



Ministerio de Producción  
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



Programa  
**111**  
**mil**  
VOS PODÉS  
SER UNO.

# MOTORES DE BASES DE DATOS RELACIONALES



# Agenda

- Repaso
  - Modelo Relacional
  - Lenguajes de Consulta
- Motores de Bases de Datos Comerciales
- Transacciones
  - Fallos
  - Estados
  - ACID
- Indices
- Procedimientos Almacenados y Funciones
- Vistas
- Triggers o Disparadores



# Modelo Relacional: Repaso

- **Base de Datos relacional:** Conjunto de tablas y relaciones entre las mismas.
- **Tabla:** Cada fila de la tabla representa una relación entre un conjunto de valores. De manera informal, cada tabla es un conjunto de instancias de entidades, y cada fila es una instancia
- **Atributos de la tabla:** columnas, propiedades de la entidad

<i>número-cuenta</i>	<i>nombre-sucursal</i>	<i>saldo</i>
C-101	Centro	500
C-102	Navacerrada	400
C-201	Galapagar	900
C-215	Becerril	700
C-217	Galapagar	750
C-222	Moralzarzal	700
C-305	Collado Mediano	350



# Esquema vs Instancia de la BD

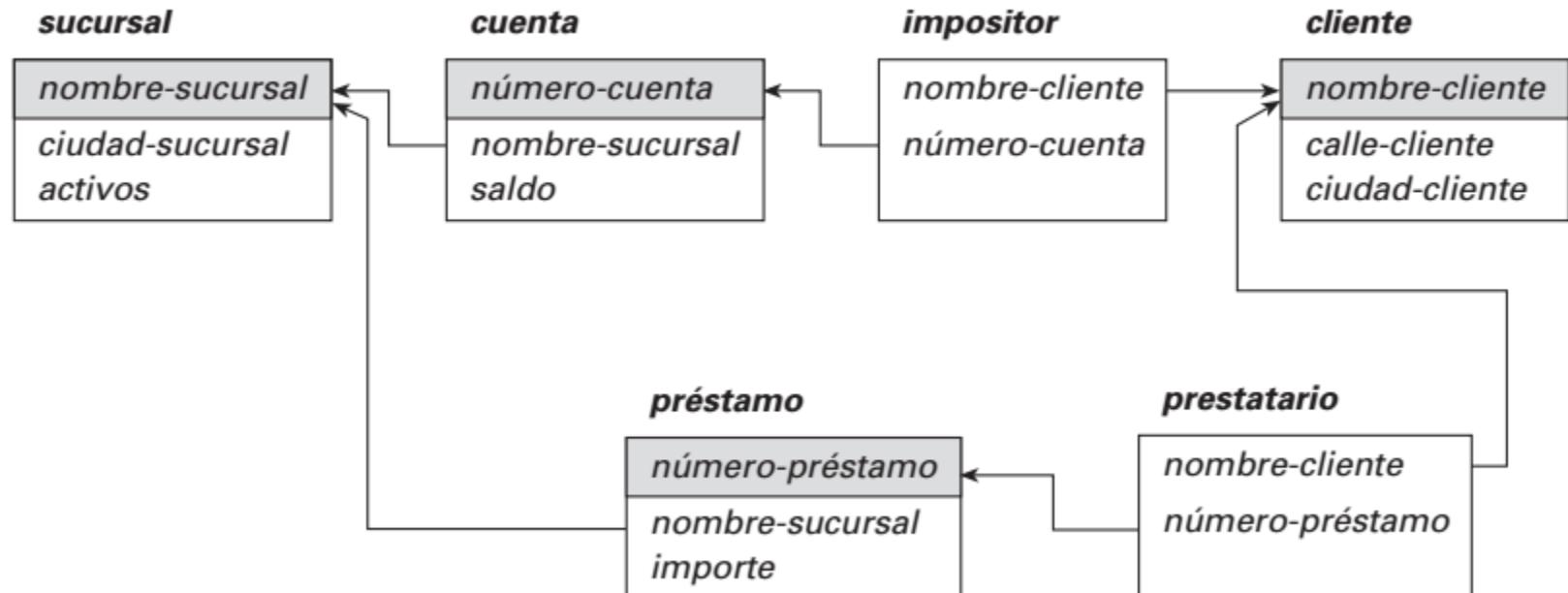
- **Esquema de BD ≠ Instancia/Ejemplar de BD**

**Esquema:** Concepto de Molde. Define la estructura de la BD considerando sus tablas, sus campos en cada tabla y las relaciones entre cada campo y cada tabla.

**Instancia/Ejemplar:** Ejemplar de la base de datos obtenida de un esquema/molde. Se define por la información que está guardada en un momento determinado

# Diagrama de Esquema

- Representación gráfica de un esquema de BD
- No es igual a un Diagrama de E-R





# Lenguajes de Consulta

- Lenguaje en el que un usuario solicita información de la base de datos.
- Suelen ser de un nivel superior que el de los lenguajes de programación habituales.
- Dependiendo de la información brindada por el usuario se clasifican en:
  - Lenguajes Procedimentales
  - Lenguajes No Procedimentales



# Tipos de Lenguajes de Consulta

- **Lenguajes procedimentales:** el usuario indica al sistema que lleve a cabo una serie de operaciones en la base de datos para calcular el resultado deseado
- **Lenguajes no procedimentales:** el usuario describe la información deseada sin dar un procedimiento concreto para obtener esa información.
- BD comerciales ofrecen lenguajes híbridos.
  - SQL (Structured Query Language)



# Lenguajes de Consulta Procedimentales

- El álgebra relacional es un lenguaje de consulta procedimental
- Consiste en un conjunto de operaciones que toman como entrada una o dos relaciones y producen como resultado una nueva relación
- Las operaciones fundamentales del álgebra relacional son selección, proyección, unión, diferencia de conjuntos, producto cartesiano y renombramiento.



# Lenguajes de Consulta No Procedimentales

- El cálculo relacional de tuplas y el cálculo relacional de dominios son lenguajes no procedimentales que representan la potencia básica necesaria en un lenguaje de consultas relacionales.
- Son lenguajes rígidos, formales, que no resultan adecuados para los usuarios ocasionales de los sistemas de bases de datos
- Describen la información deseada sin dar un procedimiento específico para obtenerla. Se diferencia del álgebra relacional donde proporcionamos una serie de procedimientos que generan la respuesta a la consulta.



# Motores de Bases de Datos Comerciales

- Aplicabilidad en la industria/negocios
  - Mantener los datos de sus aplicaciones internas y de uso externo
  - Asegurar que siempre se tendrá acceso a la información crítica de clientes, datos de productos, datos sociales y financieros particulares, tales como compras, cumplimientos beneficios y gastos
  - Uso desde simples aplicaciones de escritorio que consumen datos de una única base de datos, hasta almacenes de datos de empresas de gran escala (Terabytes, múltiples DBs)



# Características de BD Relacionales

- Un producto para calificar y ser incluido en la categoría de base de datos relacional debe:
  - Proveer almacenamiento de datos.
  - Organizar los datos en un modelo relacional, formulado con tablas de filas y columnas.
  - Permitir a todos los usuarios recuperar, editar, retornar y remover datos
- La comparación entre BD Relacionales comerciales se basa en:
  - Nivel de satisfacción (basado en revisiones de los usuarios)
  - Escala (basada en porción de mercado, tamaño del vendedor e impacto social).
  - <http://db-engines.com/en/ranking>



# Alternativas de BD Comerciales

Rank			DBMS	Database Model	Score		
Mar 2017	Feb 2017	Mar 2016			Mar 2017	Feb 2017	Mar 2016
1.	1.	1.	Oracle	Relational DBMS	1399.50	-4.33	-72.51
2.	2.	2.	MySQL	Relational DBMS	1376.07	-4.23	+28.36
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1207.49	+4.04	+71.00
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	357.64	+3.96	+58.01
5.	5.	↓ 4.	MongoDB	Document store	326.93	-8.57	+21.60
6.	6.	6.	DB2	Relational DBMS	184.91	-2.99	-3.02
7.	↑ 8.	7.	Microsoft Access	Relational DBMS	132.94	-0.45	-2.09
8.	↓ 7.	8.	Cassandra	Wide column store	129.19	-5.19	-1.14
9.	9.	↑ 10.	SQLite	Relational DBMS	116.19	+0.88	+10.42
10.	10.	↓ 9.	Redis	Key-value store	113.01	-1.03	+6.79

- <http://db-engines.com/en/ranking>
- [https://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n\\_de\\_sistemas\\_administradores\\_de\\_bases\\_de\\_datos\\_relacionales](https://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n_de_sistemas_administradores_de_bases_de_datos_relacionales)



# MySQL

- Sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation
- Considerada como la base datos open source más popular del mundo y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.
- MariaDB está basado en MySQL (fork de MySQL) y está disponible bajo los términos de la licencia GPL v2. MariaDB es desarrollado por la comunidad en conjunto con Monty Program Ab como su principal encargado.





# MySQL: Características y Ventajas

- Aprovecha la potencia de sistemas multiprocesador (implementación multihilo)
- Soporta gran cantidad de tipos de datos..
- Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc). Infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación.
- Soporta hasta 32 índices por tabla.
- Gestión de usuarios y passwords - buen nivel de seguridad en los datos.
- Open source y gratis (Community Server) - MariaDB alternativa.
- Gran popularidad entre proyectos de software Web
- Fácil instalación y configuración.
- Escalabilidad y flexibilidad
- Alto rendimiento
- Alta disponibilidad
- Robusto soporte transaccional



# Transacciones

- El concepto de transacción proporciona un mecanismo para definir las unidades lógicas del procesamiento de una base de datos.
- Una transacción se inicia por la ejecución de un programa de usuario escrito en un lenguaje de manipulación de datos de alto nivel o en un lenguaje de programación (por ejemplo SQL, COBOL, C, C++ o Java), y está delimitado por instrucciones de la forma inicio transaccion y fin transaccion.
  - La transacción consiste en todas las operaciones que se ejecutan entre inicio transaccion y el fin transaccion.
- Los sistemas de procesamiento de transacciones son sistemas con grandes bases de datos y cientos de usuarios concurrentes ejecutando transacciones de bases de datos.
  - Ejemplos: sistemas de reservas en aerolíneas, bancos, procesamiento de tarjetas de crédito, mercado de acciones, etcétera



# Transacciones: Ejemplo

- Transacción puede conceptualizarse como una colección de operaciones que forman una única unidad lógica de trabajo. Un DBMS debe asegurar que la ejecución de las transacciones se realice adecuadamente a pesar de la existencia de fallos: o se ejecuta la transacción completa o no se ejecuta en absoluto.
- **Por ejemplo**, una transferencia de fondos desde una cuenta corriente a una cuenta de ahorros es una operación simple desde el punto de vista del cliente; sin embargo, en el sistema de base de datos, está compuesta internamente por varias operaciones.
  - Es esencial que tengan lugar todas las operaciones o que, en caso de fallo, ninguna de ellas se produzca.
  - Sería inaceptable efectuar el cargo de la transferencia en la cuenta corriente y que no se abonarse en la cuenta de ahorros.



# Fallos

- **Fallo de la computadora (caída del sistema).**
  - Durante la ejecución de una transacción se produce un error del hardware, del software o de la red.
  - Las caídas del hardware normalmente se deben a fallos en los medios (por ejemplo, un fallo de la memoria principal).
- **Un error de la transacción o del sistema.**
  - Fallo en las operaciones de la transacción como un desbordamiento de entero o una división por cero.
  - Valores erróneos de los parámetros o debido a un error lógico de programación.
  - El usuario puede interrumpir la transacción durante su ejecución.
- **Errores locales o condiciones de excepción detectados por la transacción.**
  - Condiciones que necesitan cancelar la transacción. Por ejemplo, puede que no se encuentren los datos para la transacción.
  - Una condición de excepción, como un saldo de cuenta insuficiente en una base de datos bancaria, puede provocar que una transacción, como la retirada de fondos, sea cancelada.



# Fallos (2)

- **Control de la concurrencia.**
  - El método de control de la concurrencia puede optar por abortar la transacción, para restablecerla más tarde, porque viola la serialización o porque varias transacciones se encuentran en estado de bloqueo.
- **Fallo del disco rígido.**
  - Algunos bloques del disco pueden perder sus datos debido a un mal funcionamiento de la lectura o la escritura o porque se ha caído la cabeza de lectura/escritura del disco.
- **Problemas físicos y catástrofes.**
  - Se refiere a una lista interminable de problemas que incluye fallos de alimentación o aire acondicionado, fuego, robo, sabotaje, sobrescritura de discos y cintas por error, y montaje de la cinta errónea por parte del operador.



# ACID (AC--)

- **Atomicidad**

- La Atomicidad requiere que cada transacción sea "todo o nada": si una parte de la transacción falla, todas las operaciones de la transacción fallan, y por lo tanto la base de datos no sufre cambios. Un sistema atómico tiene que garantizar la atomicidad en cualquier operación y situación, incluyendo fallas de alimentación eléctrica, errores y caídas del sistema.

- **Consistencia**

- La propiedad de Consistencia se asegura que cualquier transacción llevará a la base de datos de un estado válido a otro estado válido. Cualquier dato que se escriba en la base de datos tiene que ser válido de acuerdo a todas las reglas definidas.



# ACID (--ID)

- **Aislamiento**

- El Aislamiento ("Isolation" en inglés) se asegura que la ejecución concurrente de las transacciones resulte en un estado del sistema que se obtendría si estas transacciones fueran ejecutadas una detrás de otra. Cada transacción debe ejecutarse en aislamiento total; por ejemplo, si T1 y T2 se ejecutan concurrentemente, luego cada una debe mantenerse independiente de la otra.

- **Durabilidad**

- La Durabilidad significa que una vez que se confirmó una transacción quedará persistida, incluso ante eventos como pérdida de alimentación eléctrica, errores y caídas del sistema. Por ejemplo, en las bases de datos relacionales, una vez que se ejecuta un grupo de sentencias SQL, los resultados tienen que almacenarse inmediatamente (incluso si la base de datos se cae inmediatamente después).



# Estados de una transacción

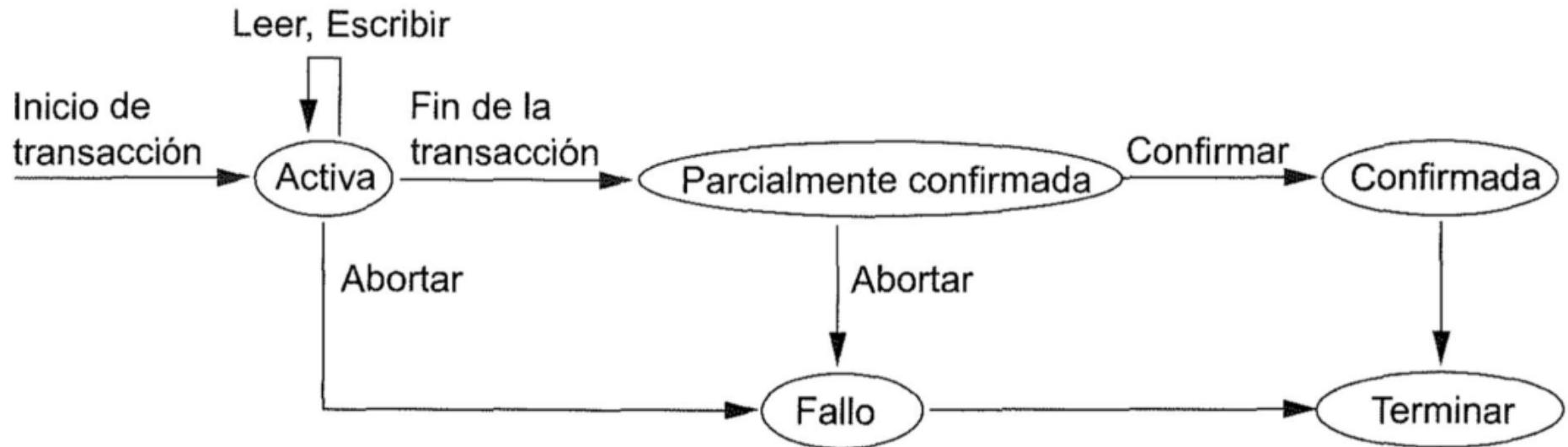
- **Activa**, el estado inicial; la transacción permanece en este estado durante su ejecución.
- **Parcialmente confirmada**, después de ejecutarse la última instrucción.
- **Fallida**, tras descubrir que no puede continuar la ejecución normal.
- **Abortada**, después de haber retrocedido la transacción y restablecido la base de datos a su estado anterior al comienzo de la transacción.
- **Confirmada**, tras completarse con éxito.



# Operaciones sobre transacciones

- **BEGIN\_TRANSACTION.** Marca el inicio de la ejecución de una transacción.
- **READ o WRITE.** Especifican operaciones de lectura o escritura en los elementos de la base de datos que se ejecutan como parte de una transacción.
- **END\_TRANSACTION.** Especifica que las operaciones READ y WRITE de la transacción han terminado y marca el final de la ejecución de la transacción. Sin embargo, en este punto puede ser necesario comprobar si los cambios introducidos por la transacción pueden aplicarse de forma permanente a la base de datos (confirmados) o si la transacción se ha cancelado porque viola la serialización o por alguna otra razón.
- **COMMIT\_TRANSACTION.** Señala una finalización satisfactoria de la transacción, por lo que los cambios (actualizaciones) ejecutados por la transacción se pueden enviar con seguridad a la base de datos y no se desharán.
- **ROLLBACK (o ABORT).** Señala que la transacción no ha terminado satisfactoriamente, por lo que deben deshacerse los cambios o efectos que la transacción pudiera haber aplicado a la base de datos.

# Diagrama de estados y operaciones



Los programadores de SQL son los responsables de iniciar y finalizar las transacciones en puntos que exijan la coherencia lógica de los datos. El programador debe definir la secuencia de modificaciones de datos que los dejan en un estado coherente en relación con las reglas de negocios de la organización. Además, deben incluir estas instrucciones de modificación en una sola transacción de forma que el Motor de base de datos puede hacer cumplir la integridad física de la misma.



# Responsabilidad del DBMS

- El DBMS debe proporcionar los mecanismos que aseguren la integridad física de cada transacción.
  - **Servicios de bloqueo** que preservan el aislamiento de la transacción.
  - **Servicios de registro** que aseguran la durabilidad de la transacción. Aunque se produzca un error en el hardware del servidor, el sistema operativo o la instancia de Motor de base de datos, la instancia utiliza registros de transacciones, al reiniciar, para revertir automáticamente las transacciones incompletas al punto en que se produjo el error del sistema
  - **Características de administración** de transacciones que exigen la atomicidad y coherencia de la transacción. Una vez iniciada una transacción, debe concluirse correctamente; en caso contrario, la instancia de Motor de base de datos deshará todas las modificaciones de datos realizadas desde que se inició la transacción.



# Concurrencia de transacciones

- **Enfoque de ejecución secuencial** de transacciones es más sencillo de implementar, pero menos eficiente. Para comenzar una transacción es necesario finalizar la anterior.
- **Enfoque de ejecución concurrente** permite varias transacciones que actualizan concurrentemente los datos y puede provocar complicaciones en la consistencia de los mismos. Asegurar la consistencia a pesar de la ejecución concurrente de las transacciones requiere un trabajo extra.
- El **esquema de control de concurrencia** de un DBMS controla la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos.



# Ventajas de la concurrencia

- **Productividad y utilización de recursos mejorados.**
  - La productividad (throughput) del sistema -es decir, en el número de transacciones que puede ejecutar en un tiempo dado- aumenta cuando varias transacciones se ejecutan en paralelo. Varias operaciones de una transacción se pueden ejecutar en paralelo. Por ej., las operaciones de E/S, como uso de CPU y discos pueden trabajar en paralelo en una computadora
  - Análogamente, la utilización del procesador y del disco aumenta también. De lo contrario, estarían desperdiciándose recursos computacionales
- **Tiempo de espera reducido.**
  - Frente a transacciones cortas y largas, la ejecución concurrente reduce los retardos impredecibles en la ejecución de las transacciones (operan en partes diferentes de la base de datos)
  - Se reduce también el tiempo medio de respuesta.



# Índices

- Estructuras complementarias en DBMS que se asocian con los atributos de las tablas y agilizan la operaciones de búsqueda.
  - Especialmente con consultas recurrentes
- Juegan el mismo papel que los índices de los libros o los catálogos de fichas de las bibliotecas.
  - Las palabras de índice están ordenadas, lo que facilita la búsqueda. Además, el índice es mucho más pequeño que el libro, con lo que se reduce aún más el esfuerzo necesario para encontrar las palabras en cuestión.
  - Por ejemplo, para recuperar un registro de cuenta dado su número de cuenta, el sistema de bases de datos buscaría en un índice para encontrar el bloque de disco en que se encuentra el registro correspondiente, y entonces extraería ese bloque de disco para obtener el registro cuenta.



# Tipos de Índices

- **Índices ordenados.** Estos índices están basados en una disposición ordenada de los valores.
- **Índices asociativos (hash index).** Estos índices están basados en una distribución uniforme de los valores a través de una serie de cajones (buckets). El valor asignado a cada cajón está determinado por una función, llamada función de asociación (hash function)

Los atributos o conjunto de atributos usados para buscar en un archivo se llaman **claves de búsqueda** (que no es lo mismo que clave primaria y/o secundaria)

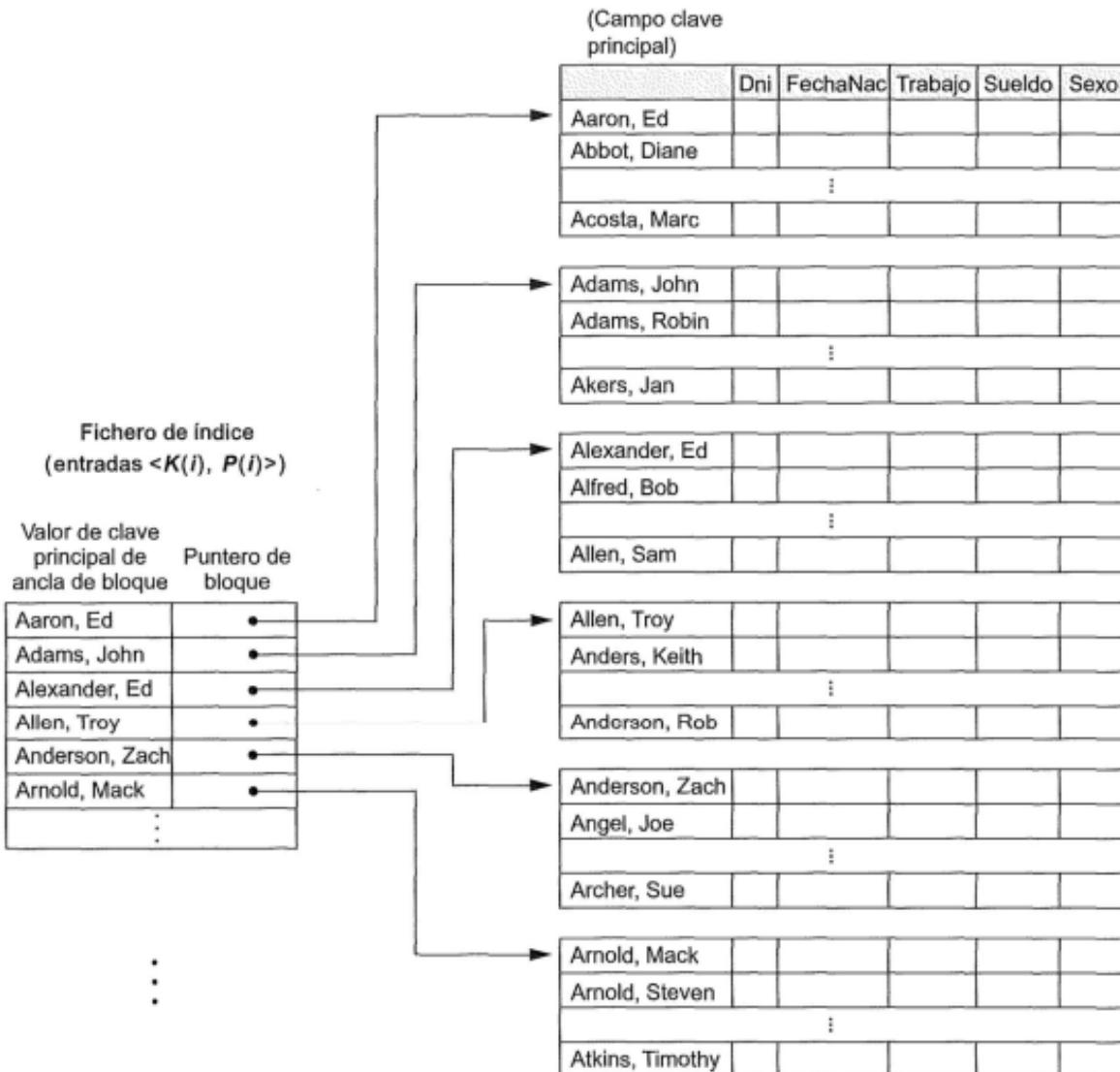


# Criterios de comparación de Índices

- **Tipos de acceso:** los tipos de acceso que se soportan eficazmente. Estos tipos podrían incluir la búsqueda de registros con un valor concreto en un atributo, o la búsqueda de los registros cuyos atributos contengan valores en un rango especificado.
- **Tiempo de acceso:** el tiempo que se tarda en hallar un determinado elemento de datos, conjunto de elementos, usando la técnica en cuestión.
- **Tiempo de inserción:** el tiempo empleado en insertar un nuevo elemento de datos. Este valor incluye el tiempo utilizado en hallar el lugar apropiado donde insertar el nuevo elemento de datos, así como el tiempo empleado en actualizar la estructura del índice.
- **Tiempo de borrado:** el tiempo empleado en borrar un elemento de datos. Este valor incluye el tiempo utilizado en hallar el elemento a borrar, así como el tiempo empleado en actualizar la estructura del índice.
- **Espacio adicional requerido:** el espacio adicional ocupado por la estructura del índice. Como normalmente la cantidad necesaria de espacio adicional suele ser moderada, es razonable sacrificar el espacio para alcanzar un rendimiento mejor.



# Ejemplo de Índice ordenado





# Índices Ordenados: subtipos

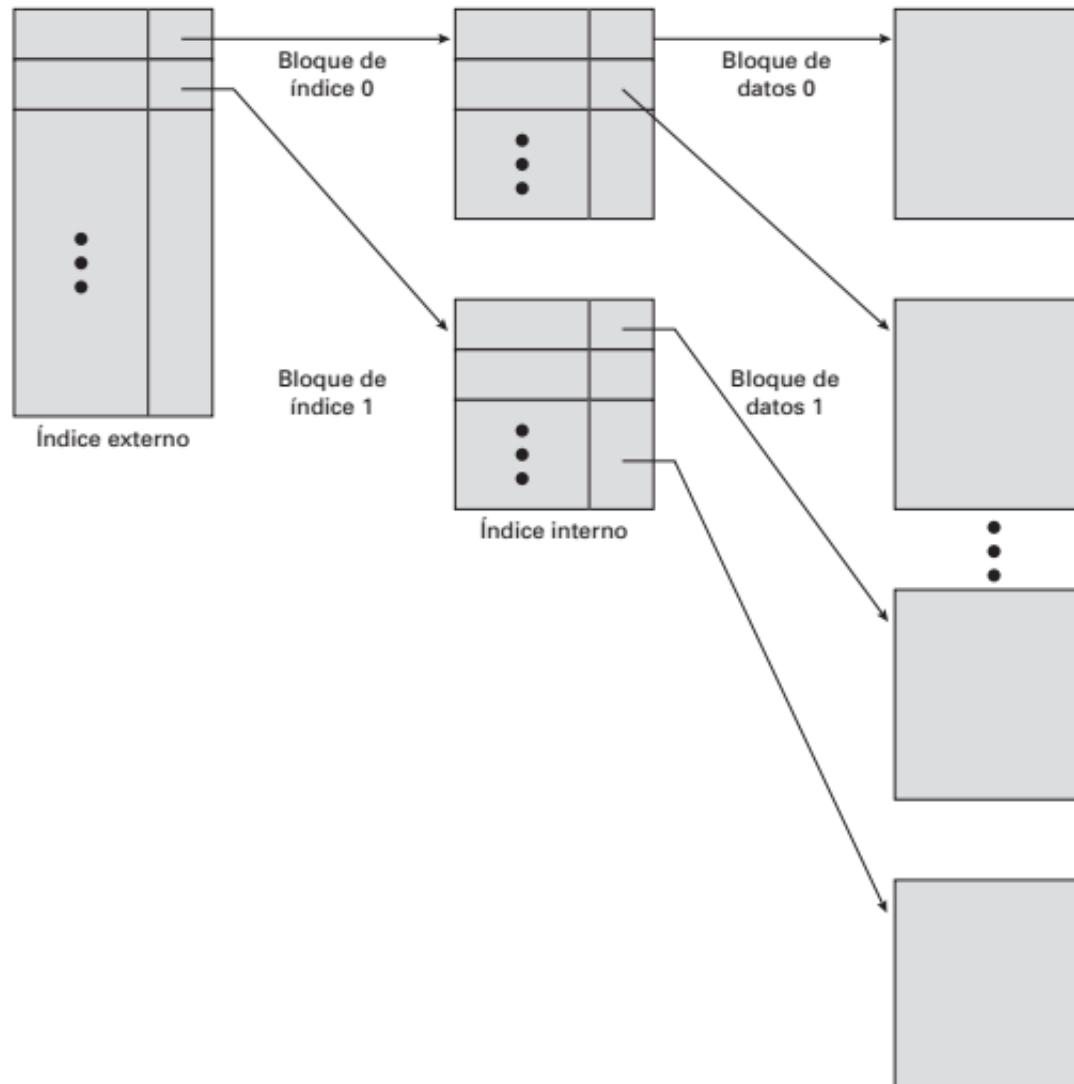
- **Índice primario o índices con agrupación.** Índice cuya clave de búsqueda especifica el orden secuencial del archivo (archivos ordenados secuencialmente). La clave de búsqueda de un índice primario es normalmente la clave primaria (pero no necesariamente)
- **Índices secundarios o índices sin agrupación.** Índices cuyas claves de búsqueda especifican un orden diferente del orden secuencial del archivo. Un archivo puede tener varios índices secundarios además de su método de acceso principal.



# Índices Multinivel

- Solución para índices muy grandes
- El índice se trata como si fuese un archivo secuencial y se construye otro índice sobre el índice con agrupación
- Para localizar un registro se usa en primer lugar una búsqueda binaria sobre el índice más externo para buscar el registro con el mayor valor de la clave de búsqueda que sea menor o igual al valor deseado.
  - La búsqueda continúa en el bloque apuntado del índice más interno (sucesivamente)
  - Si el índice exterior es demasiado grande aún (debe almacenarse en disco), se pueden construir nuevos niveles de índices.
- Los índices multinivel están estrechamente relacionados con la estructura de árbol, tales como los árboles binarios usados para la indexación en memoria.

# Índices Multinivel: Ejemplo





# Indices en SQL

- **create index** <nombre-índice> **on** <nombre-relación> (<lista-atributos>)
- **drop index** <nombre-índice>

*lista-atributos* es la lista de atributos de la relación que constituye la clave de búsqueda del índice.

Ejemplo:

**create index índice-s on sucursal (nombre-sucursal)**

índice llamado índice-s de la relación sucursal con la clave de búsqueda nombre-sucursal



# Procedimientos almacenados y funciones

- Comúnmente llamados procedimientos almacenados (en inglés stored procedures), aunque pueden ser funciones o procedimientos.
  - Al igual que en lenguajes de programación, la diferencia se encuentra en si el módulo define o no un valor de retorno explícito y la forma de uso/invocación.
- Son módulos de programa de bases de datos (procedimientos o funciones) que el DBMS almacena y ejecuta en el servidor de bases de datos.
- El término utilizado en el estándar SQL para los procedimientos almacenados es módulos almacenados persistentes, porque el DBMS almacena persistentemente estos programas.



# Procedimientos y funciones: Utilidad

- Si varias aplicaciones necesitan un mismo programa de base de datos, este último se puede almacenar en el servidor e invocarlo desde esas aplicaciones. Esto reduce la duplicidad del esfuerzo y mejora la modularidad del software.
- La ejecución de un programa en el servidor puede reducir el costo derivado de la transferencia y la comunicación de datos entre el cliente y el servidor en ciertas situaciones.
- Ayudan a mantener integridad de los datos y su consistencia. Incrementa la seguridad de las operaciones en el DBMS



# Procedimientos y funciones: Sintaxis

**CREATE PROCEDURE** <nombre del procedimiento>  
«parámetros»  
<declaraciones locales>  
<cuerpo del procedimiento>;

**CREATE FUNCTION** <nombre de la función> «parámetros»  
**RETURNS** <tipo de devolución>  
<declaraciones locales>  
<cuerpo de la función>;

**CREATE PROCEDURE** <nombre del procedimiento>  
«parámetros»  
**LANGUAGE** <nombre del lenguaje de programación>  
**EXTERNAL NAME** <nombre de ruta del fichero>;



# Procedimientos y funciones: elementos

- Parámetros y tipos de retorno

```
CREATE PROCEDURE saldo_cuenta(  
    IN dni INT,  
    OUT saldo INT,  
    OUT total_de_operaciones INT,  
    OUT fecha_ultima_operacion INT)  
BEGIN  
    ...
```

- Variables

```
DECLARE <nombre_variable> tipo_de_dato(tamaño) DEFAULT  
valor_defecto;  
DECLARE x, y INT DEFAULT 0;  
DECLARE total INT DEFAULT 0;  
SET total= 10;
```



# Procedimientos y funciones: elementos

- Estructuras de control: IF

```
IF expression THEN
    statements;
ELSEIF elseif-expression THEN
    elseif-statements;
...
ELSE
    else-statements;
END IF;
```

- Estructuras de control: CASE

```
CASE case_expression
    WHEN when_expression_1 THEN commands
    WHEN when_expression_2 THEN commands
    ...
    ELSE commands
END CASE;
```



# Procedimientos y funciones: elementos

- Estructuras de control: LOOP

```
WHILE expression DO
    statements
END WHILE
```

```
REPEAT
    statements;
UNTIL expression
END REPEAT
```

- Cursor

```
DECLARE cursor_name CURSOR FOR select_statement;
OPEN cursor_name;
FETCH cursor_name INTO variables list;
CLOSE cursor_name;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
```



# Procedimientos y funciones: MySQL

```
CREATE PROCEDURE saldo_cuenta (IN dni CHAR(8), OUT saldo INT)

BEGIN

    SELECT sum(monto) as saldo FROM movimientos_cuenta MV WHERE
MV.dni = dni;

END
```

## Desde la Aplicación:

```
CALL saldo_cuenta('12345678');
```



# Vistas

- Puede conceptualizarse como una tabla de sólo lectura o tabla virtual que deriva de otras tablas
  - Las tablas de las que se deriva pueden ser tablas base (tablas propias de la base de datos, almacenadas físicamente en la base de datos) o vistas definidas anteriormente
  - Se restringen las operaciones de actualización, sólo puede consultarse

**create view** <nombre-vista> **as** <expresión de consulta>

...donde <expresión de consulta> puede ser cualquier consulta válida.



# Vistas: utilidad

- Permiten encapsular la lógica compleja de una consulta y simplificar la consulta desde una aplicación.
  - Concepto similar a funciones en programación estructurada
- Frente a múltiples usuarios, permite esconder detalles de implementación de la DB y exponer sólo una parte de dichos datos
- En el caso de vistas indexadas, puede mejorarse el rendimiento general del sistema, sobre todo con consultas recurrentes
  - Si la vista no fuese indexada, la performance no mejora. Es equivalente a ejecutar la consulta original



# Vistas: ejemplo MySQL

```
CREATE VIEW detalle_ultimos_10_movimientos AS
SELECT CL.dni, MO.* FROM Clientes CL
JOIN Cuentas CU ON (CL.Id = CU.Id_Cliente)
JOIN Movimientos MO ON (CU.Id = MO.Id_Cuenta)
ORDER BY MO.Fecha DESC
LIMIT 10
```

## Desde la Aplicación u otra consulta:

```
SELECT * FROM detalle_ultimos_10_movimientos;
```

```
SELECT SUM(monto) FROM (SELECT * FROM
detalle_ultimos_10_movimientos WHERE dni='12345678');
```



# Disparadores o Triggers

- Es una orden que el sistema ejecuta de manera automática como efecto secundario de la modificación de la base de datos
- Modelo evento-condición-acción
  - Condiciones en las que se va a ejecutar el disparador. Esto se descompone en un evento que causa la comprobación del disparador y una condición que se debe cumplir para ejecutar el disparador.
  - Acciones que se van a realizar cuando se ejecute el disparador
- Se almacenan en la DB, por lo que son persistentes y accesibles para todas las operaciones de la base de datos.
  - Una vez se almacena un disparador en la base de datos, el sistema de base de datos asume la responsabilidad de ejecutarlo cada vez que ocurra el evento especificado y se satisfaga la condición correspondiente



# Disparadores: casos de utilidad

- Registro o log de eventos
  - Registrar modificaciones realizadas en la tabla de movimientos de cuenta, incluyendo información de fecha y usuario que efectuó la modificación.
- Monitoreo de alertas o condiciones de excepción
  - Si un cliente desea retirar más dinero del que tiene disponible, y no tiene activo la opción de descubierto, la operación puede bloquearse, evitando un estado inconsistente.
- Actualización de estados (o valores totales)
  - Puede modificarse/mantenerse actualizado atributos de tipo estado (activo, en espera, inactivo, finalizado)
  - Por ejemplo, el estado de un pedido en un sistema de venta de productos (e-commerce).



# Disparadores: Sintaxis MySQL

```
CREATE TRIGGER <nombre_trigger>
```

```
BEFORE / AFTER
```

```
-- momento de activación
```

```
INSERT / UPDATE / DELETE on <tabla_evaluada>
```

```
-- evento de activación
```

```
FOR EACH ROW / FOR EACH STATEMENT (2do no soportado en MySQL 5.7)
```

```
-- granularidad, generalmente para cada fila
```

```
BEGIN
```

```
    -- Cuerpo del trigger, acciones
```

```
END ;
```

```
DROP TRIGGER <nombre_trigger>;
```



# Disparadores: ejemplo en MySQL

- Monitoreo de alertas o condiciones de excepción

```
CREATE TRIGGER `control_alertas`
BEFORE UPDATE ON `tabla_monitoreada`
FOR EACH ROW
BEGIN
    DECLARE msg VARCHAR(255);
    IF (<Condicion_Fallo> = true) THEN
        set msg = "Condición de error!";
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
    END IF;
END;
```



# Disparadores: ejemplo en MySQL

- Actualización de estados o valores totales

```
CREATE TRIGGER `estado_pedido_aprobado`  
AFTER INSERT ON `aprobaciones_pedidos`  
FOR EACH ROW  
  
BEGIN  
  
    UPDATE pedidos P SET P.estado = `aprobado` WHERE P.Id =  
NEW.Id_Pedido;  
  
END;
```