sincrónicos (SC). Los nodos AC se pueden utilizar en failover manual.

Pueden utilizarse varios nodos en un clúster para asegurar el respaldo y la disponibilidad.



## CREATE AVAILABILITY GROUP [BasicAG]

```
WITH (AUTOMATED_BACKUP_PREFERENCE = PRIMARY, BASIC,
      DB_FAILOVER = OFF, DTC_SUPPORT = NONE,
      REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT = 0)
FOR DATABASE [AdventureWorks]
REPLICA ON N'SQLVM1\MSSQLSERVER' WITH (ENDPOINT_URL =
N'TCP://SQLVM1.Contoso.com:5022', FAILOVER_MODE = AUTOMATIC,
AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
SEEDING_MODE = AUTOMATIC, SECONDARY_ROLE(ALLOW_CONNECTIONS = NO))
N'SQLVM2\MSSQLSERVER' WITH (ENDPOINT_URL =
N'TCP://SQLVM2.Contoso.com:5022', FAILOVER_MODE = AUTOMATIC,
AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
SEEDING_MODE = AUTOMATIC, SECONDARY_ROLE(ALLOW_CONNECTIONS = NO))GO
```

### Réplicas de la DB

- Técnica utilizada para copiar y sincronizar datos y objetos de una DB a otra.
- Las DB pueden estar alojadas en una misma instancia o no.
- Mantienen los datos en un estado consistente.
- Los roles a cumplir son
  - Distributor: donde se aloja la DB de distribución.
  - Publisher: donde se aloja la DB a replicar (origen).
    - Distributor y Publisher pueden ser el mismo servidor.



- Suscriber: donde se aloja la DB destino.
- Se definen artículos (tablas, vistas, SP) para determinar los datos y objetos a replicar, pudiendo filtrarse filas y columnas.
- Las tablas a replicar deben tener una PK.

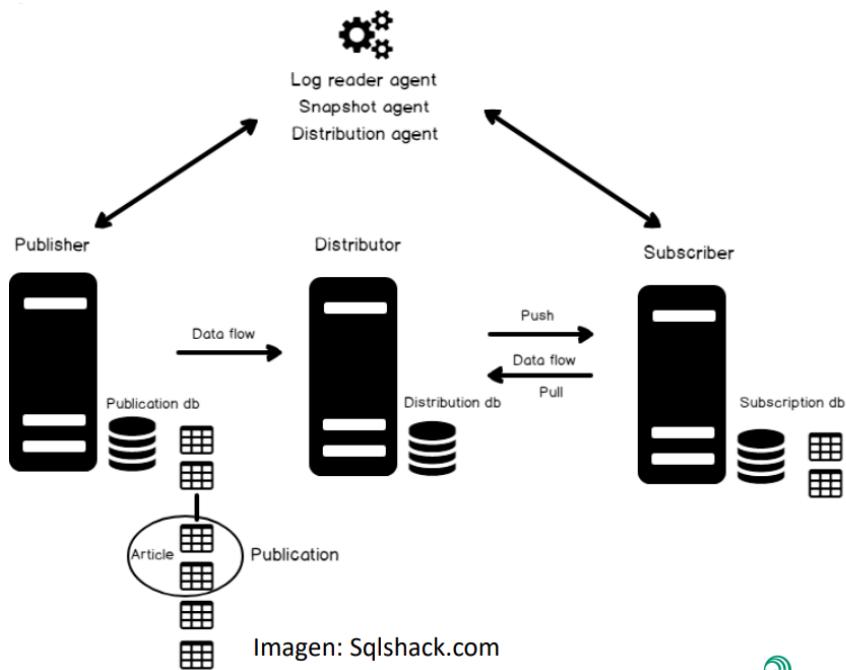


Imagen: Sqlshack.com



## Transaccional:

se usa típicamente en escenarios que requieren alta transferencia (throughput), mejorar la escalabilidad y la disponibilidad.

Por ejemplo: data warehousing, data reporting, integración desde múltiples sitios, integración de datos heterogéneos, proceso de lotes offload.

Mezcla (merge): el uso típico es en aplicaciones móviles o aplicaciones distribuidas que pueden tener conflicto de datos.

Por ejemplo: intercambio de datos con usuarios móviles, POS.

Instantánea (snapshot): Provee un estado inicial a las réplicas transaccional y mezcla.

## Log Shipping (despacho o envío de registro de transacciones)

Permite enviar en forma automática los respaldos del log de transacciones desde una base primaria en un servidor primario a una o más bases secundarias en una instancia separada.

Los backups se restauran en cada una de las bases secundarias.

Puede usarse una tercera instancia como monitor.

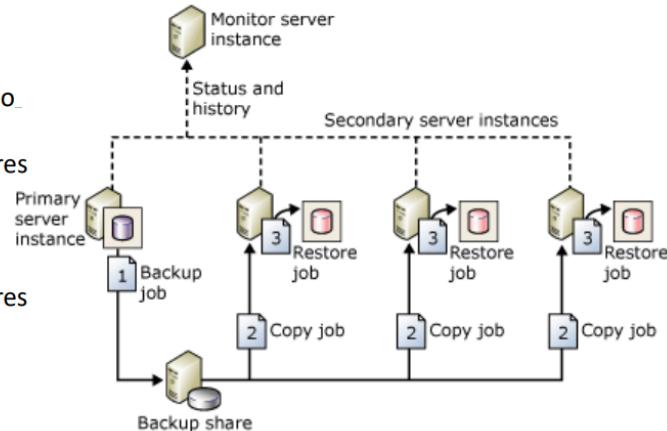
Provee así una solución DR (Disaster Recovery). No es failover automático.



Soporta acceso de solo lectura a las DB secundarias, limitado a los intervalos entre restauraciones.

La latencia del shipping es configurable (podría deshacer una catástrofe).

1. El servidor PRIMARIO ejecuta el backup y se envía a una carpeta como archivo.
2. Cada uno de los servidores secundarios realiza una copia del log en una carpeta local.
3. Cada uno de los servidores secundarios ejecuta una restauración del backup.



Todos envían información de estado e historial al servidor de monitoreo.

### Réplicas vs Alta disponibilidad

¿Cuál es la mejor herramienta?

DEPENDE la necesidad y las características del sistema.

Para un HADR completo: Alta disponibilidad

Para combinar dos DB en distintas ubicaciones: Réplica

¿Qué uso le daría a log shipping?

### Encriptación

Consiste en cifrar u ofuscar los datos por el uso de una key o password

No resuelve el problema de la protección, pero vuelve inútil los datos a quien se hace de ellos de manera ilícita.

- Se pueden utilizar contraseñas, claves simétricas y asimétricas.
- Podemos cifrar desde UN CAMPO de una tabla, un SP, a toda una DB.
- El manejo de la clave o certificado se vuelve crítico.
- La encriptación transparente (TDE) cifra los archivos de datos y se conoce como encriptación at rest.
- Cifra en tiempo real los datos y registro de transacciones.

```

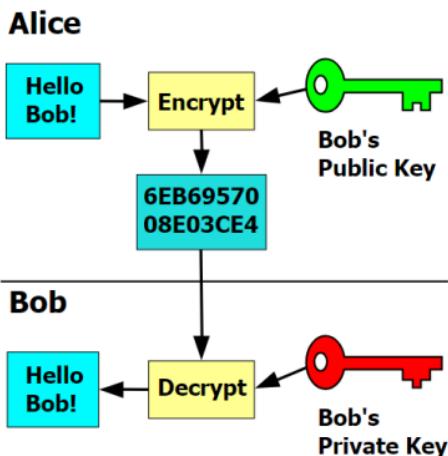
USE AdventureWorks2022;
GO
-- Agregamos un campo para los datos cifrados
ALTER TABLE Sales.CreditCard
    ADD TarjetaCreditoCifradaFraseClave VARBINARY(256);
GO
-- Obtenemos la clave de cifrado. Lo cargariamos desde otra capa.
DECLARE @FraseClaveCargadaPorUsuario NVARCHAR(128);
SET @FraseClaveCargadaPorUsuario = 'QuieroMiPanDanes';

-- Ciframos el campo de la tarjeta de crédito registro 3681.
-- Agrega un hash (el PK IdTarjetaCredito al cifrado)
UPDATE Sales.CreditCard
SET CardNumber_ TarjetaCreditoCifradaFraseClave =
EncryptByPassPhrase(@FraseClaveCargadaPorUsuario
    , NumeroTarjeta, 1, CONVERT(varbinary, IdTarjetaCredito))
WHERE IdTarjetaCredito = '3681';
GO

```

Es posible cifrar las conexiones con todos los clientes o con algunos específicos.

Requiere la configuración de certificados digitales.



### LINK CLASES

Link a las clases del segundo nivel de BDD:

<https://www.youtube.com/watch?v=EtNHuebbGM&list=PLIZHXd29biO6crZjkxUmMHGpKqFjIW5mu>