SQL

Introducción

Lenguaje de definición de datos

Lenguaje de manipulación de datos

Introducción

- En un sistema de bases de datos las operaciones sobre los datos se realizan a través de consultas formuladas con un lenguaje específico declarativo.

- SQL se considera el lenguaje estándar de base de datos relacional.

- SQL es una abreviatura de Structured Query Language (Lenguaje de Consulta Estructurado).

- El lenguaje SQL consta de tres lenguajes específicos: DDL, DML, DCL.

Introducción

- El lenguaje de definición de datos (DDL- Data Definition Language) proporciona órdenes para definir, eliminar y modificar tablas, así como para crear índices y vistas.

- El lenguaje de manipulación de datos (DML- Data Management Languaje) está basado en el álgebra relacional e incluye órdenes para insertar, suprimir, y modificar tuplas (filas) de la base de datos.

- Lenguaje de control de datos (DCL- Data Control Language) permite establecer derechos de acceso a los usuarios, comprobaciones de integridad y control de transacciones. Incluye órdenes para dar y quitar privilegios, así como para completar y abortar transacciones.

Introducción

Lenguaje de definición de datos

Lenguaje de manipulación de datos

Lenguaje de definición de datos

CREATE TABLE

Esta orden permite definir el esquema de la base de datos, es decir, las tablas que forman el sistema. Para crear una tabla es necesario indicar los campos que la componen junto con los tipos asociados a cada uno.

Su sintaxis es la siguiente:

```
CREATE TABLE nombre-tabla
    (nombre-columna-1 tipo-columna1 [NOT NULL],
    nombre-columna-2 tipo-columna2 [NOT NULL],
    ……..
    nombre-columna-N tipo-columna N[NOT NULL],
    [PRIMARY KEY lista-cols]
    [FOREIGN KEY (ref1[, ref2 [, ...]]) REFERENCES
    tabla externa [(campo externo1[,campo externo2 [, ...])];
```

Lenguaje de definición de datos

```
CREATE TABLE PROFESOR (
            C-PROF      INTEGER NOT NULL,
            NOMBRE      CHAR (25) NOT NULL,
            CATEGORIA   CHAR (10),
            C-DPTO      CHAR (3),
            F-CONTR     DATE,
PRIMARY KEY (C-PROF));
```

ALTER TABLE
Con esta orden podemos añadir (ADD), modificar (MODIFY) o eliminar (DROP) elementos de la definición de la tabla.

ALTER TABLE nombre-tabla ADD (nombre-col tipo-col [NOT NULL]);

ALTER TABLE nombre-tabla MODIFY nombre-col tipo-col;

ALTER TABLE nombre-tabla DROP nombre-col;

Lenguaje de definición de datos

Supongamos que necesitamos una tabla que contenga información de los artículos de unos grandes almacenes

ARTICULO (C-ART, DESC, SECCION, PRECIO, IVA)

```
CREATE TABLE ARTICULOS (
        C-ART           INTEGER NOT NULL,
        DESC            CHAR(20) NOT NULL,
        SECCION         CHAR(10),
        PRECIO          INTEGER,
        IVA             INTEGER,
PRIMARY KEY (C-ART));
```

Se desea incluir un nuevo campo DESCUENTO para indicar el máximo descuento que admite el artículo:

```
ALTER TABLE ARTICULOS ADD (DESCUENTO SMALLINT);
```

Una vez en funcionamiento la base datos, nos damos cuenta que la descripción de algunos artículos es demasiado larga y no tenemos suficiente espacio en 20 caracteres para almacenarla. Modificamos el campo DESC para que almacene descripciones de hasta 40 caracteres.

```
ALTER TABLE ARTICULOS MODIFY DESC CHAR(40);
```

En la tabla, existe un campo IVA que almacena los impuestos a pagar por cada artículo, sin embargo, el IVA es siempre el mismo para todos los artículos. Decidimos eliminar esta columna:

```
ALTER TABLE ARTICULOS DROP IVA;
```

DROP TABLE

Borra tanto la definición de la tabla como su contenido, índices y vistas asociadas. Es decir, destruye tanto los datos contenidos en la tabla como la estructura de la tabla.
Su sintaxis es:

DROP TABLE ARTICULOS;

Introducción

Lenguaje de definición de datos

Lenguaje de manipulación de datos

Lenguaje de manipulación de datos

Permite manipular (insertar, actualizar y borrar) datos de la base de datos, así como consultarlos. Las órdenes son:

INSERT
UPDATE
DELETE
SELECT

INSERT

Permite añadir información en la base de datos. Inserta nuevas filas en la tabla dada. Su formato es el siguiente:

INSERT INTO tabla[(lista-columnas)] VALUES (lista-valores);

Los valores deben corresponder en posición con la lista de columnas. Si lista-columnas no aparece, lista valores estará formado por un conjunto de datos separados por comas que se corresponderán en posición al orden de los campos cuando se creó la tabla. Por el contrario, si lista-columnas contiene algún campo, lista valores debe contener valores que se correspondan en posición con el tipo de datos de los campos.

Lenguaje de manipulación de datos

INSERT INTO PROFESOR (C-PROF, F-CONTR, NOMBRE)
VALUES (345,´23-9-1997´,'Alberto Gutiérrez Paz');

C-PROF	NOMBRE	CATEGORIA	DPTO	F-CONTR
: …	….	….	….	….
345	Alberto Gutiérrez Paz	NULL	NULL	23-09-1997

UPDATE

Modifica valores de determinados campos de filas existentes en una tabla dada. Si se especifica una condición, únicamente actualizará los valores de los campos de las filas que cumplan la condición. En caso contrario, todos los valores del campo en la tabla serán modificados. Su sintaxis es:

```
UPDATE      nombre tabla
            SET campo= valor-campo
            [WHERE condición];
```

Lenguaje de manipulación de datos

Supongamos la tabla siguiente que almacena la información de los empleados de una empresa:

PERSONAL (C-EMP,NOMBRE,PUESTO,SALARIO,DPTO,F-NAC)

La dirección ha decido aumentar el salario a todos los empleados un 5%:

```
UPDATE      PERSONAL
            SET SALARIO= SALARIO*1.05;
```

En una reorganización de la empresa se asciende a director al empleado Juan Sánchez Molina:

```
UPDATE      PERSONAL
            SET PUESTO='Director'
            WHERE NOMBRE= 'Juan Sánchez Molina';
```

CONDICIONES

La cláusula WHERE se utiliza para especificar las filas que se desean recuperar. Esta cláusula está formada por la palabra reservada WHERE seguida de la condición de búsqueda, la cual actúa como filtro para las tuplas de la tabla. Las filas que satisfacen la condición pasan el filtro y forman parte de los resultados de la consulta. Las filas que no cumplen la condición no atraviesan el filtro y quedan excluidas de los resultados.

CONDICIONES DE COMPARACIÓN: permiten comparar el valor de una expresión con el valor de otra. Los operadores de condición utilizados son =,<,>,<=,>=,<>.

WHERE SALARIO<= 1000
WHERE CATEGORIA= 'Ayudante'
WHERE F-CONTR < ´1-10-93´

CONDICIONES DE CORRESPONDENCIA CON PATRÓN
[NOT]LIKE
Comprueba si el valor de una columna de tipo cadena de caracteres se corresponde con un patrón especificado.
Permite comparar cadenas alfanuméricas haciendo uso de símbolos comodín.

_: Sustituye un único carácter
%: Sustituye a varios caracteres

Ej: WHERE CIUDAD LIKE 'Villa%'
WHERE DPTO LIKE '61_'

CONDICIONES DE VALOR NULO (IS [NOT] NULL): comprueba si una columna tiene valores desconocidos. Cuando el valor de una columna, o es desconocido, o no es aplicable, hacemos uso del valor nulo (NULL). Para la selección de valores nulos, se utiliza el operador IS [NOT] NULL.

WHERE CATEGORIA IS NULL

CONDICIÓN DE RANGO (BETWEEN x AND y): Selecciona valores dentro del rango x-y. Examina si el valor de una expresión está dentro de un rango especificado de valores.

WHERE SALARIO BETWEEN 3000 AND 6000
es equivalente a:
WHERE (SALARIO>=3000) AND (SALARIO <=6000)

CONDICIÓN DE PERTENENCIA A UN CONJUTO: Se tienen varios operadores para formar condiciones de conjuntos.
IN Lista-valores: Comprueba si un valor se corresponde con un elemento de un conjunto de valores

WHERE CIUDAD IN ('SALAMANCA', 'ZAMORA', 'LEÓN')

Es equivalente a

WHERE (CIUDAD=´SALAMANCA´) OR (CIUDAD=`ZAMORA´) OR (CIUDAD=´LEÓN´)

CONDICIÓN DE PERTENENCIA A UN CONJUTO:

op ALL lista-valores: Selecciona valores que cumplen la operación op con todos los elementos de la lista de valores. op puede ser <,>, <=,>=, <>.

WHERE CIUDAD <> ALL ('SALAMANCA','ZAMORA','LEÓN')

Es equivalente a:

WHERE NOT ((CIUDAD=´SALAMANCA´) OR
(CIUDAD=´ZAMORA´) OR (CIUDAD=´LEÓN))

ó
WHERE (CIUDAD<>´SALAMANCA´) AND
(CIUDAD<>´ZAMORA') AND (CIUDAD<>´LEÓN´)

CONDICIÓN DE PERTENENCIA A UN CONJUTO:

op ANY lista-valores: Selecciona valores que cumplen la operación op con algún elemento de la lista de valores

WHERE CIUDAD <> ANY ('SALAMANCA','ZAMORA','LEÓN')

Es equivalente a:

WHERE (CIUDAD<>´SALAMANCA´) OR
(CIUDAD<>´ZAMORA´) OR (CIUDAD<>´LEÓN´)

Lenguaje de manipulación de datos

Cada una de las operaciones de comparación pueden ser combinadas haciendo uso de los operadores booleanos:

OR
WHERE (NOMBRE LIKE 'ANTONIO%') OR (C-EMP BETWEEN 30 AND 60)

AND
WHERE (SALARIO >100000) AND (PUESTO IS NULL)

NOT
WHERE NOT (C-EMP=10) OR (C-EMP=12)

Los términos pueden ser agrupados haciendo uso de paréntesis:

WHERE NOT ((C-EMP=10) OR (C-EMP=12)) AND ((NOMBRE LIKE 'ANTONIO%') OR (C-EMP BETWEEN 30 AND 60))

Nótese la diferencia con esta otra expresión:

WHERE NOT (C-EMP =10) OR (C-EMP =12)
AND (NOMBRE LIKE ' ANTONIO%') OR(C-EMP BETWEEN 30 AND 60)

DELETE

Esta orden se utiliza para borrar tuplas de una relación. DELETE borra filas de una tabla especificada. Si no se pone condición, se eliminarán todas las filas de la tabla (no confundir con DROP que además de borrar la información, elimina la estructura). La orden DELETE sin condición, deja la tabla vacía como si se acabara de crear, pero no destruye la definición de la relación. Si se especifica una condición, únicamente se borran los registros que satisfagan dicha condición.

DELETE FROM nombre-tabla [WHERE condición];

DELETE FROM PERSONAL WHERE SALARIO IS NULL;

SELECT

La orden SELECT es la operación fundamental del SQL y se utiliza para la consulta de datos de una o más tablas. Selecciona un conjunto de registros de una o más tablas que cumplan una condición (en caso de no poner una condición, se seleccionan todas las filas de la tabla). Permite también crear un filtro seleccionando solamente algunos campos de la tabla.

```
SELECT      {*|ALL|DISTINCT|lista_columnas|expresiones}
FROM        lista-tablas
[WHERE      condición]
[ORDER BY   columna      orden]
[GROUP BY   columnas]
[HAVING     condición];
```

SELECT

El resultado de una orden SELECT siempre es una tabla.

C-PROF	NOMBRE	CATEGORÍA	DPTO	F-CONTR
344	Ana Osorio Bueno	Asociado	653	20-09-1997
345	Alberto Gutiérrez Paz	NULL	NULL	23-09-1997
346	María Tapia Mora	Ayudante	618	02-02-1998
347	Blas González Rus	NULL	618	15-02-1998
348	Carmen Mª Valdivieso Nuñez	NULL	652	NULL

CONSULTA DE TODAS LAS FILAS Y COLUMNAS DE UNA TABLA

```
SELECT * FROM tabla;

SELECT * FROM PROFESOR;
```

CONSULTA DE TODAS LAS FILAS Y ALGUNAS COLUMNAS DE UNA TABLA

SELECT nombre- columnas FROM nombre-tablas;
Esta operación es equivalente a la operación de proyección en el álgebra relacional.
SELECT C-PROF, NOMBRE FROM PROFESOR;

C-PROF	NOMBRE
344	Ana Osorio Bueno
345	Alberto Gutiérrez Paz
346	María Tapia Mora
347	Blas González Rus
348	Carmen Mª Valdivieso Nuñez

CONSULTA DE DETERMINADAS FILAS DE UNA TABLA

Se hace uso de la orden SELECT de la claúsula WHERE

SELECT * FROM PROFESOR WHERE DPTO LIKE '61_';

C-PROF	NOMBRE	CATEGORÍA	DPTO	F-CONTR
346	María Tapia Mora	Ayudante	618	02-02-1998
347	Blas González Rus	NULL	618	15-02-1998

SELECT * FROM PROFESOR WHERE CATEGORIA IS NOT NULL;

C-PROF	NOMBRE	CATEGORÍA	DPTO	F-CONTR
344	Ana Osorio Bueno	Asociado	653	20-09-1997
346	María Tapia Mora	Ayudante	618	02-02-1998

CONSULTA DE DETERMINADAS FILAS DE UNA TABLA

```
SELECT NOMBRE FROM PROFESOR
WHERE F-CONTR BETWEEN ´1-1-1997´ AND ´31-12-1997´;
```

NOMBRE
Ana Osorio Bueno
Alberto Gutiérrez Paz

```
SELECT *FROM PROFESOR WHERE C-PROF<347 AND
NOMBRE NOT LIKE 'A%'
```

C-PROF	NOMBRE	CATEGORÍA	DPTO	F-CONTR
346	María Tapia Mora	Ayudante	618	02-02-1998

CONSULTA USANDO OPERADORES

En las expresiones SELECT se pueden utilizar expresiones tales como operaciones aritméticas o cadenas de caracteres, pudiéndose utilizar en estas expresiones, paréntesis.

Supongamos una tabla que almacena datos económicos de artículos tales como el precio, el IVA, el % de comisión que se lleva el vendedor:

C_ART	PRECIO	IVA	COMISÓN
3	1000	12	1.0
4	500	7	0.5
5	2000	12	1.0
6	1000	12	0.75

SELECT PRECIO, IVA, PRECIO+PRECIO*IVA/100 FROM ARTICULOS;

PRECIO	IVA	PRECIO+PRECIO*IVA/100
1000	12	1120
500	7	535
2000	12	2240
1000	12	1120

SELECT C_ART, PRECIO, PRECIO*COMISION/100 FROM ARTICULOS WHERE PRECIO*(1+IVA/100)>1000;

C_ART	PRECIO	PRECIO*COMISION/100
3	1000	10
5	2000	20
6	1000	7,5

CONSULTA ORDENADA

Puede especificarse el orden en el que aparecerán las filas en el resultado de una orden SELECT mediante la cláusula ORDER BY. La ordenación puede hacerse por nombre de columna o por la posición de esa columna dentro de la cláusula SELECT. Si no se especifica el tipo de orden (ascendente o descendente), SQL utilizará por defecto el orden ascendente.

SELECT * FROM ARTICULO ORDER BY PRECIO;

C_ART	PRECIO	IVA	COMISION
4	500	7	0.5
3	1000	12	1.0
6	1000	12	0.75
5	2000	12	1.0

CONSULTA ORDENADA

SELECT * FROM ARTICULO ORDER BY PRECIO DESC;

C_ART	PRECIO	IVA	COMISION
5	2000	12	1.0
3	1000	12	1.0
6	1000	12	0.75
4	500	7	0.5

SELECT IVA, C_ART FROM ARTICULO ORDER BY IVA DESC, C_ART ASC;

IVA	C_ART
12	3
12	5
12	6
7	4

CONSULTA CON FUNCIONES AGREGADAS

COUNT: Devuelve el número de filas que cumplen la condición. COUNT (*) incluye los valores nulos en la cuenta. COUNT(DISTINCT columna) devuelve el número de valores distintos, no incluyendo nulos.
COUNT{ (*)|[DISTINCT] columna}

SELECT COUNT (*) FROM ARTICULO;

COUNT (*)
4

SELECT COUNT (DISTINCT IVA) FROM ARTICULO;

COUNT (DISTINCT (IVA))
2

SELECT COUNT (DISTINCT PRECIO) FROM ARTICULO WHERE IVA=12;

COUNT (DISTINCT (PRECIO))
2

SUM (columna/expresión): Devuelve la suma de los valores. La columna o expresión debe ser numérica.
SELECT SUM(PRECIO) FROM ARTICULO WHERE COMISION< 1.0;

SUM (PRECIO)
1500

SELECT SUM (PRECIO*COMISION/100) FROM ARTICULO;

SUM (PRECIO*COMISION/100)
40

AVG (columna/expresión): Devuelve la media de los valores. La columna o expresión debe ser numérica:
SELECT AVG(PRECIO) FROM ARTICULO;

AVG(PRECIO)
1125

MAX (columna/expresión): Devuelve el valor máximo. La columna o expresión debe ser numérica.
SELECT MAX(PRECIO) FROM ARTICULO;

MAX (PRECIO)
2000

MIN (columna/expresión): Devuelve el valor mínimo. La columna o expresión debe ser numérica:
SELECT MIN(PRECIO) FROM ARTICULO;

MIN(PRECIO)
500

CONSULTA DE INFORMACIÓN AGRUPADA.

SQL permite obtener una tabla agrupada que contiene una fila por cada grupo. Esta fila contendrá información resumen sobre el grupo. Para obtener esta información agrupada se hace uso de la cláusula GROUP BY.

C_ART	F_PED	CANT	F_SUM
3	1-3-1997	12	2-3-1997
6	1-3-1997	8	3-3-1997
3	15-8-1997	10	3-9-1997
4	27-2-1997	2	2-3-1997
5	11-12-1997	10	20-12-1997
3	27-2-1997	5	2-3-1997

```
SELECT C_ART, SUM (CANT) FROM PEDIDO
GROUP BY C_ART
ORDER BY C_ART;
```

Devuelve para cada artículo suministrado, su código y la cantidad total suministrada.

C_ART	SUM (CANT)
3	27
4	2
5	10
6	8

De forma análoga a la cláusula WHERE, se pueden establecer restricciones sobre los grupos que aparecerán. Esto se realiza mediante la cláusula HAVING.

```
SELECT C_ART, SUM(CANT) FROM PEDIDO
GROUP BY C_ART
HAVING SUM(CANT) <10;
```

C_ART	SUM (CANT)
6	8
4	2

Este ejemplo nos muestra únicamente aquellos grupos de artículos que han sido servidos en una cantidad total inferior a 10.

CONSULTA USANDO VARIAS TABLAS

Se pueden hacer consultas a más de una tabla de la base de datos, especificando las relaciones entre las distintas tablas en las cláusula WHERE y las tablas que intervienen en la cláusula FROM.

En realidad lo que se hace para consultar datos de más de una tabla es un Join con una condición de igualdad. Para poder especificar la relación, es necesario que ambas tablas tengan un campo en común o al menos que los dos campos que sirven de enlace, sean de idéntico tipo.

CONSULTA USANDO VARIAS TABLAS

Supongamos que disponemos de la tabla ARTICULO y de la tabla PEDIDO.

ARTICULO (C_ART,DESC,SECCION,PRECIO,IVA)

- C_ART: Código del artículo
- DESC: Descripción
- SECCION: Sección
- PRECIO: Precio
- IVA: Iva

PEDIDO(C_ART,F_PED,CANT,F_SUM)

- C_ART: Código del artículo
- F_PED: Fecha de pedido
- CANT: Cantidad
- F_SUM: Fecha de suministro

CONSULTA USANDO VARIAS TABLAS

Se desea comprobar los códigos y el precio de los artículos pedidos en el mes de marzo de 1997.

Para consultar el precio debemos incluir en la cláusula FROM la tabla ARTICULO puesto que es la que contiene este campo.

Pero además, se necesita también la tabla PEDIDO para especificar la condición (F_PED BETWEEN ´1-3-1997´ AND ´31-3-1997´).

```
SELECT ARTICULO.C_ART, ARTICULO.PRECIO,
, PEDIDO.F_PED
FROM ARTICULO, PEDIDO
WHERE PEDIDO.F_PED BETWEEN ´1-3-1997´ AND ´31-3-1997´;
```

Con esto se obtendría el producto cartesiano de las dos tablas, eliminando las filas que no cumplen la condición.

CONSULTA USANDO VARIAS TABLAS

Sin embargo, nosotros no queremos obtener el producto cartesiano sino la yunción o join. Queremos obtener los artículos pedidos durante el mes de marzo que son el 3 y el 6. Es decir, queremos consultar únicamente los artículos cuyo código de artículo coincide en ambas tablas.

La consulta correcta sería la siguiente:

```
SELECT ARTICULO.C_ART, ARTICULO.PRECIO, PEDIDO.F_PED
FROM ARTICULO, PEDIDO
WHERE PEDIDO.F_PED BETWEEN ´1-3-1997´ AND ´31-3-1997´
AND ARTICULO.C_ART=PEDIDO.C_ART;
```

SUBCONSULTAS

Una subconsulta es una consulta dentro de otra. Es decir, un SELECT anidado en la cláusula WHERE de otro SELECT. Esta anidación permite realizar consultas complejas, que no serían posible, o poco eficiente haciendo consultas simples.

SELECT columnas FROM tabla
WHERE columna op (SELECT columna FROM tabla WHERE condición);

El operador op a utilizar dependerá del número de valores que se van a obtener en la subconsulta.

Las subconsultas pueden aparecer en cualquier orden en la que se pueda utilizar la cláusula WHERE (SELECT, UPDATE, DELETE).

Volvamos a la tabla profesor

PROFESOR(C-PROF, NOMBRE, CATEGORIA, DPTO, F-CONT)

C-PROF	NOMBRE	CATEGORÍA	DPTO	F-CONTR
344	Ana Osorio Bueno	Asociado	653	20-09-1997
345	Alberto Gutiérrez Paz	NULL	NULL	23-09-1997
346	María Tapia Mora	Ayudante	618	02-02-1998
347	Blas González Rus	NULL	618	15-02-1998
348	Carmen Mª Valdivieso Nuñez	NULL	652	NULL

Se desea consultar los nombres de los profesores que están adscritos al mismo departamento que el profesor 347.

```
SELECT NOMBRE FROM PROFESOR
WHERE DPTO=(SELECT DPTO FROM PROFESOR WHERE C-PROF=347)
```

NOMBRE
María Tapia Mora
Blas González Rus

Se puede utilizar cualquier nivel de anidación. Por ejemplo si deseamos consultar el precio de los artículos pedidos el mismo día que se pidió el artículo 3.

SELECT CART,PRECIO FROM ARTICULO WHERE C-ART IN (SELECT DISTINCT(C-ART) FROM PEDIDO WHERE F-PED IN (SELECT F-PED FROM PEDIDO WHERE C-ART=3));

Vamos a analizar por partes estas subconsultas. La primera es la que se encuentra en un mayor nivel de anidación.

SELECT F-PED FROM PEDIDO WHERE C-ART=3

SELECT F-PED FROM PEDIDO WHERE C-ART=3

C-ART	F-PED	CANT	F-SUM
3	1-3-1997	12	2-3-1997
6	1-3-1997	8	3-3-1997
3	15-8-1997	10	3-9-1997
4	27-2-1997	2	2-3-1997
5	11-2-1997	10	20-12-1997
3	27-2-1997	5	2-3-1997

Por lo tanto en la subconsulta se obtiene un conjunto de 3 fechas (las fechas marcadas en la tabla).

La siguiente subconsulta se convierte en:

SELECT DISTINCT(C-ART) FROM PEDIDO
WHERE F-PED IN ({1-3-1997},{15-8-1997},{27,2-1997})

C-ART	F-PED	CANT	F-SUM
3	1-3-1997	12	2-3-1997
6	1-3-1997	8	3-3-1997
3	15-8-1997	10	3-9-1997
4	27-2-1997	2	2-3-1997
5	11-2-1997	10	20-12-1997
3	27-2-1997	5	2-3-1997

Es decir, se tienen 3 códigos de artículo (3,6,4).

Sólo falta la primera consulta:

SELECT PRECIO FROM ARTICULO WHERE C-ART IN (3,6,4)

El resultado será el precio de los artículos 3,4 y 6:

C-ART	PRECIO
3	1000
4	500
6	1000

Si la tabla artículos era esta:

C-ART	F-PED	CANT	F-SUM
3	1-3-1997	12	2-3-1997
6	1-3-1997	8	3-3-1997
3	15-8-1997	10	3-9-1997
4	27-2-1997	2	2-3-1997
5	11-2-1997	10	20-12-1997
3	27-2-1997	5	2-3-1997

Obtenemos:

C-ART	PRECIO
3	1000
4	500
6	1000

En general se tienen en cuenta una serie de normas a la hora de especificar subconsultas:

- Las subconsultas deben generar una única columna.
- La subconsulta debe estar limitada por paréntesis
- Las subconsultas que generen una única fila se pueden usar tanto con operadores simples como con operadores multivaluados (que utilizan más de un valor, por ejemplo IN).
- Las subconsultas que produzcan más de una fila se pueden usar sólo con operadores multivaluados.
- El operador BETWEEN no se puede utilizar con una subconsulta.

CONSULTA USANDO INNER JOIN

Inner Join es otro tipo de composición de tablas, permite emparejar filas de distintas tablas de forma más eficiente que en el producto cartesiano cuando una de las columnas de emparejamiento está indexada. Ya que en vez de hacer el producto cartesiano completo y luego seleccionar las filas que cumplen la condición de emparejamiento, para cada fila de unas de las tablas, busca directamente en la otra tabla las filas que cumplen la condición, con lo cual, se emparejan sólo las filas que luego aparecen en el resultado.

FROM tabla1 INNER JOIN tabla2 ON tabla1.col1 comp tabla 2.col2

CONSULTA USANDO INNER JOIN

FROM tabla1 INNER JOIN tabla2 ON tabla1.col1 comp tabla 2.col2

- tabla1 y tabla2 son especificaciones de tabla
- col1 y col2 son las columnas de emparejamiento
- Dentro de la cláusula ON los nombres de columna deben ser nombres cualificados (lleva delante del nombre de la tabla y un punto).
- Las columnas de emparejamiento deben contener la misma clase de datos.
- comp representa cualquier operador de comparación (=,<,>,<=,>=,<>) y se utiliza para establecer la condición de emparejamiento.

CONSULTA USANDO INNER JOIN

Ejemplo:

SELECT * FROM Pedidos INNER JOIN Clientes ON Pedidos.clie= Clientes.numclie

CONSULTA USANDO INNER JOIN

Se pueden definir varias condiciones de emparejamiento unidas por los operadores AND y OR poniendo cada condición entre paréntesis.

```
SELECT
FROM Pedidos INNER JOIN Productos ON
(Pedidos.fab=Productos.idfab)
AND (Pedidos.producto=Productos.idproducto)
```

CONSULTA USANDO INNER JOIN

Se pueden combinar más de dos tablas
En este caso hay que sustituir en la sintaxis una tabla por un INNER JOIN completo

```
SELECT *
FROM (Pedidos INNER JOIN Clientes ON Pedidos.clie=Clientes.numclie) INNER JOIN Empleados ON Pedidos.rep=Empleados.numemp
```

En vez de la tabla1 hemos escrito un INNER JOIN completo.

CONSULTA USANDO INNER JOIN

También podemos escribir:

```
SELECT *
FROM Clientes INNER JOIN (Pedidos INNER JOIN Empleados ON Pedidos.rep=Empleados.numemp) ON Pedidos.clie=Clientes.numclie
```

En este caso hemos sustituido tabla2 por un INNER JOIN completo.

VARIABLES DE USUARIO (I)

Permiten almacenar un valor y hacer referencia a él, más tarde posibilitando pasar valores de una sentencia a otra. Las variables de usuario son específicas de la conexión, lo que significa que una variable definida por un usuario, no puede ser vista por otros usuarios.

Todas las variables de un usuario son automáticamente liberadas cuando éste abandona la conexión.

SET **@nombre de variable** = expresión;

SELECT **@nombre de variable**;

VARIABLES DE USUARIO (II)

Ej: Añadir 50 euros de prima a todos aquellos empleados con un sueldo menor que la media

NOMBRE	SUELDO	DEPARTAMENTO	PRIMAS
Ana	1700.50	RRHH2	0.00
Víctor	2400.00	INF5	10.00
Fermin	1400.99	INF2	41.03
Olga	900.50	ING3	0.00
Jesús	3521.52	ERE1	50.00

VARIABLES DE USUARIO (III)

Ej: Añadir 50 euros de prima a todos aquellos empleados con un sueldo menor que la media

SET **@SUELDOMEDIO**= SELECT (AVG(SUELDO) FROM TABLA;

SUELDOMEDIO=1984.702000

VARIABLES DE USUARIO (IV)

NOMBRE	SUELDO	DEPARTAMENTO	PRIMAS
Ana	1700.50	RRHH2	0.00
Víctor	2400.00	INF5	10.00
Fermin	1400.99	INF2	41.03
Olga	900.50	ING3	0.00
Jesús	3521.52	ERE1	50.00

```
SELECT SUELDOMEDIO= AVG (SUELDO)
FROM TABLA
```

```
SUELDOMEDIO=1984.702000
```

```
SET @SUELDOMEDIO= SELECT AVG (SUELDO) FROM TABLA;
```

```
UPDATE TABLA= SET
PRIMAS= (PRIMAS +50.00)
WHERE
SUELDO< SELECT (@SUELDOMEDIO);
```

=

```
UPDATE TABLA=
SET PRIMAS= (PRIMAS +50.00)
WHERE
SUELDO<
( SELECT (AVG(SUELDO)FROM TABLA);
```

VARIABLES DE USUARIO (V)

NOMBRE	SUELDO	DEPARTAMENTO	PRIMAS
Ana	1700.50	RRHH2	50.00
Víctor	2400.00	INF5	10.00
Fermin	1400.99	INF2	91.03
Olga	900.50	ING3	50.00
Jesús	3521.52	ERE1	50.00

VARIABLES DE USUARIO (VI)

Ej2: Añadir 20 euros de prima a todos aquellos empleados con un sueldo menor que la media, pero sólo a aquellos que tienen primas asignadas mayores que la media

NOMBRE	SUELDO	DEPARTAMENTO	PRIMAS
Ana	1700.50	RRHH2	50.00
Víctor	2400.00	INF5	10.00
Fermin	1400.99	INF2	91.03
Olga	900.50	ING3	50.00
Jesús	3521.52	ERE1	50.00

SET **@PRIMAS**=(SELECT AVG(PRIMAS) FROM TABLA;
SET **@SUELDOMEDIO**= (SELECT AVG(SUELDO) FROM TABLA WHERE PRIMAS>(SELECT **@PRIMAS**)

UPDATE TABLA SET **PRIMAS**= (PRIMAS+20.00) WHERE SUELDO< (SELECT **@SUELDOMEDIO**)

OPTIMIZACIÓN DE CONSULTAS

Optimización de consultas

Introducción

Utilización de Heurísticas

Reglas generales de transformación para operaciones de álgebra relacional

Etapas en el proceso de optimización

Optimización de las operaciones algebraicas

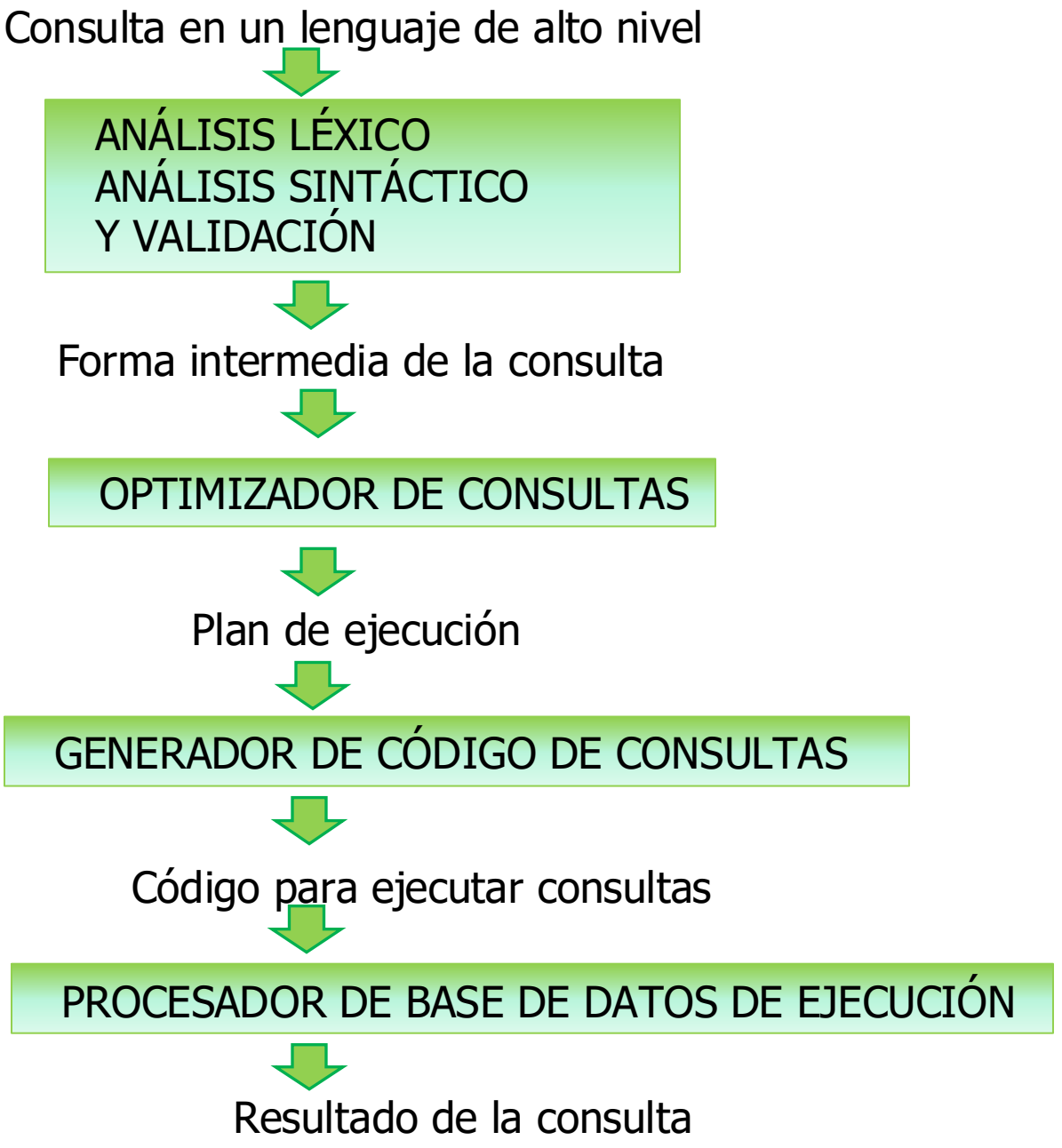
Introducción

- En este tema veremos las técnicas con las que los SGBD procesan, optimizan y ejecutan las consultas de alto nivel.

- Para expresar una consulta en un lenguaje de alto nivel como SQL:
  - Primero debe pasar por un **análisis léxico** que identifica los símbolos o tokens del lenguaje (como las palabras clave de SQL, los nombres de los atributos y de las relaciones) en el texto de la consulta.
  - Después un **análisis sintáctico** que revisa la sintaxis para determinar si está formulada de acuerdo con las reglas sintácticas del lenguaje de consulta.
  - Por último la consulta debe ser **validada** para lo que ha de comprobarse que tanto los nombres de las relaciones como los atributos son válidos y tengan sentido desde el punto de vista semántico en el esquema de la B.D..

- A continuación se crea una representación interna de la consulta, por lo regular en forma de árbol o grafo (**árbol o grafo de consultas**).

- Lo siguiente que debe hacer el SGBD es crear una estrategia de ejecución para obtener el resultado de la consulta a partir de los archivos internos. El proceso de elegir la alternativa más adecuada para procesar una consulta se denomina **optimización de consultas.**



Pasos en el procesamiento de una consulta de alto nivel

- El módulo **optimizador de consultas** se encarga de producir un plan de ejecución.

- El **generador de código** genera el código necesario para ejecutarlo.

- El **procesador de base de datos** en tiempo de ejecución se encarga de ejecutar el código de la consulta, que será compilado o interpretado, para producir el resultado de la consulta.

Si se presenta un error durante la ejecución, este procesador genera un mensaje de error.

Existen dos técnicas principales para ejecutar consultas:

- La primera se basa en **reglas heurísticas** para ordenar las operaciones en una estrategia de ejecución de una consulta (reordenar las operaciones en un árbol o grafo de consulta). Una regla heurística es una regla que funciona bien en la mayoría de los casos pero que no garantiza que funcione bien en todos los casos posibles.

- La segunda técnica implica estimar sistemáticamente **el costo de diferentes estrategias de ejecución y elegir el plan de menor coste.**

Optimización de consultas

Introducción

Utilización de Heurísticas

Reglas generales de transformación para operaciones de álgebra relacional

Etapas en el proceso de optimización

Optimización de las operaciones algebraicas

- **El objetivo principal** es modificar la estructura de datos árbol que representa la consulta internamente a fin de mejorar el rendimiento esperado durante la ejecución.

- La principal regla heurística se basa en **aplicar las operaciones seleccionar y proyectar antes de aplicar la reunión y otras operaciones binarias.**

- El motivo de hacer esto es porque la selección y la proyección casi siempre reducen el tamaño de un fichero y nunca lo aumentan; por lo que conviene aplicarlas antes de una reunión o de cualquier operación binaria.

Utilización de la heurística en la optimización de consultas

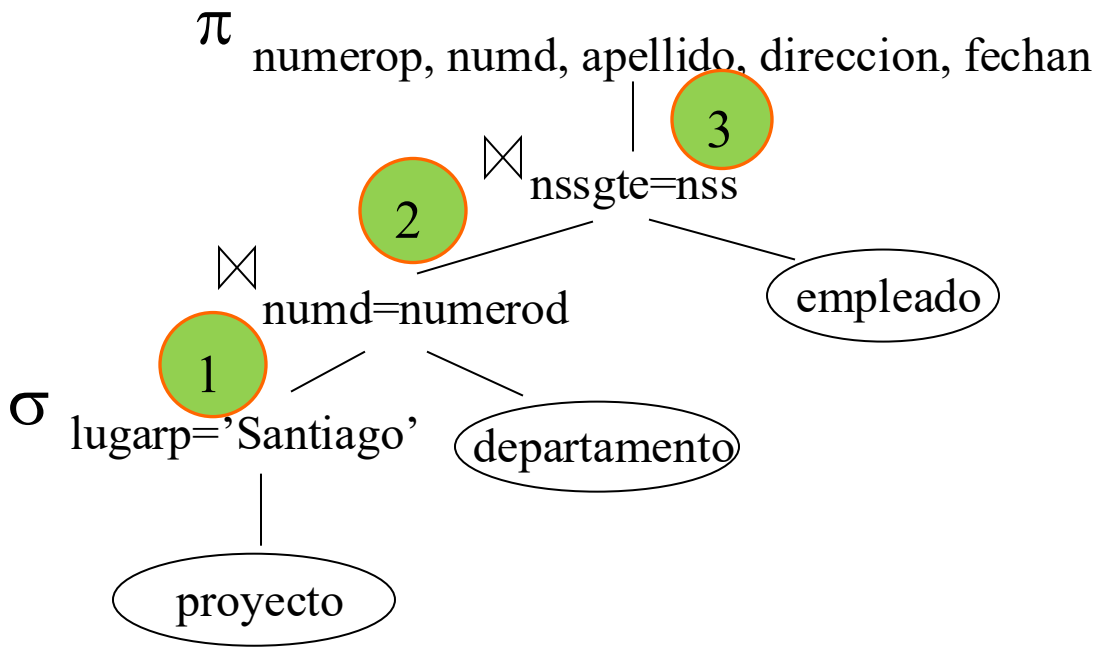
- Para poder aplicar la técnica heurística es **necesario crear un árbol de consulta**
  - Es una estructura de árbol que corresponde a una expresión del álgebra relacional.
  - Las tablas se representan como nodos hojas.
  - Las operaciones del álgebra relacional como nodos intermedios.
- Una ejecución del árbol de consulta consiste en la ejecución de una operación de un nodo interno siempre que sus operandos estén disponibles, sustituyendo después ese nodo interno por la tabla que resulte de ejecutar la operación. La ejecución termina cuando se ejecuta el nodo raíz y produce la tabla resultado para la consulta.
- La optimización heurística transforma el árbol de consulta empleando un conjunto de reglas que generalmente mejoran el rendimiento de la ejecución.

Utilización de la heurística en la optimización de consultas

- Por ejemplo, para la consulta:

SELECT numerop, numd, apellido, dirección, fechan
FROM proyecto, departamento, empleado
WHERE numd=numerod AND nssgte=nss AND
lugarp='Santiago';

un árbol de consulta sería:



- En la figura se representa tres relaciones proyecto, departamento y empleado.

- Las tres tablas son los nodos hoja.

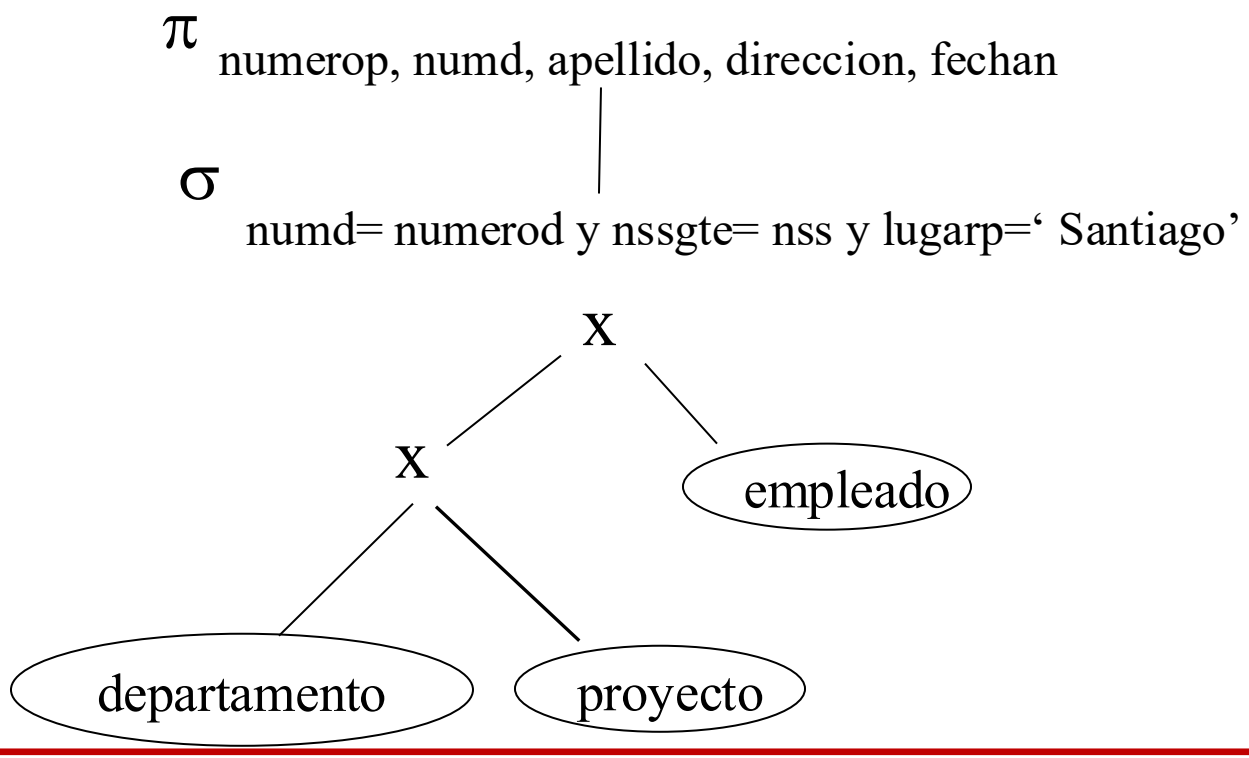
- Las operaciones del álgebra relacional de la expresión se representan en los nodos internos del árbol.

- Cuando se ejecuta el árbol de consulta, el nodo numerado con (1) debe comenzar su ejecución antes del nodo (2) porque las tuplas deben estar disponibles antes de poder ejecutar la operación (2). De manera similar, el nodo (2) debe ejecutarse antes del nodo (3), y así sucesivamente.

- Muchos árboles de consulta pueden ser equivalentes (es decir, producir idéntico resultado) por lo que hay que saber cual escoger.

- Lo que el optimizador de consultas suele hacer es crear un árbol de consulta inicial que corresponde a una consulta sin efectuar optimización alguna que resulta, por tanto, muy ineficiente.

- Por ejemplo, la figura siguiente representa el árbol inicial de la consulta anterior. Como se puede ver, se realiza el producto cartesiano de las tablas, obteniendo todas las combinaciones posibles de sus tuplas, a continuación se seleccionan aquellas tuplas que cumplen con todas las condiciones de la consulta y, finalmente, se proyectan sólo aquellos atributos que debe devolver la consulta.



Utilización de la heurística en la optimización de consultas

- El optimizador de consultas modifica este árbol inicial hasta un árbol de consulta final cuya ejecución sea eficiente. Para poder realizar esta transformación el optimizador debe de aplicar una serie de reglas de equivalencia entre expresiones del álgebra relacional.

- Vamos a ver un ejemplo de transformación.

  Supongamos la siguiente consulta que busca los apellidos de los empleados nacidos después de 1957 que trabajan en un proyecto llamado Acuario:

```
SELECT apellido
FROM empleado, trabaja_en, proyecto
WHERE nombrepr='Acuario' AND numerop=nump
AND nsse=nss AND fecha>'31-dic-1957';
```

Utilización de la heurística en la optimización de consultas

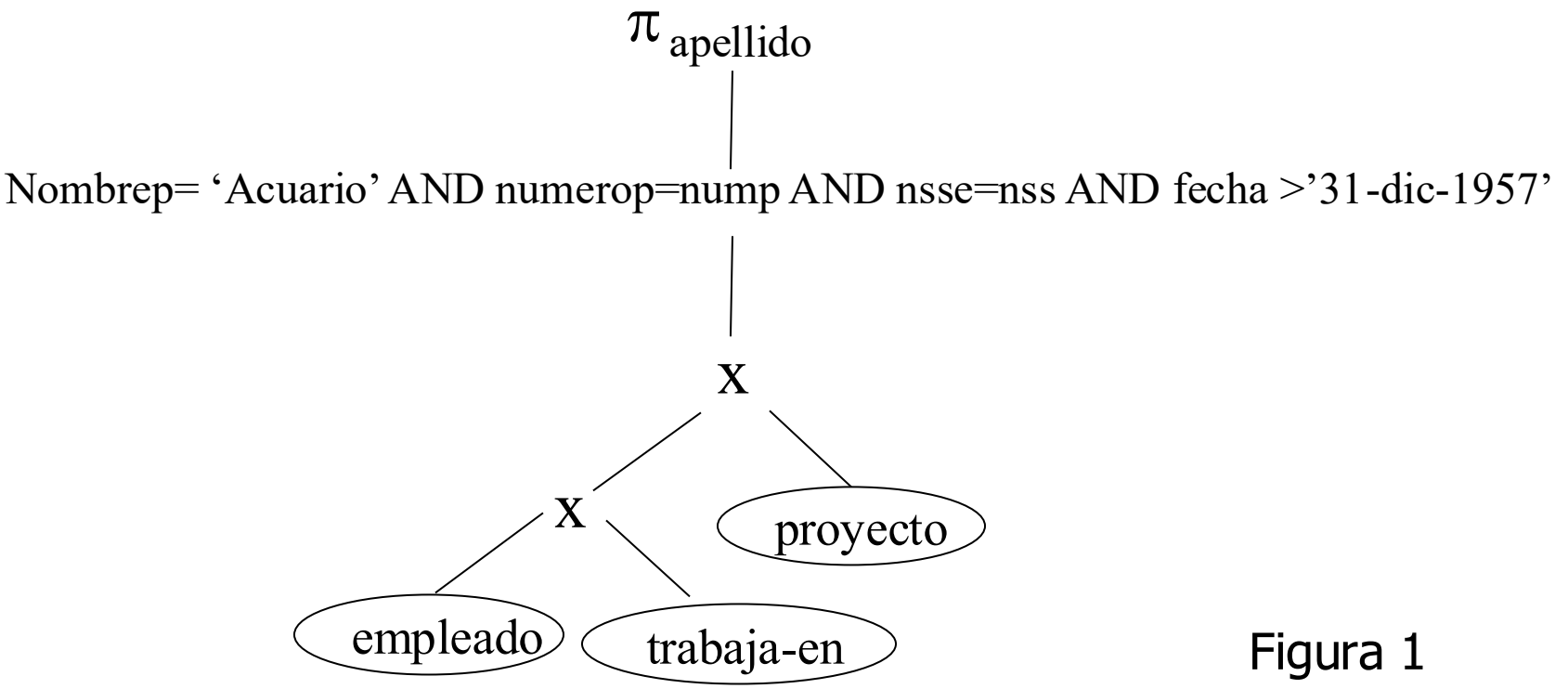


Figura 1

- La ejecución directa de este árbol crea un archivo muy grande que contiene el producto cartesiano de las tres tablas completas. Sin embargo sólo necesitamos un registro de Proyecto (el proyecto "Acuario") y sólo los registros Empleado en los que la fecha de nacimiento sea posterior a 31-dic-1957.

Utilización de la heurística en la optimización de consultas

- Podemos conseguir un árbol de consulta mejorado con sólo aplicar primero las operaciones de selección, reduciendo de esta forma el número de tuplas a las que se aplica el producto cartesiano.

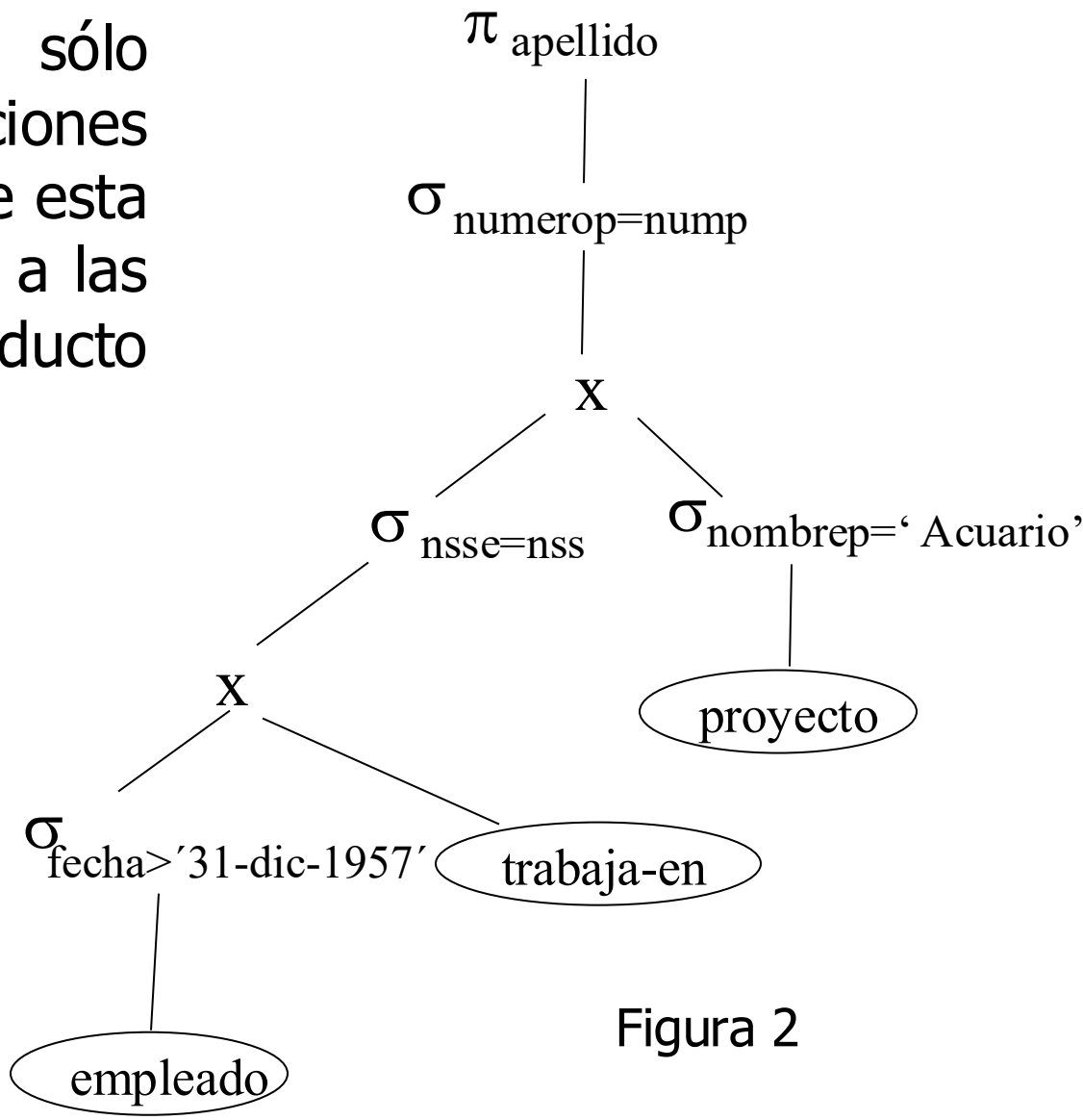


Figura 2

Utilización de la heurística en la optimización de consultas

- Aún es posible mejorar este árbol de ejecución si intercambiamos las posiciones de las tablas empleado y proyecto en el árbol, ya que de esta forma aprovechamos la información de que numerop es clave de la tabla proyecto y, por tanto, la operación de selección sobre la relación proyecto obtendrá un sólo registro.

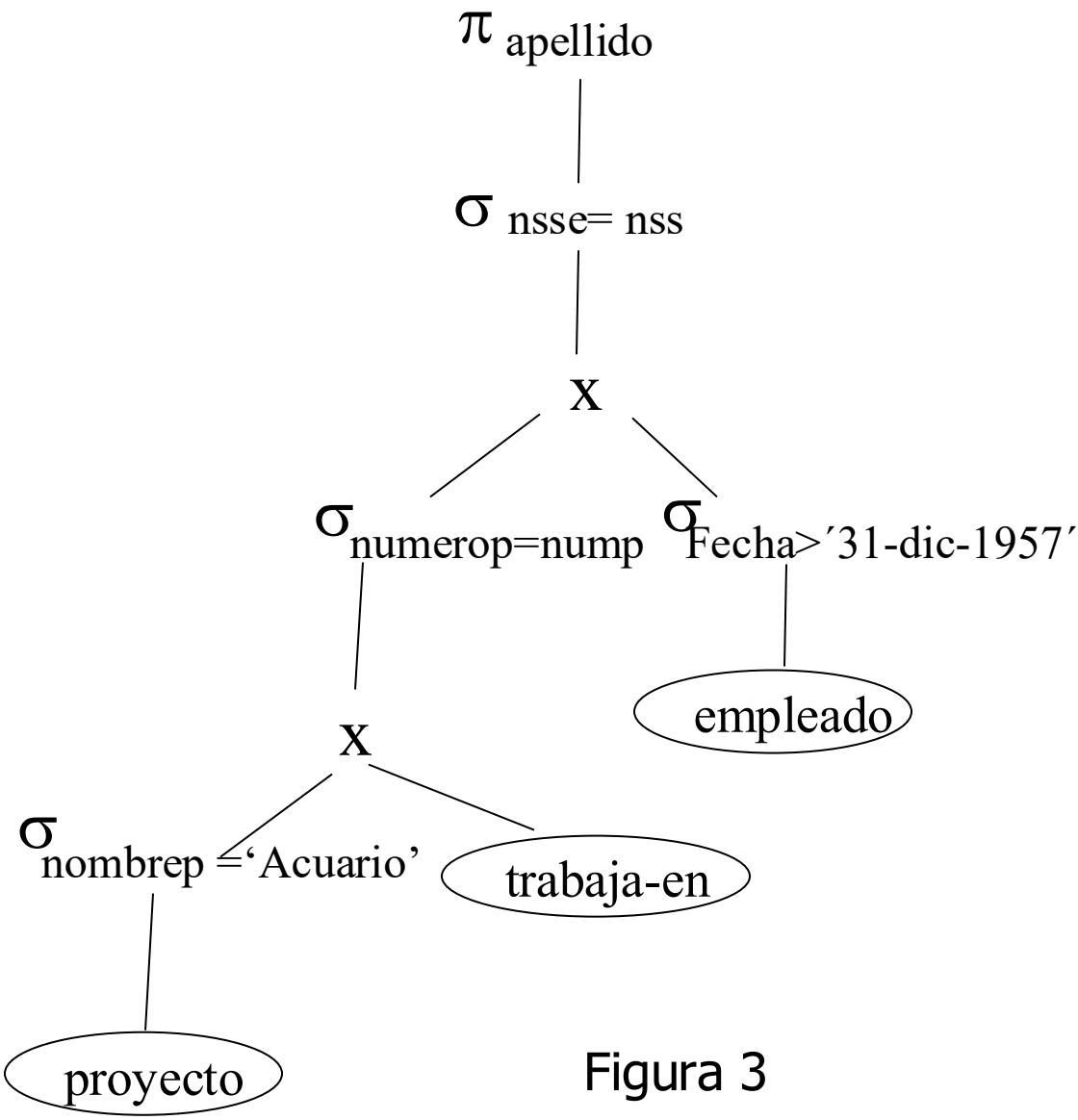


Figura 3

- Todavía mejora más, si reemplazamos toda operación de producto cartesiano seguida de una condición de reunión por una operación de reunión:

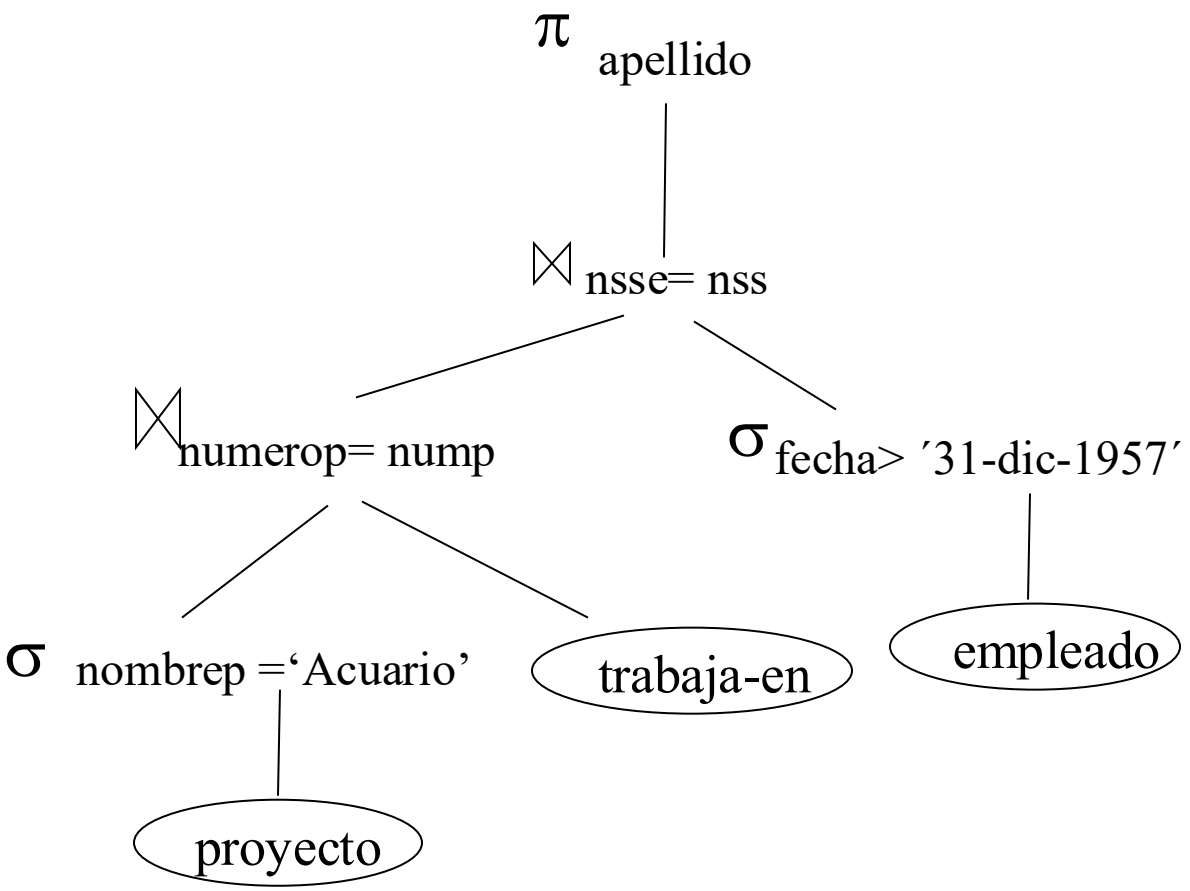


Figura 4

Utilización de la heurística en la optimización de consultas

- Otra mejora consiste en conservar en las relaciones intermedias temporales sólo los atributos requeridos por las operaciones siguientes, incluyendo las operaciones de proyectar lo antes posible en el árbol de consulta. Esto reduce el número de atributos (columnas) de las relaciones intermedias temporales, mientras que las operaciones seleccionar sólo reducen el número de tuplas (registros):

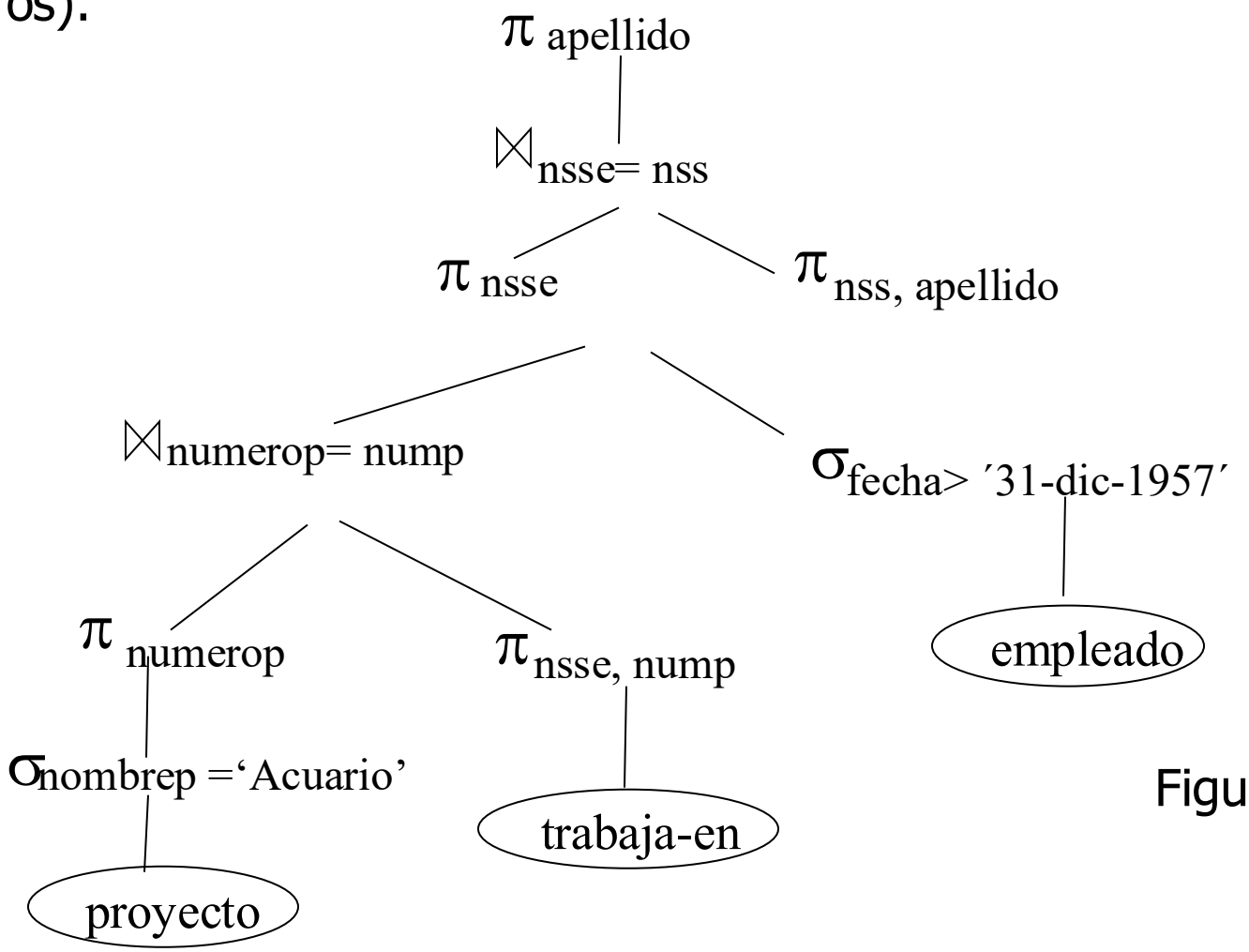


Figura 5

Utilización de la heurística en la optimización de consultas

- Resulta, por tanto, evidente que es posible transformar un árbol de consulta paso a paso en otro árbol cuya ejecución sea más eficiente.

- Sin embargo, hay que estar seguro de que las transformaciones que se realizan lleven a un árbol de consulta equivalente.

- Para ello, el optimizador de consultas debe saber qué reglas de transformación preservan esta equivalencia.

Optimización de consultas

Introducción

Utilización de Heurísticas

Reglas generales de transformación para operaciones de álgebra relacional

Etapas en el proceso de optimización

Optimización de las operaciones algebraicas

1. Cascada de σ. Una condición de selección conjuntiva puede descomponerse en una cascada (secuencia) de operaciones individuales.

$$\sigma_{c1\ \text{AND}\ c2\ \text{AND}\ \ldots\ \text{AND}\ cn}(R) = \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$$

2. Conmutatividad de σ. La operación σ es conmutativa:

$$\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$$

3. Cascada de π. En una cascada (secuencia) de operaciones π, podemos ignorar todas menos la última:

$$\pi_{\text{Lista1}}(\pi_{\text{Lista2}}(\ldots(\pi_{\text{Listan}}(R))\ldots)) = \pi_{\text{Lista1}}(R)$$

4. Conmutación de σ con π. Si en la condición de selección c intervienen sólo los atributos A1, ...,An de la lista de proyección, se pueden conmutar ambas operaciones:

$$\pi_{A1, .., An}(\sigma_{c}(R)) = \sigma_{c}(\pi_{A1, ..., An}(R))$$

5. Conmutatividad de $\bowtie$ (o de X). La operación $\bowtie$ es conmutativa, como lo es la operación X:

$$R \bowtie_{c} S = S \bowtie_{c} R \qquad \text{o} \qquad R \ X \ S = S \ X \ R$$

6. Conmutación de σ con $\bowtie$ (o X) . Si todos los atributos de la condición de selección c pertenecen a sólo una de las relaciones por reunir digamos, R, las dos operaciones pueden conmutarse como sigue:

$$\sigma_c(R \bowtie S) = (\sigma_c(R)) \bowtie S$$

O bien, si la condición de selección c puede escribirse como $(c1$ AND $c2)$, y en la condición $c1$ intervienen sólo atributos de R y en la condición $c2$ intervienen sólo atributos de S, las operaciones se conmutan como sigue:

$$\sigma_c(R \bowtie S) = \sigma_{c1}(R) \bowtie \sigma_{c2}(S)$$

7. Conmutación de π con $\bowtie$ (o X) . Supongamos que la lista de proyección es $L = \{ A_1,\ldots,A_n, B_1,\ldots,B_m\}$ donde $A_1,\ldots,A_n$ son atributos de R y $B_1,\ldots,B_m$ son atributos de S. Si en la condición de reunión c intervienen sólo atributos comprendidos en L, las dos operaciones se pueden conmutar como:

$$\pi_L(R \bowtie_c S) = (\pi_{A1,\ldots,An}(R)) \bowtie_c (\pi_{B1,\ldots,Bm}(S))$$

Si la condición de reunión c contiene atributos adicionales que no están en L, estos deben añadirse a la lista de proyeccción, para lo que se requiere una operación π final.

Por ejemplo, si los atributos $A_{n+1},\ldots,A_{n+k}$ de R y $B_{m+1},\ldots,B_{m+p}$ de S intervienen en la condición de reunión c , pero no están en la lista de proyección L, las operaciones se conmutan como sigue:

$$\pi_L(R \bowtie_c S) = \pi_L(\pi_{A1,\ldots,Am,\ An+1,\ldots,An+k}(R)) \bowtie_c (\pi_{B1,\ldots,Bm,\ Bm+1,\ldots,Bm+p}(S))$$

8. Conmutatividad de operaciones de conjuntos : las operaciones de conjuntos $\cup$ y $\cap$ son conmutativas, pero – (la operación de diferencia) no lo es.

9. Asociatividad de $\bowtie$, X, $\cup$ y $\cap$.
Estas cuatro operaciones son asociativas individualmente:
es decir, si Θ representa cualquiera de estas cuatro operaciones (pero la misma en toda la expresión), tenemos:

$$(R \Theta S) \Theta T = R \Theta (S \Theta T)$$

10. Conmutación de σ con operaciones de conjuntos: la operación σ se conmuta con $\cup$, $\cap$ y - . Si Θ representa cualquiera de estas cuatro operaciones (pero la misma en toda la expresión), tenemos:

$$\sigma_c (R \Theta S) = (\sigma_c (R)) \Theta (\sigma_c (S))$$

11. La operación π se conmuta con $\cup$:

$$\pi_L (R \cup S) = (\pi_L (R)) \cup (\pi_L (S))$$

12. Conversión de una secuencia (σ, X) a $\bowtie$: si la condición c de una operación σ seguida de una X corresponde a una condición de reunión, convertir la secuencia (σ, X) en $\bowtie$ de la manera siguiente:

$$(\sigma_c (R \times S)) = (R \bowtie_c S)$$

Optimización de consultas

Introducción

Utilización de Heurísticas

Reglas generales de transformación para operaciones de álgebra relacional

Etapas en el proceso de optimización

Optimización de las operaciones algebraicas

a) Conversión a formato interno
b) Conversión a formato canónico
c) Creación de un conjunto de planes de ejecución y elección del de menor coste

- **Conversión a formato interno**

Etapa previa al procesamiento de la consulta en la que el sistema debe traducirla a una forma que pueda manejar.

- Se basa en el Álgebra Relacional y el proceso de traducción es similar al analizador sintáctico de un compilador:

- Primero debe pasar por un análisis léxico que identifica los componentes del lenguaje.
- Después un análisis sintáctico para la revisión de la sintaxis.
- Se puede corregir la consulta (comprobación de nombres de las relaciones, atributos,...) o buscar otras equivalentes manipulando el/los predicado/s de selección (optimización sintáctica).
- Puede también realizarse optimización semántica utilizando las condiciones de integridad.

Finalmente se crea una representación interna de la consulta, por lo general en forma de árbol o grafo (árbol o grafo de consultas).

- **Conversión a forma canónica**

Consiste en encontrar una expresión equivalente a la dada que pueda ejecutarse de manera más eficiente: **forma canónica.**

- Necesario ya que el desempeño del sistema no debe depender de cómo el usuario haya decidido expresar la consulta.
- Esta etapa se basa en las propiedades de los operadores algebraicos y se aplican las técnicas de optimización de estos operadores.
- Se intenta reducir el resultado de los tamaños intermedios.

- **Construir planes de ejecución y elegir uno**

En esta etapa el optimizador debe decidir cómo evaluar la forma canónica equivalente de la etapa anterior.

- Se consideran la existencia de índices u otras rutas de acceso, la distribución de los valores de los datos almacenados, el agrupamiento físico de los registros, etc.
- Plan de ejecución: estrategia detallada para procesar una consulta. Incluirá:
    a) Las operaciones relacionales que se van a ejecutar.
    b) Los caminos de acceso que se van a emplear.
    c) El orden en que se van a realizar las operaciones.
- El optimizador dispondrá de un conjunto de procedimientos para procesar cada una de las operaciones de bajo nivel, cada procedimiento estará asociado a una medida de coste y se analizará el coste respectivo de acuerdo a diversas suposiciones. Esto dará lugar a un conjunto de planes de ejecución.

- **Construir planes de ejecución y elegir uno**

En general no será conveniente generar todos los planes posibles, pues la tarea de elegir el más económico puede ser costosa por sí sola: reducción del espacio de búsqueda.

- La elección del plan más económico requiere la estimación del coste de cada uno de ellos, esto último depende de:
    a) El tamaño de los resultados intermedios de todas las operaciones consideradas.
    b) Caminos de acceso.

- **Optimización semántica**

Comienza con una optimización de carácter sintáctico cuya validez es independiente de la base de datos.

- Consiste en transformar la sentencia original utilizando propiedades semánticas conocidas de las entidades que intervienen (condiciones de integridad definidas sobre una relación).

- A cada predicado P de una consulta puede añadirse un número arbitrario de condiciones de integridad C_1, C_2, ..., C_n y formar un nuevo predicado:

$$P \text{ and } C_1 \text{ and } C_2 \text{ and } \ldots \text{ and } C_n$$

- Esta operación no cambia el resultado de la consulta pues, por definición, todas las restricciones de integridad siempre son ciertas.

- Simplificación semántica si la consulta resultante tiene menos términos o menos variables libres que el predicado original.

- **Optimización semántica**
- Ejemplos:

- Si la consulta tiene un subpredicado P que puede deducirse directamente de las restricciones de integridad, P es redundante y puede omitirse.

- Si el predicado P es contradictorio con las condiciones de integridad, el resultado de la consulta será el conjunto vacío. Sin necesidad de acceder a la base de datos.

- Objetivo: **reducir el espacio de búsqueda transformando el predicado de la consulta para facilitar su evaluación**.

Optimización de consultas

Introducción

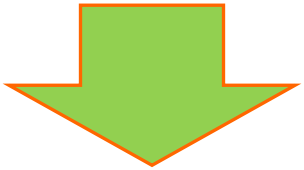
Utilización de Heurísticas

Reglas generales de transformación para operaciones de álgebra relacional

Etapas en el proceso de optimización

Optimización de las operaciones algebraicas

ÁRBOL DE CONSULTA INICIAL



ÁRBOL OPTIMIZADO CUANDO LA EJECUCIÓN SEA MÁS EFICIENTE

1.- Empleando la regla 1, descomponer cualquier operación SELECCIONAR que tenga condiciones conjuntivas en una cascada de operaciones SELECCIONAR. Esto permite desplazar hacia abajo más libremente las operaciones de selección por las diferentes ramas del árbol.

2.- Empleando las reglas 2, 4, 6 y 10, de conmutatividad de SELECCIONAR con otras operaciones, desplazar cada operación de selección tan abajo en el árbol de consulta como lo permitan los atributos que intervengan en la condición de selección.

3.- Empleando las reglas 5 y 9 de conmutatividad y asociatividad de las operaciones binarias, reacomodar los nodos hoja del árbol, usando los criterios siguientes:

- Posicionar las relaciones de los nodos hoja con las operaciones SELECCIONAR más restrictivas (aquellas operaciones que producen una relación con el menor número de tuplas) para que sean ejecutados primero en la representación del árbol de consultas.

- Cerciorarse de que el orden de los nodos hoja no da lugar a operaciones de producto cartesiano, por ejemplo si las dos relaciones con el SELECCIONAR más restrictivo no tienen una condición de reunión directa entre ellas, puede ser preferible cambiar el orden de los nodos hoja para evitar productos cartesianos.

4.- Empleando la regla 12, combinar una operación de PRODUCTO CARTESIANO con una operación SELECCIONAR posterior en el árbol, convirtiéndolas en una operación REUNIÓN, si la condición representa una condición de reunión.

5.- Empleando las reglas 3, 4, 7 y 11, de cascada de PROYECTAR y de conmutación de PROYECTAR con otras operaciones, descomponer las listas de atributos de proyección y desplazarlas lo más abajo posible en el árbol creando nuevas operaciones PROYECTAR según sea necesario. Después de cada operación de PROYECTAR, sólo deberían guardarse aquellos atributos que se necesiten en el resultado de la consulta y en operaciones posteriores en el árbol de consulta.

6.- Identificar subárboles que representen grupos de operaciones que se pueden ejecutar con un solo algoritmo.

- En el ejemplo visto, la Figura 2 muestra el árbol de la Figura 1 después de aplicar los pasos 1 y 2 del algoritmo; la Figura 3 muestra el árbol después de la aplicación del paso 3; la Figura 4 después del paso 4 y la Figura 5 después del paso 5.

- En el paso 6 podríamos agrupar en un solo algoritmo las operaciones del subárbol cuya raíz es la operación π_{nsse}

- También podemos agrupar la operaciones restantes en otro subárbol, donde las tuplas que resultan del primer algoritmo sustituyen al subárbol cuya raíz es la operación π_{nsse} porque la primera agrupación significa que este subárbol se ejecutará primero.

Resumen de la heurística para la optimización algebraica.

- Aplicar primero las operaciones que reducen el tamaño de los resultados intermedios. En esto entra la ejecución de la operaciones SELECCIONAR tan pronto como sea posible para reducir el número de tuplas, y la ejecución de las operaciones PROYECTAR tan pronto como sea posible para reducir el número de atributos.
- Esto se logra desplazando las operaciones SELECCIONAR y PROYECTAR lo más abajo que se pueda en el árbol.
- Además, se deben ejecutar las operaciones SELECCIONAR y REUNIÓN más restrictivas (es decir, las que producen relaciones con el menor número de tuplas o los nodos o el tamaño absoluto más pequeño) antes que otras operaciones similares. Para ello se reacomodan los nodos hoja del árbol entre sí al tiempo que se evitan los productos cartesianos y se ajusta el árbol según sea apropiado.

- Propiedades de los operadores algebraicos:

1.- Conmutatividad del JOIN.
2.- Asociatividad del JOIN.
3.- Reagrupamiento de las selecciones.
4.- Conmutatividad entre selección y proyección.
5.- Conmutatividad entre selección y JOIN.
6.- Conmutatividad entre selección y unión.
7.- Conmutatividad entre selección y diferencia.
8.- Conmutatividad entre proyección y JOIN.
9.- Conmutatividad entre proyección y unión.
10.- Conmutatividad entre proyección y producto cartesiano.

- Algoritmo de optimización:

1. Separar las selecciones que comparten varios predicados.
   Aplicamos la propiedad 3: reagrupamiento de selecciones.

2. Gestionar cada selección en el nivel más bajo posible.
   Aplicamos las propiedades 5, 6 y 7: Conmutatividad entre selección y JOIN, selección y unión, y selección y diferencia.

3. Reagrupar las selecciones sucesivas que se aplican sobre una misma relación aplicando de nuevo la regla de reagrupamiento de selecciones.
   Aplicamos la propiedad 3: Reagrupamiento de selecciones.

Algoritmo de optimización

4. Desplazar las proyecciones al nivel más bajo posible. *Aplicamos las propiedades 4, 8, 9 y 10: Conmutatividad entre selección y proyección, proyección y producto cartesiano y; selección y unión.*

5. Reagrupar proyecciones sucesivas conservando los atributos restantes y eliminando las proyecciones inútiles que hayan podido aparecer (proyecciones sobre todos los atributos de una relación).

Algoritmo de optimización

6. Agrupar los nodos internos del árbol resultante de forma que se eviten varios accesos sobre la misma relación:
   - Ejecución simultánea de una selección seguida de una proyección o de un producto natural seguido de una proyección.
   - Sustitución de un producto cartesiano seguido de una selección sobre los atributos comunes por un producto natural.
7. Evaluar en cualquier orden cada grupo teniendo en cuenta que ningún grupo puede evaluarse antes que sus descendientes.

RECUPERACIÓN DE LA INFORMACIÓN

Recuperación de la información

Introducción

- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- **OBJETIVO:** proteger la base de datos contra posibles fallos que destruyan los datos bien en su totalidad o simplemente en alguna parte
- Fallos que afectan al SGBD:
  - Los que provocan la pérdida de memoria volátil (memoria principal).
  - Los que provocan la pérdida del contenido de memoria secundaria.

- El efecto del fallo es la **INCONSISTENCIA DE LA BASE DE DATOS.**

SISTEMA DE RECUPERACIÓN:

- Mecanismo que proporciona un SGBD para evitar o remediar estos fallos.
- Restaura la base de datos al estado consistente en que se encontraba antes del fallo.

Recuperación de la información

- Introducción
- **Concepto de transacción**
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

Concepto de transacción

- Una **transacción** es una unidad lógica de procesamiento de la base de datos que incluye una o más operaciones de acceso a la base de datos, que pueden ser de inserción, eliminación, modificación, o recuperación.

- Las operaciones de la base de datos que forman una transacción pueden estar insertadas dentro de un programa de aplicación o pueden especificarse interactivamente a través de un lenguaje de consulta como el SQL.

- Propiedades que debe tener una transacción:

  - Atomicidad
  - Conservación de la consistencia
  - Aislamiento
  - Persistencia

Recuperación de la información

- Introducción
- Concepto de transacción
- **Propiedad ACID**
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

Propiedad ACID

- Las transacciones deben poseer cuatro propiedades (ACID) recibe este nombre por sus iniciales en inglés:
  - **Atomicidad:** una transacción es una unidad atómica de procesamiento; o bien se realiza por completo o no se realiza en absoluto.
  - **Conservación de la consistencia:** una transacción conserva la consistencia si su ejecución completa lleva la base de datos de un estado consistente a otro.
  - **Aislamiento (en inglés Isolation):** una transacción debería parecer que se está ejecutando de forma aislada de las demás transacciones. Es decir, la ejecución de una transacción no debería interferir con otras transacciones que se ejecuten concurrentemente.
  - **Durabilidad o permanencia:** los cambios aplicados a la base de datos por una transacción confirmada deben perdurar en la base de datos. Estos cambios no deben perderse por un fallo posterior.

- Transacción para transferir 50 € desde una cuenta A a una cuenta B:

1. leer(A)
2. A := A – 50
3. escribir(A)
4. leer(B)
5. B := B + 50
6. escribir(B)

- **Requisito de consistencia** – la suma de A y B no se altera por la ejecución de la transacción.

- **Requisito de atomicidad** – si la transacción falla después del paso 3 y antes del paso 6, el sistema debería asegurar que sus actualizaciones no se reflejan en la base de datos, de lo contrario resultará una inconsistencia.

- **Requisito de durabilidad –** desde que se notifica al usuario que se ha completado la transacción (es decir, que ha tenido lugar la transferencia de 50 €), las actualizaciones de la base de datos producidas por la transacción deben permanecer, a pesar de los fallos.

- **Requisito de aislamiento –** si entre los pasos 3 y 6 se permite acceder a otra transacción a la base de datos parcialmente actualizada, verá una base de datos inconsistente (la suma de A + B será menor de lo que debería).

Se puede asegurar de forma trivial ejecutando las transacciones secuencialmente, es decir, una detrás de otra.
Sin embargo la ejecución de diversas transacciones concurrentemente tiene, como se verá, significativos beneficios.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- **Estado de una transacción**
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- Una transacción es una unidad atómica de trabajo que se realiza por completo o bien no se efectúa en absoluto.

- El gestor de recuperación utiliza las siguientes operaciones:

  - **BEGIN_TRANSACCIÓN (inicio de la transacción):** ésta marca el principio de la ejecución de la transacción.

  - **READ (leer) o WRITE (escribir):** éstas especifican operaciones de lectura o escritura de elementos de la base de datos que se ejecutan como parte de la transacción.

  - **END_TRANSACCIÓN (fin_de_transacción):** ésta especifica que las operaciones de LEER ó ESCRIBIR de la transacción han terminado y marca el fin de la ejecución de la transacción.

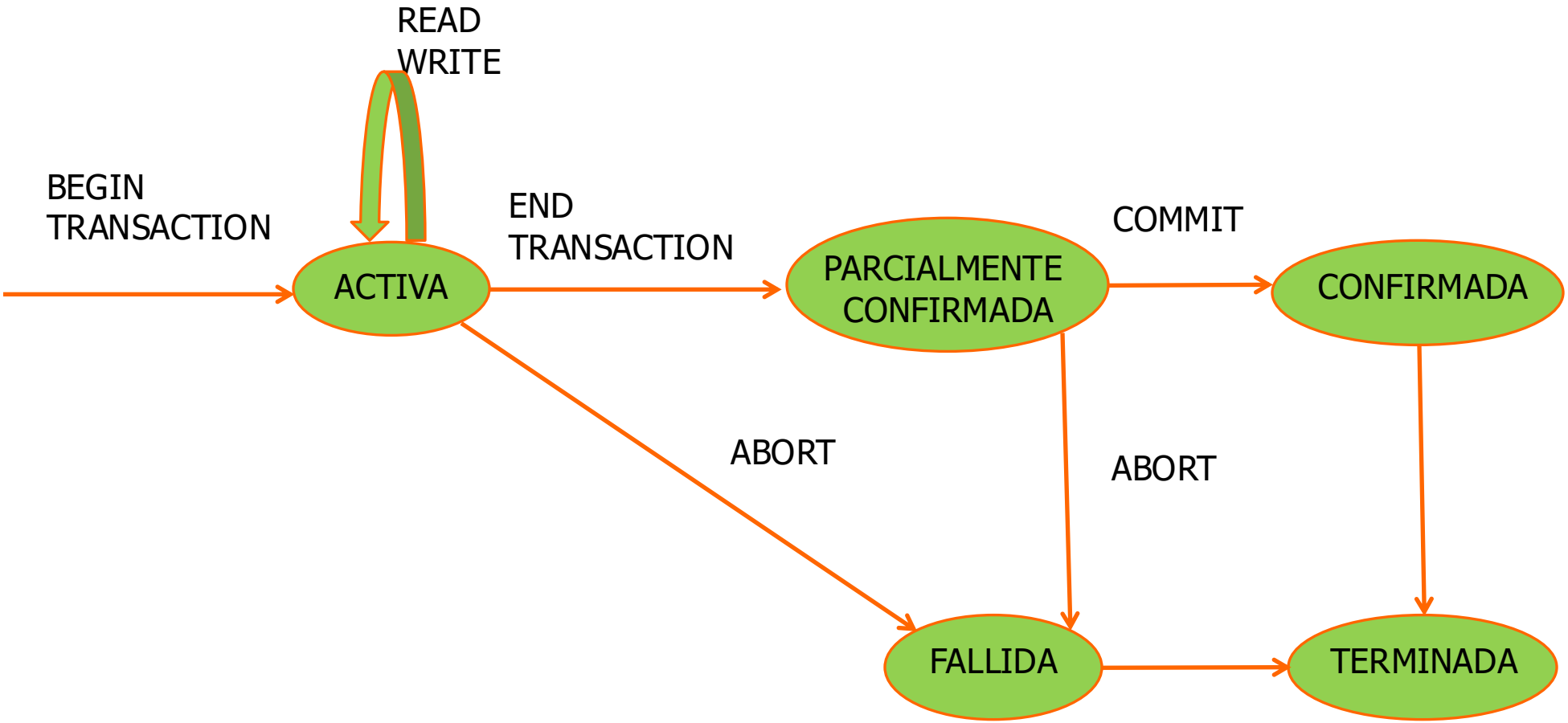
- **COMMIT_TRANSACCIÓN (confirmar transacción):** ésta señala que la transacción terminó con éxito y que cualquier cambio (actualizaciones) ejecutado por ella se puede confirmar sin peligro en la base de datos y que no se deshará.

- **ROLLBACK (restaurar) o ABORT (abortar):** éstas señales indican que la transacción terminó sin éxito y que cualquier cambio o efecto que puede haberse aplicado a la base de datos se debe deshacer.

Pasos de una transacción por sus estados de ejecución

1. La transacción entra en un **estado activo** inmediatamente después de que se inicie su ejecución y ahí puede emitir operaciones READ y WRITE.
2. Cuando la transacción termina pasa al estado **parcialmente confirmado.**
3. Se asegura que un fallo en el sistema no provocará incapacidad para registrar permanentemente los cambios hechos por la transacción.
4. Realizada con éxito esta verificación, se dice que la transacción ha llegado a su punto de confirmación y pasa al **estado confirmado**.
5. Sin embargo una transacción puede pasar al **estado fallido** si una de las verificaciones falla o si la transacción se aborta mientras está en el estado activo. En este caso es posible que deba hacerse un ROLLBACK de la transacción para anular los efectos de sus operaciones WRITE sobre la base de datos.
6. El estado terminado corresponde al abandono del sistema por parte de la transacción.

Paso de una transacción por sus estados de ejecución



Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- **Clasificación de los fallos**
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- Siempre que se introduce una transacción a un SGBD para ejecutarla, el sistema tiene que asegurarse de que todas las operaciones de la transacción se superen con éxito y su efecto quede registrado permanente en la base de datos, o que la transacción no tenga efecto alguno sobre la base de datos ni sobre cualquier otra transacción.

- El SGBD no debe permitir que se apliquen a la base de datos algunas operaciones de una transacción T y otras operaciones T no lo hagan.

- Tipos de fallos.

Fallos de transacción

Fallos del sistema

Fallos de los medios

- **Fallos en la transacción:**
  - ❖ **Errores lógicos:** la transacción no se puede completar debido a alguna condición de error interna
  - ❖ **Errores del sistema:** el sistema de la base de datos debe finalizar una transacción activa debido a una condición de error (por ejemplo, interbloqueo)
- **Fallos del sistema:** Caída del sistema: un fallo de potencia u otro fallo de hardware o software motiva la caída del sistema.
  - ❖ Supuesto de fallo-parada: se supone que los contenidos de almacenamiento no volátil no se corrompen por la caída del sistema

  Los sistemas de bases de datos tienen numerosas comprobaciones de integridad para prevenir la corrupción de los datos del disco

- **Fallos de los medios:** una caída de la cabeza o fallo similar del disco destruye todo o parte del almacenamiento del disco

  Se supone que la destrucción es detectable: los controladores del disco utilizan sumas de comprobación para detectar los fallos

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- **El diario del sistema**
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- Para poderse recuperar los fallos de transacciones, el sistema mantiene un diario (log) que sigue la pista a todas las operaciones de transacciones que afectan a los valores de la base de datos.

- Esta información puede necesitarse para realizar la recuperación en caso de fallos. El diario se mantiene en disco, de modo que no le afecta ningún tipo de fallo, más que los de disco y los catastróficos.

- **Registros del diario-** sirven para escribir en el diario cada una de las acciones que se realizan.
- En las entradas, T se refiere a un identificador de transacción único que el sistema genera automáticamente y que sirve para identificar la transacción.

1.- [start_transaction, T] : indica que se ha iniciado la ejecución de la transacción T.

2.- [write_item, T, X, valor-anterior, valor-nuevo]: indica que la transacción T ha cambiado el valor del elemento de la base de datos X del valor_anterior al nuevo_valor.

3.- [read_item, T,X]: registra que la transacción T leyó el valor del elemento X de la base de datos.

4.- [commit, T]: indica que la transacción T terminó con éxito y establece que su efecto se puede confirmar (registrar permanentemente) en la base de datos.

5.- [abort, T]: indica que se abortó la transacción T.

- Estamos suponiendo que todos los cambios permanentes sobre la base de datos ocurren en las transacciones, por lo que la notación de recuperarse de una transacción equivale a deshacer o rehacer las operaciones de la transacción individualmente a partir del diario.

- Si el sistema se cae, podemos recuperar la base de datos a un estado consistente examinando el diario y utilizando una serie de técnicas.

- Dado que el diario contiene un registro con cada operación write que altere el valor de algún elemento de la base de datos, es posible **deshacer** el efecto de estas operaciones write de la transacción T rastreando hacia atrás en el diario e iniciando todos los elementos alterados con una operación write de T con su valor_anterior.

- También puede ser necesario **rehacer** las operaciones de una transacción si todas sus actualizaciones se han grabado en el diario pero el fallo ocurrió antes de que estemos seguros de que todos esos nuevos valores se han escrito permanentemente en la base de datos del disco.

- Para **rehacer** el efecto de las operaciones write de una transacción T debe rastrearse **hacia delante** en el diario y cambiar todos los elementos modificados por una función escribir de T a su nuevo_valor.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- **Punto de confirmación de una transacción**
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

Punto de confirmación de una transacción

Una transacción T llega a un punto de confirmación cuando todas sus operaciones que tienen acceso a la base de datos se han ejecutado con éxito y el efecto de todas estas operaciones se ha registrado en el diario.

Cuando la transacción ha sido confirmada se supone que su efecto se registró permanentemente en la base de datos. En este momento escribe un registro [commit, T] en el diario. Si ocurre un fallo en el sistema, buscaríamos hacía atrás en el diario todas las transacciones T que han escrito una entrada [start-transaccion, T] en el diario pero no han escrito todavía su entrada [commit, T].
Las transacciones que ya escribieron su registro de confirmación en el diario también deberán haber registrado en él todas sus operaciones Write, y por tanto su efecto sobre la base de datos puede rehacerse a partir de los registros del diario.

El fichero del diario debe mantenerse en disco.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- **Recuperación y atomicidad**
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- Modificar la base de datos sin asegurarse de que la transacción se comprometerá puede dejar la base de datos en un estado inconsistente.

- Considérese la transacción T_i que transfiere 50€ de la cuenta A a la cuenta B; el objetivo es, bien realizar todas las modificaciones de la base de datos hechas por T_i o no realizar ninguna de ellas.

- Pueden ser necesarias diversas operaciones de salida para T_i (las salidas A y B). Se puede producir un fallo después de que se haya hecho una de estas modificaciones pero debe ser antes de que se hagan todas ellas.

- Para asegurar la atomicidad, a pesar de los fallos, primero se obtiene información describiendo las modificaciones en el almacenamiento estable, sin modificar la propia base de datos.

- Se estudian dos enfoques:
  - **recuperación basada en el registro histórico**
  - **paginación en la sombra**

- Se asume (inicialmente) que las transacciones se ejecutan secuencialmente, una tras otra.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- **Recuperación basada en el registro histórico**
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- Método más utilizado para guardar las modificaciones en un base de datos.

- El registro histórico es una secuencia de registros que mantiene un registro de todas las actividades de actualización de la base de datos.

- Cuando se **inicia** la transacción $\mathbf{T_i}$, se registra a si misma escribiendo un registro del registro histórico <$\mathbf{T_i}$ **iniciada**>

- **Antes** de que $\mathbf{T_i}$ **ejecute escribir(X)**, se escribe un registro del registro histórico <T_i, Xj, V_1, V_2>, en el que V_1 es el valor de Xj antes de la escritura, y V_2 es el valor que se va a dar a Xj.
  - El registro del registro histórico observa que T_i ha realizado una escritura en el elemento de datos X_j. X_j que tenía el valor V_1antes de la escritura, tendrá el valor V_2 después de la escritura.
- Cuando $\mathbf{T_i}$ **acaba** su última instrucción, se escribe en el registro del registro histórico <$\mathbf{T_i}$ **commit**>. Indica que la transacción se realizó con éxito y establece que su efecto se puede confirmar (registrar permanentemente) en la base de datos.

Dos enfoques utilizando registros históricos:

- **Modificación diferida de la base de datos**
- **Modificación inmediata de la base de datos**

- Cuando una transacción realiza una escritura es fundamental que se cree el registro del registro histórico correspondiente a esa escritura antes de modificar la base de datos. Una vez que el registro del registro histórico existe, se puede realizar la salida de la modificación de la base de datos si se desea. Además, es posible deshacer una modificación que ya haya salido a la base de datos. Se deshará utilizando el campo valor-anterior de los registros del registro histórico.

- Para que los registros del registro histórico sean útiles para recuperarse frente a errores de disco o del sistema, el registro histórico debe residir en almacenamiento estable.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- **Modificación diferida de la base de datos**
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

Modificación diferida de la base de datos

- La técnica de modificación diferida de la base de datos garantiza la atomicidad de las transacciones mediante el almacenamiento de todas las modificaciones de la base de datos en el registro histórico, pero retardando la ejecución de todas las operaciones de escribir de una transacción hasta que la transacción se compromete parcialmente (cuando se ejecuta la acción final de la transacción).
- Se supone que las transacciones se ejecutan secuencialmente.

- La ejecución de una transacción T_i opera de esta manera:
  - Antes de que T_i comience su ejecución se escribe en el registro histórico <T_i iniciada>.
  - Una operación escribir(X) realizada por T_i produce un registro en el registro histórico <T_i, X, V> que se escribe, donde V es el valor nuevo de X.
  - No se realiza la escritura en X en este momento, sino que se difiere.
  - Cuando T_i se compromete parcialmente, se escribe <T_i commit> en el registro histórico.
  - Por último, los registros del registro histórico se leen y se utilizan para ejecutar realmente las escrituras previamente diferidas.

- Durante la recuperación después de una caída, es necesario rehacer una transacción si y sólo si ambas <T_i iniciada> y <T_i commit> están en el registro histórico.

- Al rehacer una transacción T_i (rehacerT_i) se establece el valor de todos los elementos de datos actualizados por la transacción a los valores nuevos.

- Pueden producirse caídas mientras:
  - la transacción ejecuta las actualizaciones originales, o
  - mientras se lleva a cabo la acción de recuperación

- Ejemplo: T_0 y T_1 (T_0 se ejecuta antes que T_1):

T_0:	**leer** (A)	T_1 : **leer** (C)
	A: = A - 50	C:=C- 100
	Escribir (A)	**escribir** (C)
	leer (B)	
	B:= B + 50	
	escribir (B)	

- A continuación se muestra el registro histórico tal y como aparece en los tres ejemplos de tiempo.

(a)	(b)	(c)
$<T_0$ start>	$<T_0$ start>	$<T_0$ start>
$<T_0, A, 950>$	$<T_0, A, 950>$	$<T_0, A, 950>$
$<T_0, B, 2050>$	$<T_0, B, 2050>$	$<T_0, B, 2050>$
	$<T_0$ commit>	$<T_0$ commit>
	$<T_1$ start>	$<T_1$ start>
	$<T_1, C, 600>$	$<T_1, C, 600>$
		$<T_1$ commit>

- Si el registro histórico del almacenamiento estable en el momento de la caída es como el de ese caso:
  - (a) No es necesario rehacer las acciones que se vayan a tomar
  - (b) se debe llevar a cabo **rehacer(T_0)** ya que **<T_0 commit>** está presente
  - (c) se debe llevar a cabo **rehacer(T_0)** seguida por **rehacer(T_1)** ya que están presentes **<T_0 commit>** y **<T_1 commit>**

- Garantiza la atomicidad de una transacción grabando todas las modificaciones en el diario y aplazando la grabación de las modificaciones en la base de datos hasta que la transacción ha finalizado (se compromete parcialmente).

- Utiliza los registros del fichero de diario para actualizar la base de datos (modificación diferida) que deben estar en memoria estable por si ocurre un fallo durante el proceso de recuperación.

- Si el sistema cae antes de que la transacción termine o si la transacción aborta, la información correspondiente del fichero de diario se ignora (se puede simplificar la estructura del fichero diario omitiendo el valor inicial del dato modificado).

Modificación diferida de la base de datos

- No actualizan físicamente la base de datos en disco hasta después de que una transacción llega a su punto de confirmación; en este momento, las actualizaciones se graban en el espacio de transacciones local (o bufers).

- Durante la confirmación, las actualizaciones se graban primero definitivamente en el diario y luego se escriben en la base de datos.

- Si una transacción falla antes de llegar a su punto de confirmación, no habrá modificado en absoluto la base de datos, por lo que no es necesario DESHACER. Puede ser necesario REHACER el efecto de las operaciones de una transacción confirmada desde el diario, ya que es posible que su efecto todavía no se haya grabado en la base de datos. Por ello, la actualización diferida se conoce también como algoritmo NO-DESHACER/REHACER.

- Pasos a realizar en caso de un FALLO:

1. El sistema de recuperación consulta el diario para determinar las transacciones que deben repetirse.
2. Una transacción debe repetirse si y sólo si el fichero de diario contiene el registro $<T_i$ starts> y $<T_i$ commits>.
3. El proceso de recuperación asigna los nuevos valores a todos los datos que actualiza la transacción utilizando la información del diario: operación REDO(T_i).

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- **Modificación inmediata de la base de datos**
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- El esquema de modificación inmediata de la base de datos permite realizar la salida de las modificaciones de la base de datos a la propia base de datos mientras que la transacción está todavía en estado activo.

- En caso de un fallo o una caída en la transacción, el sistema debe utilizar el campo del valor anterior de los registros del registro histórico para restaurar los elementos de datos modificados a los valores que tuvieran antes de comenzar la transacción. Esta operación se lleva a cabo a través de la operación deshacer.

- Antes de comenzar la ejecución de una transacción T_i, se escribe en el registro histórico del registro <T_i iniciada>.

- Durante su ejecución, cualquier operación escribir(X) realizada por T_i, es precedida por la escritura del registro histórico de un registro actualizado apropiado.

- Cuando T_i se compromete parcialmente se escribe en el registro histórico el registro <Ti commit>.

- Como la información del registro histórico se utiliza para reconstruir el estado de la base de datos, la actualización real de la base de datos no puede permitirse antes de que el registro del registro histórico correspondiente se haya escrito en almacenamiento estable. Por lo tanto es necesario que antes de la ejecución de la operación de salida(B), se escriban en almacenamiento estable los registros del registro histórico correspondientes a B.

- La grabación de los bloques actualizados puede tener lugar en cualquier momento antes o después del compromiso de la transacción.

- El orden en el que se graban los bloques puede ser diferente del orden en el que se escriben.

- Ejemplo de modificación inmediata de la base de datos

Registro histórico	**Escritura**	**Salida**
<T_0 **iniciada**>		
<T_0, A, 1000, 950>		
<T_0, B, 2000, 2050>		
	A = 950 B = 2050	
<T_0 **commit**>		
<T_1 **iniciada**>		
<T_1, C, 700, 600>		
	C = 600	
		B_B, B_C
<T_1 **commit**>		
		B_A

Nota: B_X indica el bloque que contiene X.

- El procedimiento de recuperación utiliza dos procedimientos en vez de uno:
  - **deshacer (T_i)** restaura el valor de todos los elementos de datos actualizados por T_i a los valores anteriores, yendo hacia atrás desde el último registro del registro histórico de T_i.
  - **rehacer (T_i)** fija el valor de todos los elementos de datos actualizados por T_i a los valores nuevos, yendo hacia adelante desde el primer registro del registro histórico de T_i.
- Ambas operaciones deben ser idempotentes, es decir, incluso si se ejecuta la operación muchas veces el efecto debe ser el mismo que el de ejecutarse sólo una vez.
  - Necesario ya que las operaciones pueden reejecutarse durante la recuperación.

- Cuando la recuperación se produce después de un fallo:
  - Es necesario **deshacer** la transacción T_i si el registro histórico contiene el registro <**T_i iniciada**>, pero **no** contiene el registro <**T_i commit**>.
  - Es necesario **rehacer** la transacción Ti si el registro histórico contiene ambos registros <**T_i iniciada**> y <**T_i commit**>.
- **Primero** se realizan las operaciones de **deshacer**, después las de rehacer.

- Ejemplo de modificación inmediata de la base de datos:
  A continuación se representa el registro histórico tal y como aparecen en los tres ejemplos de tiempo

(a)	(b)	(c)
<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 commit>

Las acciones de recuperación en cada caso anterior son:
(a) deshacer (T_0): B se restituye a 2000 y A a 1000.
(b) deshacer (T_1) y rehacer (T_0): C se restituye a 700, y entonces A y B se establecen en 950 y 2050 respectivamente.
(c) rehacer (T_0) y rehacer (T_1): A y B se establecen en 950 y 2050 respectivamente. Entonces C se establece en 600.

- Si ocurre un fallo utiliza el valor inicial de los registros del fichero de diario para restaurar los datos modificados al valor anterior a la transacción: operación UNDO (T_i).

- Los registros del diario deben estar en almacenamiento estable antes de la modificación de la base de datos.

- Después de un fallo el sistema de recuperación consulta el diario para determinar que transacciones se deben repetir y cuales deben deshacerse:

  UNDO(T_i): el diario contiene <T_i starts> pero no contiene <T_i commits>.
  REDO(T_i): el diario contiene tanto <T_i starts> como <T_i commits>.

- Propiedad IDEMPOTENTE de las operaciones REDO y UNDO.

- Si el sistema cae antes de que la transacción termine o si la transacción aborta, la información correspondiente del fichero de diario se ignora (se puede simplificar la estructura del fichero diario omitiendo el valor inicial del dato modificado).

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- **Puntos de revisión**
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- Cuando ocurre un fallo en el sistema se debe consultar el registro histórico para determinar las transacciones que deben rehacerse y las que deben deshacerse. En principio es necesario recorrer completamente el registro histórico para hallar esta información.

- Los problemas del procedimiento de recuperación son:

1. La búsqueda del registro histórico supone consumo de tiempo
2. La mayoría de las transacciones que deben rehacerse ya tienen reescritas sus actualizaciones en la base de datos.Aunque el hecho de volver a ejecutar estas transacciones no produzcan resultados erróneos, sí repercutirá en un aumento del tiempo de ejecución del proceso de recuperación.

- Para reducir este tipo de sobrecarga se introducen los puntos de revisión. Durante la ejecución, el sistema actualiza el registro histórico utilizando una de las técnicas vistas anteriormente (modificación diferida o inmediata de la base de datos).
- El procedimiento de recuperación de actualización se realiza periódicamente mediante **los puntos de revisión.**

1. Escritura en almacenamiento estable de todos los registros del registro histórico que actualmente residen en la memoria principal.
2. Escritura en el disco de todos los bloques de memoria intermedia que se hayan modificado.
3. Escritura en almacenamiento estable de un registro del registro histórico <**revisión**>.

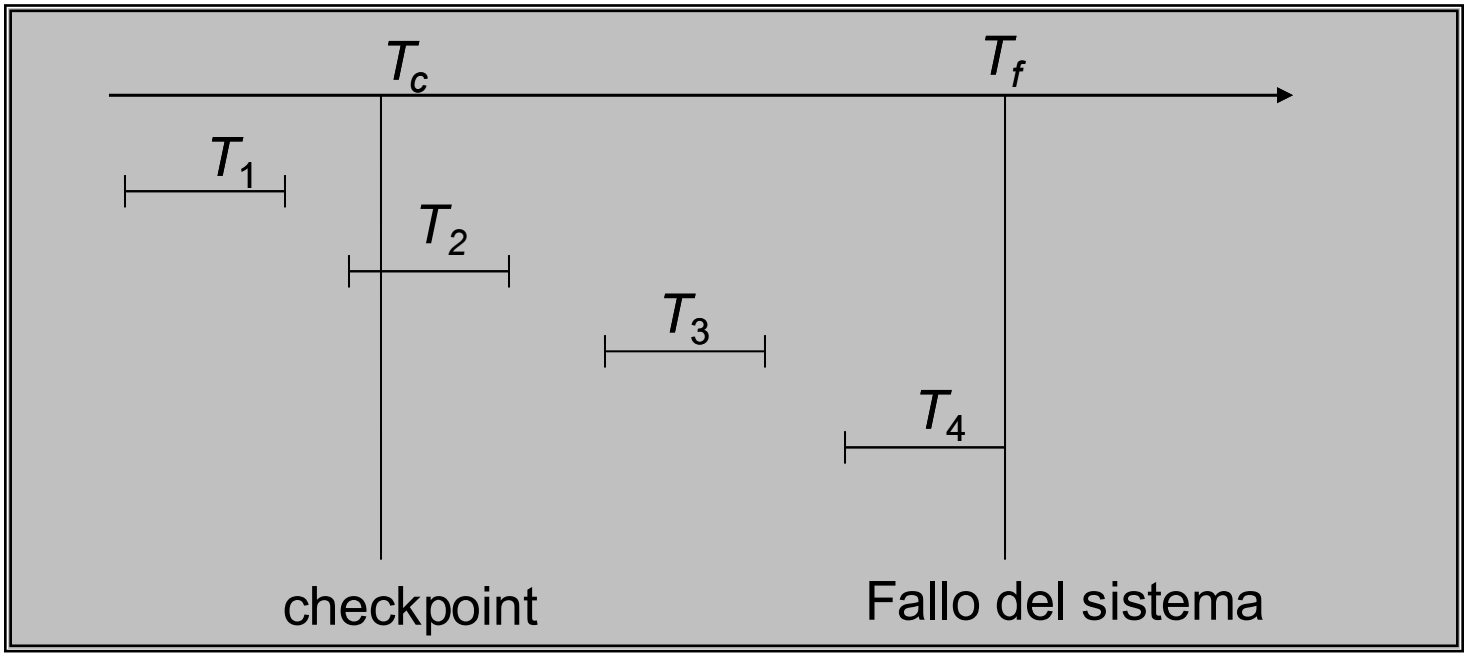
- Mientras se lleva a cabo un punto de revisión no se permite que ninguna transacción realice acciones de actualización, tales como escribir en un bloque de memoria intermedia o escribir un registro en el registro histórico.

- La presencia de un registro <**revisión**> en el registro histórico permite que el sistema pueda hacer más eficiente su procedimiento de recuperación.

- Durante la recuperación es necesario considerar sólo la transacción más reciente T_i que se inició antes del punto de revisión, y las transacciones que se iniciaron después de T_i.

1. Rastrear hacia atrás desde el final del registro histórico para hallar el registro <**revisión**> más reciente.
2. Continuar rastreando hacia atrás hasta que se encuentre el registro <$\mathbf{T_i}$ **iniciada**>.
3. Sólo es necesario considerar la parte del registro histórico que sigue al registro anterior **iniciada**. Se puede ignorar durante la recuperación la parte anterior del registro histórico, y se puede borrar cuando se desee.
4. Para todas las transacciones (comenzando desde T_i o posteriores) sin <$\mathbf{T_i}$ **commit**>, se ejecuta **deshacer($\mathbf{T_i}$)**. (Se hace sólo en caso de modificación inmediata).
5. Rastreando hacia adelante en el registro histórico, para todas las transacciones que empiecen desde T_i o posteriores con un registro <$\mathbf{T_i}$ **commit**>, ejecutar **rehacer($\mathbf{T_i}$)**.

- Ejemplo de puntos de revisión



- T_1 se puede ignorar (las actualizaciones ya han sido escritas en disco)
- T_2 y T_3 rehacer
- T_4 deshacer

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- **Paginación en la sombra**
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- La **paginación en la sombra** es una alternativa a la recuperación basada en el registro histórico; este esquema es útil si se ejecutan las transacciones secuencialmente.

- La base de datos se divide en un número determinado de bloques de longitud fija a los que se denomina páginas. Para localizar cada una de estas páginas se usa una tabla de páginas.

- La tabla de páginas tiene n entradas una para cada página de la base de datos. Cada entrada contiene un puntero a una página en el disco. La primera entrada contiene un puntero a la primera página de la base de datos, la segunda entrada apunta a la segunda página y así sucesivamente.

- Idea: mantener dos tablas de páginas durante la vida de una transacción –**la tabla de páginas actual** y **la tabla de páginas sombra.**

- Almacenar la tabla de páginas sombra en almacenamiento no volátil, de forma que se pueda recuperar el estado de la base de datos anterior a la ejecución de la transacción.

  - La tabla de páginas sombra nunca se modifica durante la ejecución.

- Cuando comienza una transacción, ambas tablas de páginas son idénticas. Mientras dura la transacción no se altera el contenido de la tabla de páginas sombra.

- Supóngase que la transacción realiza una operación escribir(X) y que X pertenece a la página i-ésima. La operación escribir se ejecutará como sigue:

  1.- Ejecutar entrada(X) si la página i-ésima (es decir, la página en la que se encuentra X) no está todavía en memoria principal.

  2.- Si es la primera escritura que esta transacción realiza sobre la página i-ésima, modificar la tabla actual de páginas siguiendo los pasos:

  a) Encontrar una página en el disco que no se haya utilizado. Normalmente el sistema de bases de datos tiene acceso a una lista de páginas no utilizadas (libres).
  b) Borrar la página encontrada en el paso anterior de la lista de marcos de páginas libres.
  c) Modificar la tabla actual de páginas de modo que la entrada i-ésima apunte a la página encontrada en el paso 2 a.

  3.- Asignar el valor de Xj a X en la página de memoria intermedia.

Paginación en la sombra

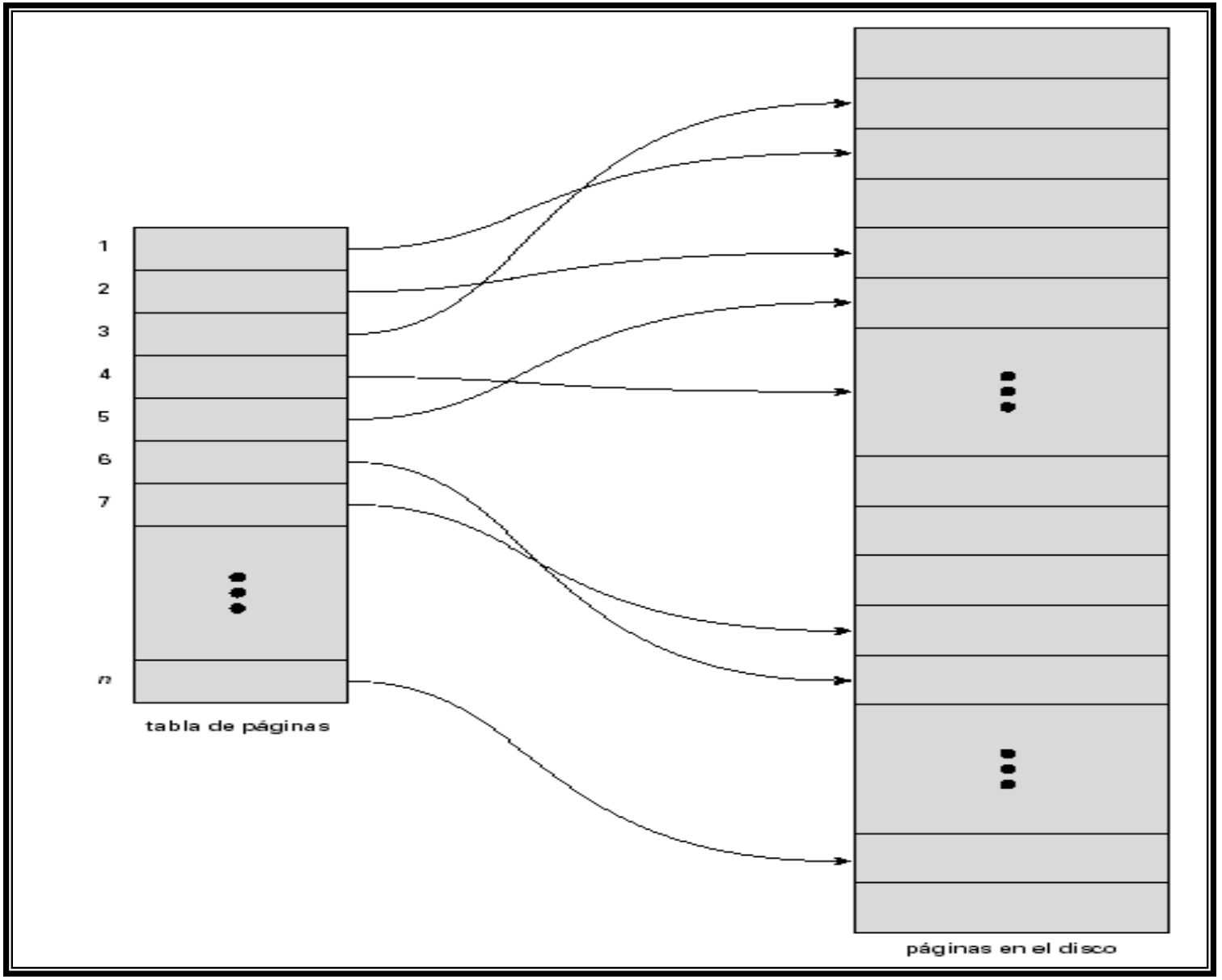
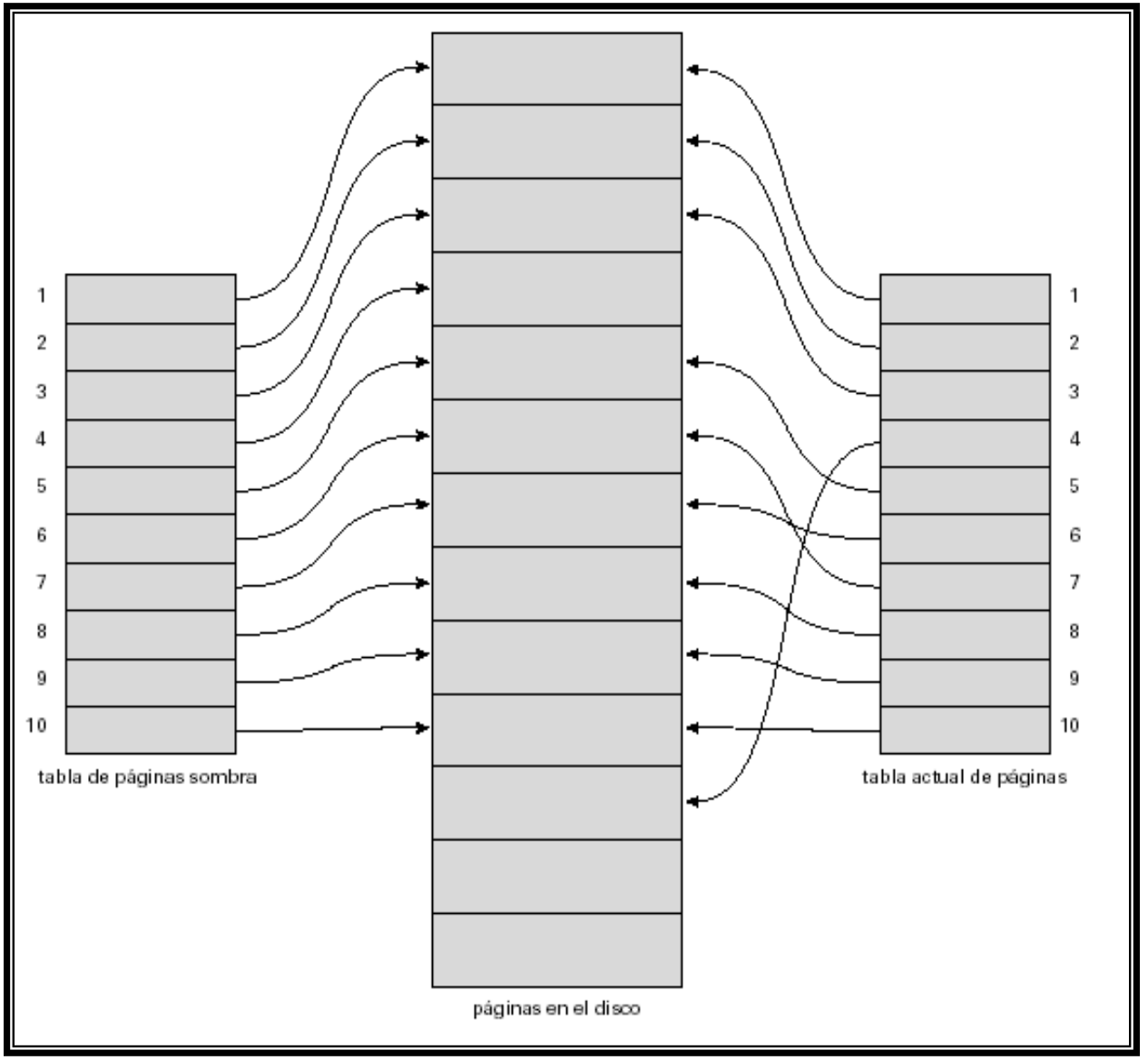


Tabla actual y en la sombra para una transacción que está realizando una escritura en la cuarta página de una base de datos que tiene 10 páginas.



- Para comprometer una transacción:
  1. Pasar todas las páginas modificadas de la memoria principal al disco.
  2. Escribir la tabla de páginas actual en el disco. No se debe sobreescribir la tabla de páginas sombra pues puede ser necesaria para el proceso de recuperación si ocurriese una caída.
  3. Hacer de la tabla de páginas actual la nueva tabla de páginas sombra, de la siguiente forma:
     - Mantener un puntero en la tabla de páginas sombra en una localización fija (conocida) del disco.
     - Hacer de la tabla de páginas actual la nueva tabla de páginas sombra, simplemente actualizar el puntero para que apunte la tabla de páginas actual del disco.
- Una vez que se ha escrito el puntero en la tabla de páginas sombra, la transacción está comprometida.
- No es necesaria la recuperación después de una caída – las transacciones nuevas pueden iniciarse al instante, utilizando la tabla de páginas sombra.
- Deberían liberarse las páginas no apuntadas a/desde la tabla de páginas sombra actual (recogida de basura).

- **Recogida de basura**:

- Cada vez que se compromete una transacción, las páginas de la base de datos que contenían la versión anterior de los datos y que fueron modificadas por la transacción, se hacen inaccesibles.

- Estas páginas son consideradas como basura debido a que no forman parte del espacio libre y no contienen ninguna información útil.

- Puede producirse basura también como efecto lateral de las caídas.Es necesario encontrar periódicamente todas las páginas basura para añadirlas a la lista de páginas libres. Este proceso de denomina recogida de basura.

- **Ventajas** de la paginación en la sombra frente a los esquemas basados en el registro histórico
  - No hay sobrecarga de escrituras en los registros del registro histórico
  - La recuperación es trivial
- **Desventajas:**
  - Grabar la tabla completa de páginas es muy costoso
  - La sobrecarga de compromiso es elevada
    - Es necesario grabar cada página actualizada, y la tabla de paginación
  - Los datos se fragmentan (las páginas relacionadas se separan en el disco).
  - Después de la finalización de cada transacción, las páginas de la base de datos que contienen las versiones anteriores de datos modificados, necesitan recoger la basura.
  - Difícil de extender algoritmos para permitir que las transacciones funcionen de forma concurrente.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- **Recuperación con transacciones concurrentes**
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- Se modifican los esquemas de recuperación basados en el registro histórico para permitir que múltiples transacciones se ejecuten de forma concurrente.
  - Todas las transacciones comparten una sola memoria intermedia del disco y un sólo registro histórico independientemente del número de transacciones concurrentes.
  - Como todas las transacciones comparten los bloques de memoria intermedia. Un bloque de la memoria intermedia puede tener elementos de datos actualizados por medio de una o más transacciones.
- El esquema de recuperación depende del esquema de control de concurrencia que se use.
- Para retroceder los efectos de una transacción fallida deben deshacerse las modificaciones realizadas por esa transacción.

- Supóngase que se debe retroceder una transacción T_0 y que un dato Q que fue modificado por T_0 tiene que recuperar su antiguo valor.
  - **Si se está usando un esquema basado en registro histórico** para la recuperación, es posible restablecer el valor de Q utilizando la información contenida en el registro histórico.

  - Supóngase ahora que una segunda transacción T_1 realiza una nueva modificación sobre Q antes de retroceder T_0. En este caso, al retroceder T_0 ,se perdería la modificación realizada por T_1.

  - Es necesario, por tanto, que si una transacción T modifica el valor de un elemento de datos Q, ninguna otra transacción pueda modificar el mismo elemento de datos hasta que T se haya comprometido o se haya retrocedido.

  - Esto se consigue con el **bloqueo estricto de dos fases**, es decir, **las actualizaciones de transacciones no comprometidas no deberían ser visibles en otras transacciones**.

- Si utilizamos el método basado en el **registro histórico** para retroceder una transacción T_i fallida, se actúa como se ha explicado en transparencias anteriores.

- El registro histórico se explora hacia atrás; para cada registro del registro histórico de la forma $<T_i,X_j,V_1,V_2>$, se restablece el valor del elemento de datos X_j con su valor anterior V_1.

- La exploración del registro histórico termina cuando se encuentra el registro $<T_i$ iniciada$>$.

- Es importante recorrer el registro histórico empezando por el final, ya que una transacción puede haber actualizado más de una vez el valor de un elemento de datos.

- Ejemplo : considérense estos registros del registro histórico.

  <T_i, A, 10, 20>

  <T_i, A, 20, 30>

- Representan una modificación del elemento A por parte de la transacción T_i, seguida de otra modificación de A hecha también por T_i.

- Al recorrer el registro histórico al revés, se establece correctamente el valor de A como 10. Si el registro histórico se recorriera hacia delante, A tomaría el valor de 20, lo cual es incorrecto.

- Si se utiliza el bloqueo estricto de dos fases, los bloqueos llevados a cabo por una transacción T sólo pueden ser desbloqueados después de que la transacción se haya retrocedido según se acaba de describir.

- Una vez que T (que se está retrocediendo) haya actualizado un elemento de datos, ninguna otra transacción podría haber actualizado el mismo elemento de datos (requisito de control de concurrencia).

- Si se usan **puntos de revisión**:
- Varias transacciones pueden estar activas en el momento en que se produce el último punto de revisión.
- Es necesario que el registro del registro histórico correspondiente a un punto de revisión sea de la forma <revisión L>, donde L es una lista con las transacciones activas en el momento del punto de revisión.
- Se supone que mientras que se realiza un punto de revisión, las transacciones no efectúan modificaciones ni sobre los bloques de memoria intermedia ni sobre el registro histórico.

- Cuando **el sistema se recupera de una caída**, el sistema construye dos listas: **lista-deshacer** y la **lista-rehacer**.
- Se hace lo siguiente:

1. Inicializar la lista-deshacer y la lista-rehacer con el valor vacío
2. Rastrear el registro histórico hacia atrás desde el final, deteniéndose cuando se encuentre el primer registro <revisión L>
3. Para cada registro encontrado durante el rastreo hacia atrás:
   - si el registro es <T_i commit>, añadir T_i a la lista-rehacer
   - si el registro es <T_i iniciada>, entonces si T_i no está en la lista-rehacer, añadir T_i a la lista-deshacer
4. Para cada T_i de L, si T_i no está en la lista-rehacer, añadir T_i a la lista-deshacer

- En este momento la lista-deshacer consta de transacciones incompletas que deben deshacerse, y la lista-rehacer consta de transacciones acabadas que deben rehacerse.
- La recuperación continúa ahora de la siguiente forma:

1. Se rastrea el registro histórico **hacia atrás** desde el registro más reciente, deteniéndose cuando se hayan encontrado los registros <T_i iniciada> de cada T_i en la lista-deshacer.
2. Localizar el último registro <revisión L> del registro histórico.
3. Rastrear el registro histórico **hacia delante** desde el último registro <revisión L> y se realiza una operación rehacer por cada registro del registro histórico que pertenezca a una transacción T_i de la lista-rehacer.

- Es importante procesar el registro histórico hacía atrás en el paso 1 para garantizar que el estado de la base de datos es el correcto.
- Después de haber deshecho todas las transacciones de la lista-deshacer se rehacen aquellas transacciones que pertenezcan a la lista-rehacer. En este caso es importante procesar el registro histórico hacia delante.
- Cuando se ha completado el proceso de recuperación, se continúa con el procesamiento normal de las transacciones.

- **Es importante deshacer las transacciones de la lista-deshacer antes de rehacer las transacciones de la lista-rehacer al utilizar los pasos del 1 al 3 del algoritmo anterior.**

- Supóngase que el elemento de datos A vale inicialmente 10. Supóngase también que una transacción T_i modifica el valor de A situándolo en 20 y aborta a continuación; el retroceso de la transacción devolvería a A el valor de 10. Supóngase que otra transacción T_j cambia entonces a 30 el valor de A, se compromete y, seguidamente, el sistema cae. El estado del registro histórico en el momento de la caída es:

  <Ti , A, 10,20>
  <Tj , A, 10,30>
  <Tj , comprometida>

  Si se rehace primero, A tomará el valor de 30; y luego, al deshacer, A acabará valiendo 10, lo cual es incorrecto. El valor de A debe ser 30, lo que puede garantizarse si se deshace antes de rehacer.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Punto de confirmación de una transacción
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- **Registro histórico con memoria intermedia**
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

- **Registro histórico con memoria intermedia:** los registros del registro histórico se escriben en una memoria intermedia de la memoria principal, en vez de grabarse directamente en almacenamiento estable.
    - Los registros del registro histórico se graban en almacenamiento estable cuando está completo un bloque de los registros del registro histórico en la memoria intermedia, o se ejecuta una operación de **forzar el registro histórico** (escritura en disco de la memoria intermedia del registro histórico).
- Para comprometer una transacción se debe forzar el registro histórico, forzando todos sus registros.
- Se pueden grabar así varios registros del registro histórico utilizando una única operación de salida, reduciendo el coste de E/S.

- Si los registros del registro histórico están en memoria intermedia, deben cumplirse las siguientes reglas:
  - Los registros del registro histórico se graban en almacenamiento estable, en el orden en que se han creado.
  - La transacción T_i entra al estado de compromiso, sólo cuando el registro del registro histórico <T_i *comprometida*> se ha grabado en almacenamiento estable.
  - Antes de que un bloque de datos de la memoria principal se grabe en la base de datos, todos los registros del registro histórico vinculados a los datos del bloque deben haber sido llevados a almacenamiento estable.

    - Esta regla se denomina, regla del registro histórico de escritura anticipada o (REA).
      - Estrictamente hablando REA sólo necesita que se haya grabado la información deshacer.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- **Memoria intermedia de la base de datos**
- Fallo con pérdida de almacenamiento no volátil
- Sistemas remotos de copias de seguridad

Memoria intermedia de la base de datos

- La base de datos mantiene una memoria intermedia en memoria de los bloques de datos.
  - Cuando es necesario un bloque nuevo, si está completa la memoria intermedia, es necesario retirar el bloque existente de la memoria intermedia.
  - Si se ha actualizado el bloque elegido para la eliminación, se debe grabar en el disco.
- Como resultado de la regla del registro histórico de escritura anticipada, si un bloque con actualizaciones no comprometidas se graba en el disco, los registros del registro histórico con información de rehacer las actualizaciones se graban en el registro histórico del primer almacenamiento estable.

- No debería haber actualizaciones en progreso en un bloque cuando se graba en el disco.

- Nos podemos asegurar de la siguiente forma:
  - Antes de escribir un elemento de datos, la transacción adquiere el bloqueo exclusivo en el bloque que contiene el elemento de datos.

  - Puede eliminarse el bloqueo una vez que se completa la escritura.
    - Dichos bloqueos de corta duración se denominan **pestillos**.

  - Antes de que un bloque se grabe en un disco, el sistema adquiere un pestillo exclusivo del bloque.
    - Se asegura de que no puede haber ninguna actualización en progreso en el bloque.

Memoria intermedia de la base de datos

- La memoria intermedia de la base de datos puede implementarse bien:
  - en un área de la memoria principal real reservada para la base de datos, o
  - en la memoria virtual
- Implementar la memoria intermedia en la memoria principal reservada tiene inconvenientes:
  - La memoria se divide de antemano entre la memoria intermedia de la base de datos y las aplicaciones, limitando la flexibilidad.
  - Necesita poder cambiar, y aunque el sistema operativo sabe mejor como se debería dividir la memoria en cualquier momento, no puede cambiar la división de la memoria.

- Las memorias intermedias de la base de datos se implementan generalmente en la memoria virtual a pesar de algunos inconvenientes:
  - Cuando el sistema operativo necesita expulsar una página que se ha modificado, para hacer espacio a otra página, esta página se escribe en un espacio en disco llamado espacio de intercambio.
  - ¡Cuándo la base de datos decide escribir una página de memoria intermedia en el disco, la página de la memoria intermedia puede estar en el espacio de intercambio, y puede que haya tenido que leerse desde el espacio de intercambio del disco y grabarse en la base de datos de dicho disco, produciendo una E/S adicional!
    - Conocido como el problema de **paginación dual**.

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- **Fallo con pérdida de almacenamiento no volátil**
- Sistemas remotos de copias de seguridad

Fallo con pérdida de almacenamiento no volátil

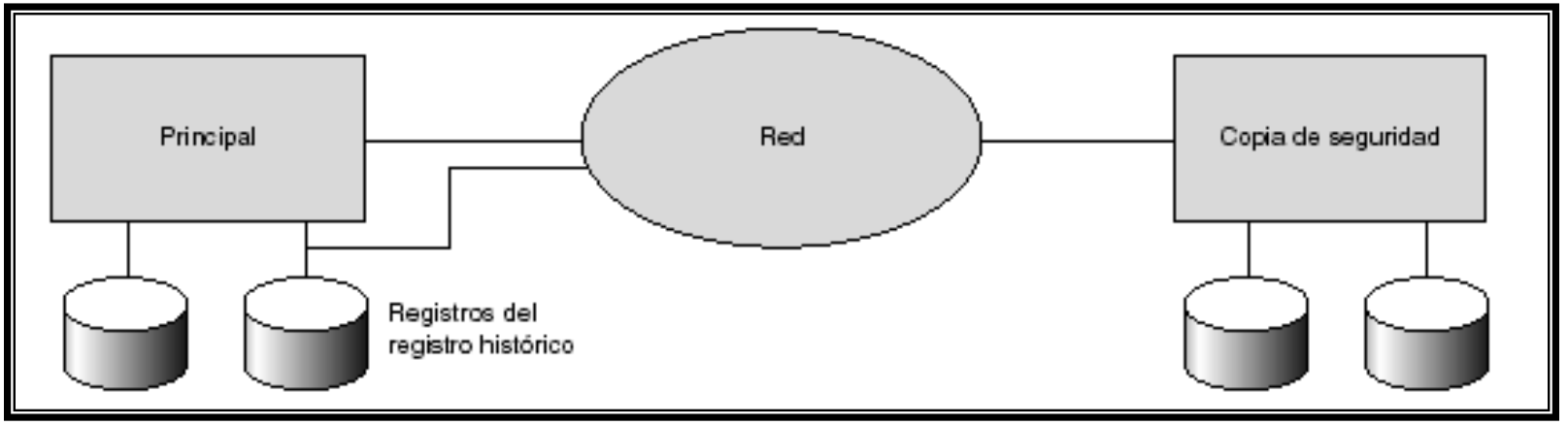
- Volcar periódicamente el contenido completo de la base de datos en almacenamiento estable.
- La transacción no puede estar activa durante el procedimiento de volcado; debe tener lugar un procedimiento similar al de revisión:
  - Grabar en almacenamiento estable todos los registros del registro histórico que actualmente residan en la memoria principal.
  - Grabar en el disco todos los bloques de memoria intermedia.
  - Copiar los contenidos de la base de datos en almacenamiento estable.
  - Grabar un registro <volcar> en el registro histórico en almacenamiento estable.
- Para la recuperación del fallo del disco:
  - Restaurar la base de datos desde el volcado más reciente.
  - Consultar el registro histórico y rehacer todas las transacciones que estén comprometidas después de volcar.
- Se pueden extender para permitir que se activen las transacciones durante el volcado; conocido como **volcado difuso o volcado en línea.**

Recuperación de la información

- Introducción
- Concepto de transacción
- Propiedad ACID
- Estado de una transacción
- Clasificación de los fallos
- El diario del sistema
- Recuperación y atomicidad
- Recuperación basada en el registro histórico
- Modificación diferida de la base de datos
- Modificación inmediata de la base de datos
- Puntos de revisión
- Paginación en la sombra
- Recuperación con transacciones concurrentes
- Registro histórico con memoria intermedia
- Memoria intermedia de la base de datos
- Fallo con pérdida de almacenamiento no volátil
- **Sistemas remotos de copias de seguridad**

Sistemas remotos de copias de seguridad

- Los sistemas remotos de copia de seguridad proporcionan elevada disponibilidad permitiendo que el procesamiento de transacciones continúe incluso si se destruye el sitio primario.



- **Detección de fallos:** El sitio de la copia de seguridad debe detectar cuando ha fallado el sitio principal para distinguir el fallo del sitio principal del fallo del enlace. Mantener varios enlaces de comunicación entre la copia de seguridad principal y la remota.
- **Transferencia del control:**
  - Para tener bajo control el sitio de la copia de seguridad llevar a cabo primero la recuperación utilizando su copia de la base de datos y todos los registros que ha recibido del sitio principal.
    Así, las transacciones completas se rehacen y las incompletas se retroceden.
  - Cuando el sitio de la copia de seguridad se encarga de procesarla se convierte en el nuevo sitio principal.
  - Para transferir el control de nuevo al sitio principal anterior cuando se recupera, el sitio principal anterior debe recibir rehacer registro históricos de la anterior copia de seguridad y aplicar todas las actualizaciones localmente.

- **Tiempo de recuperación:** Para reducir la demora en la toma de control, el sitio de la copia de seguridad periódicamente procesa los registros del registro histórico de rehacer (en efecto, lleva a cabo la recuperación del estado de la base de datos anterior), realiza un punto de revisión, y después puede borrar las partes más antiguas del registro histórico.

- La configuración de **relevo en caliente** permite tomas de control muy rápidas:

  - La copia de seguridad procesa continuamente el registro del registro histórico rehacer tal y como llega, aplicando las actualizaciones localmente.
  - Cuando se detecta el fallo del sitio principal, la copia de seguridad retrocede las transacciones incompletas, y se prepara para procesar nuevas transacciones.

- Alternativa a la copia de seguridad remota: base de datos distribuida con datos replicados
  - La copia de seguridad remota es más rápida y más barata, pero menos tolerante a los fallos

- Asegurar la durabilidad de las actualizaciones demorando el compromiso de la transacción hasta que la actualización se registre en la copia de seguridad.

- **Uno seguro:** se comprometen tan pronto como sus registros de compromiso del registro histórico se escriben en el sitio principal.
  - Problema: las actualizaciones no alcanzan la copia de seguridad antes de que se asuma el control.
- **Dos muy seguro:** se comprometen cuando sus registros de compromiso del registro histórico se escriben en el sitio principal y en el sitio copia de seguridad.
  - Reduce la disponibilidad ya que las transacciones no pueden comprometerse si falla el sitio.
- **Dos seguro:** se procede como en el de dos muy seguro si están activos tanto el sitio principal como el de la copia de seguridad. Si solo está activo el sitio principal, la transacción se compromete tan pronto como se escribe el registro de compromiso del registro histórico en el sitio principal.
  - La mejor disponibilidad es la de dos muy seguro; evita el problema de transacciones perdidas de uno seguro.

CONTROL DE LA CONCURRENCIA

Control de la concurrencia

- Ejecuciones concurrentes
- Planificaciones
- Protocolos basados en bloqueos
- Control de concurrencia basado en marcas de tiempo
- Granularidad de los ítems de datos

Ejecuciones concurrentes

- Los SGBD admiten múltiples usuarios simultáneamente y por tanto cualquier número de transacciones pueden tener acceso a la base de datos al mismo tiempo.

- Las ventajas son:

  - Aumento de la utilización del procesador y del disco, conduciendo a mejorar la productividad de la transacción, ya que una transacción puede estar utilizando la CPU, mientras otras están leyendo desde o escribiendo a disco.

  - Reducción del tiempo medio de respuesta de las transacciones: las transacciones pequeñas no tienen necesidad de esperar detrás de las grandes.

Ejecuciones concurrentes

- Para evitar la interacción entre transacciones y la destrucción de la consistencia de la base de datos tenemos los **Esquemas de control de Concurrencia.**

- Esquemas de control de concurrencia - mecanismos para conseguir aislamiento, es decir, para controlar la interacción entre las transacciones concurrentes a la hora de impedir que destruyan la consistencia de la base de datos.

- Las principales situaciones en las que una transacción (correcta individualmente) puede producir un resultado incorrecto al interferir con otra transacción (correcta también individualmente) son:

  - Problema de la modificación perdida.
  - Problema de la dependencia no comprometida.
  - Problema del análisis inconsistente.

Control de la concurrencia

Ejecuciones concurrentes

Planificaciones

Protocolos basados en bloqueos

Control de concurrencia basado en marcas de tiempo

Granularidad de los ítems de datos

Planificaciones – Diferentes secuencias que indican el orden cronológico en el que se ejecutan las instrucciones de las transacciones concurrentes.

- Una planificación para un conjunto de transacciones debe constar de todas las instrucciones de estas transacciones.

- Debe preservar el orden en el que aparecen las instrucciones en cada transacción individual.

Ejemplo:
Sean T1={a1,a2,a3} y T2={b1,b2} son planificaciones válidas:
P1={a1,a2,a3,b1,b2}; P2={a1,b1,a2,a3,b2};
P3={b1,a1,b2,a2,a3} y son planificaciones no válidas:
P1={a2,a1,b1,a3,b2}; P2={a1,b2,b1,a3,a2};
P3={b1,b2,a3,a2,a1}

Seriabilidad de los planes

- Cada transacción (individualmente) preserva la consistencia de la base de datos. De esta forma, la ejecución en secuencia de un conjunto de transacciones también preserva la consistencia de la base de datos. Por tanto, por medio del concepto de SERIABILIDAD (o secuencialidad) se nos facilita la tarea de identificar las ejecuciones en las que se garantiza la consistencia de la base de datos.

- La ejecución de transacciones SERIALIZABLES es equivalente a la ejecución en secuencia de las mismas transacciones.

Seriabilidad de los planes

- Un aplicación es serializable si es equivalente a una planificación secuencial. Existen varias formas de equivalencia:

  - Equivalencia en cuanto a conflictos.
  - Equivalencia en cuanto a vistas.
  - Equivalencia por resultados.

Seriabilidad de los planes

- **Ejecución en serie de transacciones:**
  - Todas las instrucciones pertenecientes a una única transacción aparecen juntas, asegurando así la consistencia.
  - Si tenemos n transacciones, entonces existen n! planificaciones en serie diferentes.

Seriabilidad de los planes

Planificaciones SERIE:

Planificación 1

T1	T2
read(A,a1)	
S ← a1	read(A,a2)
read(B,b1)	A2 ← a2 – 100
S ← S + b1	write(A,a2)
read(C,c1)	read(C,c2)
S ← S + c1	C2 ← c2 + 100
commits	write(C,c2)
	commits

Planificación 2

T1	T2
	read(A,a2)
read(A,a1)	A2 ← a2 – 100
S ← a1	write(A,a2)
read(B,b1)	read(C,c2)
S ← S + b1	C2 ← c2 + 100
read(C,c1)	write(C,c2)
S ← S + c1	commits
commits	

Seriabilidad de los planes

- **Ejecución concurrente de transacciones:**

- La planificación correspondiente no debe ser necesariamente en serie.

  - Número de planificaciones posibles mucho mayor que n!

  - No todas las planificaciones consiguen un estado consistente de la base de datos. Por todo ello son necesarias las **Técnicas de control de concurrencia**.

Seriabilidad en cuanto a conflictos (I)

- Sea P una planificación con dos instrucciones consecutivas, Ii e Ij, de las transacciones Ti y Tj respectivamente, entonces, las instrucciones Ii e Ij están en conflicto si y sólo si existe algún elemento Q accedido por ambas y al menos una de ellas realiza una operación de escritura sobre dicho elemento.

- Intuitivamente, un conflicto entre Ii e Ij fuerza (lógicamente) un orden temporal entre ellos. Si Ii e Ij son consecutivas en una planificación y no están en conflicto, sus resultados continuarían siendo los mismos, incluso si se hubieran intercambiado en la planificación.

Seriabilidad en cuanto a conflictos (II)

- Una planificación P es equivalente en cuanto a conflictos a otra planificación P', si P puede transformarse en P' mediante una serie de intercambios de instrucciones no conflictivas.
  - Si Ii e Ij se refieren a datos diferentes las dos instrucciones pueden intercambiarse sin que ello afecte al resultado de la planificación.
  - Si Ii e Ij se refieren al mismo dato (Q) el orden de las dos instrucciones puede afectar al estado final de la planificación.
    - No importa el orden si Ii = read(Q) e Ij = read(Q).
    - Si importa el orden si:

      Ii = read(Q), Ij = write(Q).

      Ii = write(Q), Ij = read(Q).

      Ii = write(Q), Ij = write(Q).

Seriabilidad en cuanto a conflictos (III)

- Se dice que dos operaciones de un plan **están en conflicto** si satisfacen las tres condiciones siguientes:
  - Pertenecen a distintas transacciones.
  - Tienen acceso al mismo elemento.
  - Al menos una de las operaciones es una operación de escritura.

Seriabilidad en cuanto a conflictos (IV)

- Una planificación P es serializable en cuanto a conflictos si es equivalente en conflictos a una planificación en serie.

- Ejemplos:
- Ejemplo 1: **Planificación serializable**. La planificación P se puede transformar en la planificación serie P' donde T1 sigue a T2, mediante una serie de intercambios de instrucciones no conflictivas. Por lo tanto la planificación P es serializable en cuanto a conflictos.

Seriabilidad en cuanto a conflictos (V)

P:

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

P_1:

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

P_2:

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
write(B)	
	write(A)
	read(B)
	write(B)

P_3:

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
write(B)	
	write(A)
	read(B)
	write(B)

P':

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Planificación serializable

Seriabilidad en cuanto a conflictos (VI)

- Ejemplo 2: **Planificación NO serializable**. La planificación P no es equivalente en cuanto a conflictos a las planificaciones serie P1 y P2, debido a que no se pueden intercambiar instrucciones.

P

T3	T4
read (A)	
	write (A)
write (A)	

P1

T3	T4
read (A)	
write (A)	
	write (A)

P2

T3	T4
	write (A)
read (A)	
write (A)	

Seriabilidad en cuanto a conflictos (VII)

- Ejemplo de una planificación que **no es secuenciable** en cuanto a conflictos:

T3	T4
leer(Q)	
	escribir(Q)
escribir(Q)	

- No se pueden intercambiar instrucciones en la planificación anterior para obtener, o la planificación secuencial <T3, T4>, o la planificación secuencial <T4, T3>.

Seriabilidad en cuanto a conflictos (VIII)

- La planificación siguiente se puede transformar en la planificación en serie, una planificación en serie donde T2 sigue a T1, mediante conjuntos de intercambios de instrucciones no conflictivas. Por lo tanto, la planificación **es serializable en cuanto a conflictos.**

T_1	T_2
leer(A)	
escribir(A)	
	leer(A)
	escribir(A)
leer(B)	
escribir(B)	
	leer(B)
	escribir(B)

Pruebas de seriabilidad en cuanto a confictos (I)

- Dada una planificación P se construye un grafo dirigido, llamado **grafo de precedencia de P: G(V, A)**

  - El conjunto de vértices (V) contiene todas las transacciones que participan en la planificación.

  - El conjunto de aristas (A) conecta los vértices (Ti → Tj) para las cuales se cumple una de las tres condiciones siguientes:

    - Ti ejecuta write(Q) antes que Tj ejecute read(Q).
    - Ti ejecuta read(Q) antes que Tj ejecute write(Q).
    - Ti ejecuta write(Q) antes que Tj ejecute write(Q).

Pruebas de seriabilidad en cuanto a confictos (II)

- Una arista Ti → Tj , en el grafo de precedencia implica que en cualquier planificación serie P' equivalente a P, Ti debe aparecer antes que Tj.

- Si el grafo de precedencia de P no contiene ciclos la planificación P es serializable en conflictos.

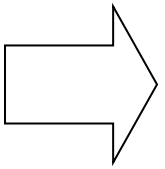
- Orden de serialiabilidad: clasificación topológica que determina un orden lineal consistente con el orden parcial del grafo de precedencia. Pueden obtenerse varios órdenes lineales mediante clasificación topológica.

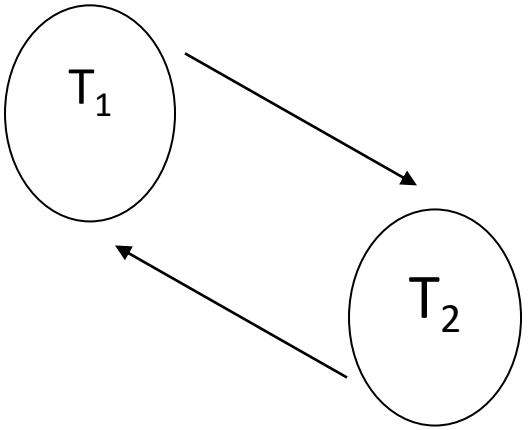
Pruebas de seriabilidad en cuanto a confictos (III)

- Ejemplo:

P_1:

T_1	T_2
read(A)	
	read(A)
write(A)	
read(B)	
	write(A)
write(B)	





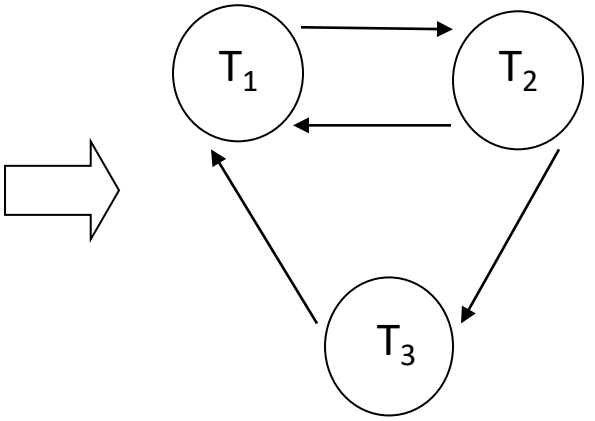
¿Quién precede a quién?

Pruebas de seriabilidad en cuanto a confictos (IV)

- Ejemplo:

P_2:

T_1	T_2	T_3
	read(Z)	
	read(Y)	
	write(Y)	
		read(Y)
		read(Z)
read(X)		
write(X)		
		write(Y)
		write(Z)
	read(X)	
read(Y)		
write(Y)		
	write(X)	



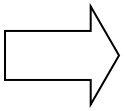
¡Ciclos!

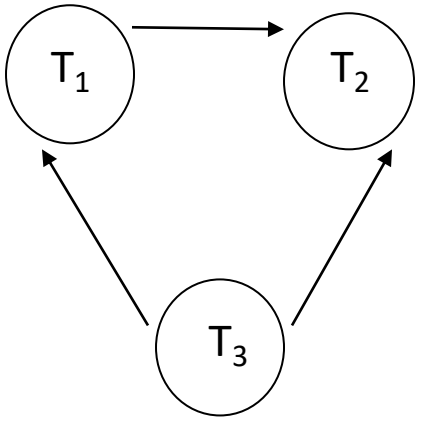
Pruebas de seriabilidad en cuanto a confictos (V)

- Ejemplo:

P_2:

T_1	T_2	T_3
		read (Y)
		read (Z)
read(X)		
write(Y)		
		write(Y)
		write(Z)
	read(Z)	
read(Y)		
write(Y)		
	read(Y)	
	write(Y)	
	read(X)	
	write (X)	





¡Sin ciclos!

Seriabilidad en cuanto a vistas (I)

- Sean S y S´ dos planificaciones con el mismo conjunto de transacciones. S y S´ **son equivalentes en cuanto a vistas** si se cumplen las tres condiciones siguientes:

1. Por cada elemento de datos Q, si la transacción T_i lee el valor inicial de Q en la planificación S, entonces la transacción T_i debe, en la planificación S´, leer también el valor inicial de Q.
2. Por cada elemento de datos Q, si la transacción T_i ejecuta leer(Q) en la planificación S y ese valor fue producido por la transacción T_j entonces la transacción T_i debe leer también, en la planificación S´, el valor de Q que fue producido por la transacción T_j .
3. Por cada elemento de datos Q, la transacción (si existe) que lleva a cabo la operación final escribir(Q) en la planificación S, debe realizar la última operación escribir(Q) en la planificación S´.

- Como se puede ver, la equivalencia en cuanto a vistas también está totalmente sólo basada en lecturas y escrituras.

Seriabilidad en cuanto a vistas (II)

- La equivalencia de vistas se basa en la idea de que, en tanto cada operación lea el resultado de la misma operación de escritura en ambos planes, las operaciones de escritura de ambas transacciones producirán los mismos resultados. Así pues, se dice que las operaciones de lectura perciben la misma vista en ambos planes.

- La última condición garantiza que la operación de escritura final de cada elemento de datos sea la misma en ambos planes, con lo que el estado de la base de datos debería ser el mismo al final de los dos.

Seriabilidad en cuanto a vistas (III)

- Las definiciones de seriabilidad por conflictos y por vistas resultan similares si se cumple una condición conocida como suposición de escritura restringida en todas las transacciones del plan.
- Esta condición establece que cualquier operación de escritura wi(X) en T_i, va precedida por una operación ri(X) en T_i y que el valor escrito por wi(X) en T_i depende sólo del valor de X que ri(X) lee.

- Sin embargo, la definición de seriabilidad por vistas es menos restrictiva que la seriabilidad por conflictos si se supone la escritura no restringida, donde el valor escrito por una operación wi(X) en T_i, puede ser independiente de cualquier valor anterior de la base de datos. Esto se conoce como **escritura ciega**.

- Ej. T_i: r1(X), w2(X); w1(X);W3(X); c1, c2, c3.
- Las operaciones w2(X) y w3(x) son escrituras ciegas, porque ni T_2 ni T_3 leen el valor de X.

Seriabilidad en cuanto a vistas (IV)

- Una planificación S que es serializable en cuanto a vistas, es equivalente en cuanto a vistas a una planificación secuencial.

- Cada planificación secuenciable en cuanto a conflictos es también secuenciable en cuanto a vistas.

Seriabilidad en cuanto a vistas (V)

- Cada operación de lectura en S_2 debe leer el mismo valor que en S_1. Además, el último valor escrito en la BD debe ser el mismo en ambos planes.

Plan S_1

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	

Plan S_2

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)

El plan S_1 es equivalente al plan S_2, por tanto, S_1 es secuenciable en cuanto a vistas.

Seriabilidad en cuanto a vistas (VI)

- La Planificación S_1 **No** es equivalente en cuanto a vistas a la Planificación S_2.

- En la Planificación S_1 el valor de la cuenta A que lee la transacción T_2 lo produce T_1, mientras que esto no ocurre en la planificación S_2

Plan S_1

T_1	T_2
Leer(A) A := A–50 Escribir(A) Leer(B) B:=B+50 Escribir (B)	
	Leer(A) Temp:= A*0.1 A:= A- Temp Escribir(A) Leer(B) B:=B+ Temp Escribir (B)

Plan S_2

T_1	T_2
	Leer(A) Temp:= A*0.1 A:= A- Temp Escribir(A) Leer(B) B:=B+ Temp Escribir (B)
Leer(A) A:= A-50 Escribir(A) Leer (B) B:=B+50 Escribir(B)	

Seriabilidad en cuanto a vistas (VII)

- La Planificación S_1 **es** equivalente en cuanto a vistas a la Planificación S_2 ya que los valores de las cuentas A y B que lee la transacción T_2 lo produce T_1 en ambas planificaciones.

Plan S_1

T_1	T_2
Leer(A)	
A := A–50	
Escribir(A)	
Leer(B)	
B:=B+50	
Escribir (B)	
	Leer(A)
	Temp:= A*0.1
	A:= A- Temp
	Escribir(A)
	Leer(B)
	B:=B+ Temp
	Escribir (B)

Plan S_2

T_1	T_2
Leer(A)	
A:= A-50	
Escribir(A)	
	Leer(A)
	Temp:=A *0.1
	A:= A – Temp
	Escribir (A)
Leer (B)	
B:= B+50	
Escribir (B)	
	Leer (B)
	Temp:= B+Temp
	Escribir (B)

Control de la concurrencia

Ejecuciones concurrentes

Planificaciones

Protocolos basados en bloqueos

Control de concurrencia basado en marcas de tiempo

Granularidad de los ítems de datos

- Un bloqueo es un mecanismo para controlar el acceso concurrente a un elemento de datos.
- Aseguran la seriabilidad exigiendo que mientras una transacción accede a un dato ninguna otra pueda modificarlo.

- Una forma de asegurar la secuencialidad es mediante la **exclusión mutua**, es decir, **mientras que una transacción accede a un elemento de datos, ninguna otra puede modificar dicho elemento**.

Modos de bloqueos

Compartido (lectura)

- Más de una transacción puede adquirir un bloqueo de lectura sobre un recurso.
- Una transacción que tiene bloqueo de lectura sobre un recurso no puede modificarlo, solo leerlo.
- Ninguna otra transacción puede leer el recurso sin bloquearlo en modo de lectura.
- Ninguna otra transacción puede modificar los datos.
- Sirve para garantizar una estabilidad en el valor leído mientras alguna transacción mantenga un bloqueo.

Exclusivo (escritura)

- Sólo la transacción que adquiere el bloqueo puede modificar el dato.
- Ninguna otra transacción puede ni siquiera leer el dato.

- Cuando una transacción accede a un dato, en primer lugar debe bloquearlo.
  - **MODO COMPARTIDO(C).** Los elementos de datos sólo se pueden leer. Un bloqueo de este tipo se solicita con la instrucción bloquear-C.
  - **MODO EXCLUSIVO (X).** El elemento de datos además de leerse se puede escribir. Un bloqueo de este tipo se solicita con la instrucción bloquear-X.

- Todas las transacciones solicitan un bloqueo en el modo adecuado dependiendo del tipo de operaciones que van a realizar.

- Las solicitudes de bloqueo se dirigen al gestor de control de concurrencia. La transacción puede realizar la operación sólo después de que se conceda la solicitud. Si el dato ya está bloqueado por otra transacción en modo incompatible, la transacción debe esperar a que se liberen todos los bloqueos incompatibles.

Reglas de uso de los bloqueos(I)

1. T debe emitir bloquear_lectura(X) o bloquear_escritura(X) antes de ejecutar una operación leer_elemento(X).
2. T debe emitir bloquear_escritura(X) antes de realizar una operación escribir_elemento(X) en T.
3. T debe emitir desbloquear(X) una vez completadas todas las operaciones leer_elemento(X) y escribir_elemento(X).
4. Si T ya posee un bloqueo, compartido o exclusivo, sobre X no emitirá bloquear_lectura(X) ni bloquear_escritura(X).

   esta regla puede permitir excepciones: mejora y reducción de bloqueos
5. T no emitirá desbloquear(X) salvo si posee un bloqueo, compartido o exclusivo, sobre X.

Protocolos basados en bloqueos

- **Matriz de compatibilidad de bloqueos (I)**
- Dado un conjunto de modos de bloqueo, se puede definir sobre ellos una función de compatibilidad de la manera siguiente:
- Usamos A y B para representar dos modos de bloqueo arbitrarios. Supóngase que la transacción T_i solicita un bloqueo en modo A sobre el elemento Q, en el que la transacción T_j ($T_i \neq T_j$) posee actualmente un bloqueo de modo B.
- Si a la transacción T_i se le puede conceder un bloqueo sobre Q a pesar de la presencia del bloqueo de modo B, entonces se dice el modo A es compatible con el modo B.
- Un elemento comp(A,B) de la matriz tiene el valor cierto si y sólo si el modo A es compatible con el modo B.

	C	**X**
C	Cierto	Falso
X	Falso	Falso

Protocolos basados en bloqueos

- **Matriz de compatibilidad de bloqueos (II)**

- A una transacción se le puede garantizar un bloqueo en un elemento si el bloqueo solicitado es compatible con los bloqueos que ya tengan otras transacciones sobre ese mismo elemento.

- Cualquier número de transacciones puede tener bloqueos compartidos sobre un elemento, pero si una de ellas tiene uno exclusivo sobre un determinado elemento, ninguna otra puede tener ningún otro bloqueo sobre dicho elemento.

- Si no se puede garantizar un bloqueo, la transacción que lo solicita tiene que esperar hasta que los bloqueos incompatibles que tienen otras transacciones se hayan liberado. A continuación se autoriza el bloqueo.

- Cuando una transacción T solicita un bloqueo…
  - Si el elemento no ha sido ya bloqueado por otra transacción, se le concede el bloqueo.
  - Si el elemento sí está bloqueado, el SGBD determina si la solicitud es compatible con el bloqueo existente:
    - Si se pide un bloqueo compartido sobre un elemento que ya tiene un bloqueo compartido, el bloqueo será concedido a T.
    - En otro caso, T debe esperar hasta que se libere el bloqueo existente.
- Una transacción que obtiene un bloqueo lo mantiene hasta que lo libera explícitamente o termina (commit o rollback)
  - Sólo cuando se libera un bloqueo exclusivo los efectos de la escritura serán visibles para las demás transacciones.

Fallos en los protocolos basados en bloqueos (I)

- Considerar la planificación parcial

T_3	T_4
bloquear-X(B)	
leer(B)	
$B := B - 50$	
escribir(B)	
	bloquear-C(A)
	leer(A)
	bloquear-C(B)
bloquear-X(A)	

- Ni T_3 ni T_4 pueden progresar — la ejecución de **bloquear-X**(B) ocasiona que T_4 espere a que T_3 libere su bloqueo sobre B, mientras que la ejecución de **bloquear-X**(A) ocasiona que T_3 espera a que T_4 libere su bloqueo sobre A.
- Esta situación se denomina **interbloqueo**.

  Para manejar un interbloqueo uno de los dos, T_3 o T_4, debe retroceder y liberar sus bloqueos.

Fallos en los protocolos basados en bloqueos (II)

- En la mayoría de los protocolos de bloqueo se puede producir un interbloqueo potencial.
- También es posible que se produzca la inanición si el gestor de control de concurrencia se ha diseñado defectuosamente. Por ejemplo:
  - Puede que una transacción esté esperando un bloqueo exclusivo sobre un elemento determinado mientras que una secuencia de transacciones diferentes solicitan y obtienen la autorización de bloqueo compartido sobre el mismo elemento.
  - La misma transacción retrocede repetidamente debido a los interbloqueos.
- El gestor de control de concurrencia se puede diseñar para impedir que se produzca la **inanición**.

Fallos en los protocolos basados en bloqueos (III)

- **Inanición**.
  - Una transacción sufre inanición cuando es seleccionada para ser abortada (víctima) sucesivamente: nunca termina su ejecución.
    - Es similar al bloqueo indefinido.
  - La solución es asignar prioridades más altas a las transacciones abortadas varias veces, para no ser siempre las víctimas.

El protocolo de bloqueo en dos fases (I)

- Es necesario seguir un protocolo adicional que indique dónde colocar las operaciones de bloqueo y desbloqueo dentro de las transacciones.
- El más conocido es el Bloqueo en Dos Fases (B2F).
- Una transacción T sigue el protocolo de bloqueo en **dos fases** si **todas las operaciones de bloqueo preceden a la primera operación de desbloqueo.**
  - De este modo, podemos ver T dividida en dos fases:
    - **Fase de expansión (o crecimiento)**
      - T puede **adquirir bloqueos**.
      - T **no** puede **liberar ningún bloqueo**.
    - **Fase de contracción**
      - T puede **liberar bloqueos** existentes.
      - T **no** puede **adquirir** ningún bloqueo.

El protocolo de bloqueo en dos fases (II)

T_3	T_4
bloquear -X(B)	
leer(B)	
B:=B-50	
escribir (B)	
	bloquear-C(A)
	leer (A)
	bloquear-C(B)
bloquear -X(A)	

- Sí Transacciones de dos fases

T_3	T_4
bloquear -X(B)	
leer(B)	
B:=B-50	
escribir (B)	
desbloquear (B)	
	bloquear-C(A)
	leer (A)
	desbloquear(A)
	bloquear-C(B)
	leer (B)
	desbloquear(B)
	visualizar (A+B)
bloquear -X(A)	
leer(A)	
A:=A+50	
escribir (A)	
desbloquear (A)	

- No Transacciones de dos fases

El protocolo de bloqueo en dos fases (III)

- Si toda transacción de una planificación sigue **el protocolo de bloqueo en dos fases**, entonces la **planificación es serializable**.
- Ventaja
  - Ya no es necesario comprobar la serializabilidad de las planificaciones.

- Se puede probar que las transacciones se pueden secuenciar en el orden de sus puntos de bloqueo (es decir, el punto de la planificación en el cual la transacción obtiene su bloqueo final).

El protocolo de bloqueo en dos fases (IV)

- El protocolo de bloqueo de dos fases no asegura la ausencia de interbloqueos. Las transacciones T3 y T4 son de dos fases pero se produce interbloqueo.

T_3	T_4
bloquear -X(B)	
leer(B)	
B:=B-50	
escribir (B)	
	bloquear-C(A)
	leer (A)
	bloquear-C(B)
bloquear -X(A)	

El protocolo de bloqueo en dos fases (V)

- El retroceso en cascada puede ocurrir en el bloqueo de dos fases.
- **Retroceso en cascada** – Cuando se ha producido un fallo hay veces que es necesario retroceder varias transacciones para recuperar correctamente el estado previo de un fallo en una transacción T_i. Tales situaciones ocurren si las transacciones leen datos que ha escrito T_i.
- La transacción T_{10} escribe un valor que lee la transacción T_{11}. La transacción T_{11} escribe un valor que lee la transacción T_{12}. Si falla T_{10}, se debe retroceder T_{10}, puesto que T_{11} depende de T_{10}, se debe retroceder T_{11}. Puesto que T_{12} depende de T_{11}, de debe retroceder T_{12}.

T_{10}	T_{11}	T_{12}
leer (A)		
leer (B)		
escribir (A)		
	leer (A)	
	escribir (A)	
		leer (A)

El protocolo de bloqueo en dos fases (VI)

- Ejemplo de retroceso en cascada en planificaciones en dos fases.

T_5	T_6	T_7
bloquear-X (A) leer (A) bloquear-C (B) leer(B) escribir(A) desbloquear (A)		
	bloquear-X (A) leer (A) escribir(A) desbloquear (A)	
		bloquear-C (A) leer (A)

Cada transacción sigue el protocolo de bloqueo en dos fases pero un fallo de T_5 después del paso leer(A) de T_7 lleva un retroceso en cascada de T_6 y T_7.

El protocolo de bloqueo en dos fases (VII)

- Se pueden evitar por medio de una modificación del protocolo denominado **protocolo de bloqueo estricto de dos fases.** En él, **una transacción T no libera ninguno de sus bloqueos exclusivos antes de confirmarse o de abortar**.

- Este requisito asegura que todo dato que escribe una transacción no comprometida esté bloqueado en modo exclusivo hasta que la transacción se complete, evitando que ninguna otra transacción lea el dato.

El protocolo de bloqueo en dos fases (VIII)

- Otra variante del bloqueo de dos fases es el **protocolo de bloqueo riguroso de dos fases.** Este protocolo es incluso más estricto: en él una transacción T no libera ninguno de sus bloqueos (exclusivos o compartidos) **hasta después** de confirmarse o de abortar.

- En este protocolo las transacciones se pueden secuenciar en el orden en que se comprometen.

El protocolo de bloqueo en dos fases (VIII)

- **Diferencia entre el bloqueo estricto y el riguroso**:

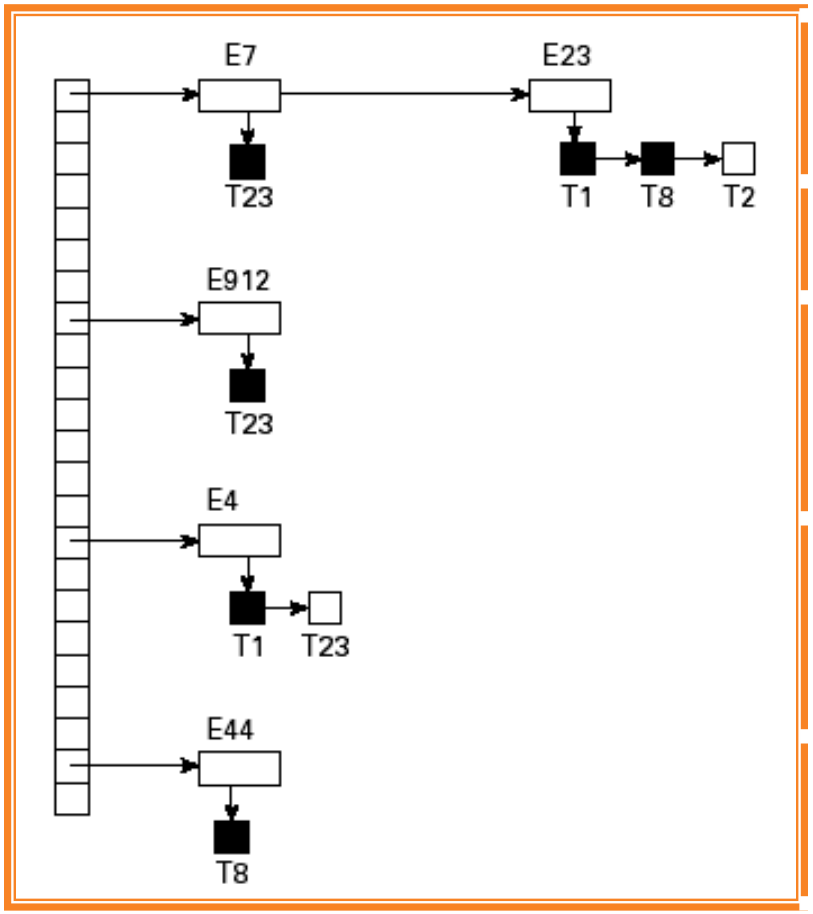
- En el **bloqueo estricto** se bloquean todos los elementos **antes** de iniciarse, por lo que una vez que se ha iniciado la transacción se encuentra en su fase de contracción.

- En el **bloqueo riguroso** no se desbloquea ninguno de sus elementos hasta **después** de terminar (sea confirmándose o abortando) por lo que la transacción se encuentra en sus fase de expansión hasta que finaliza.

Gestor de bloqueos

- Un Gestor de bloqueos puede ser implementado como un proceso separado que recibe peticiones de bloqueo y desbloqueo.
- El gestor de bloqueo responde a un mensaje de **petición** con un mensaje de **concesión**.
- Una transacción que ha solicitado un bloqueo debe esperar hasta recibir el mensaje de concesión o un mensaje de retroceso.
- El gestor de bloqueo mantiene una **tabla de bloqueos** que almacena los bloqueos concedidos y las peticiones solicitadas.
- La tabla de bloqueos es una tabla hash en memoria cuyas entradas corresponden a los elementos de datos bloqueados.
- Por cada elemento de datos hay una lista enlazada con registros que representan solicitudes concedidas (pero no liberadas) y solicitudes pendientes.

Tabla de bloqueo (I)



Los **rectángulos negros** indican **bloqueos concedidos**, los **blancos** indican las solicitudes **en espera**.

La tabla de bloqueos también registra el tipo de bloqueo concedido o solicitado.

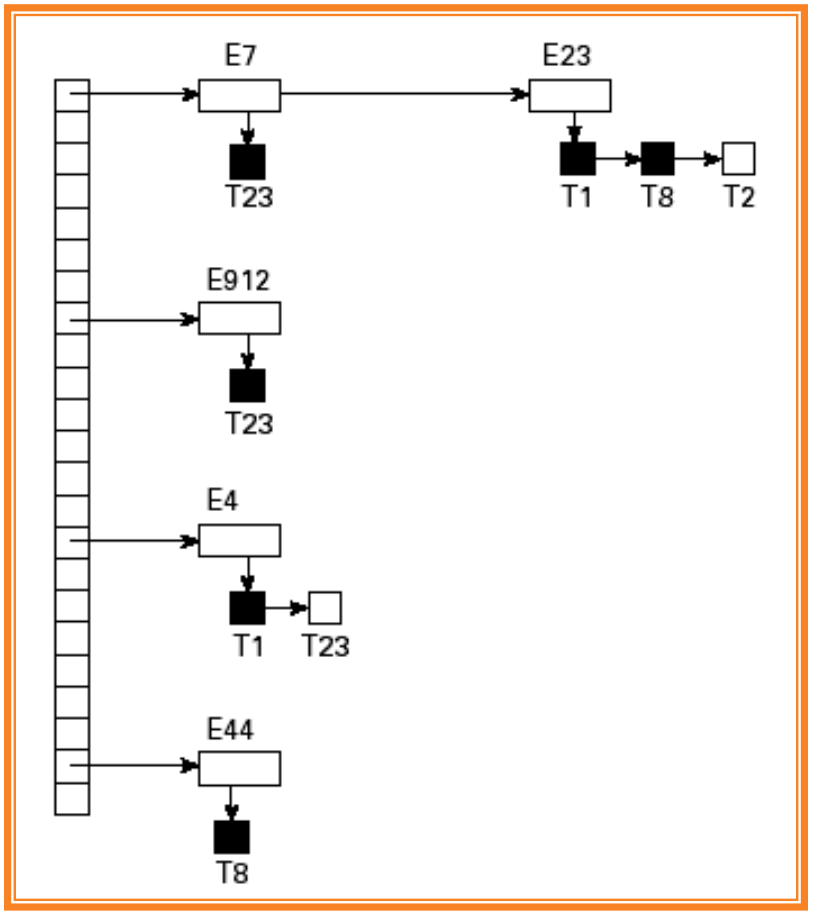
Una nueva solicitud se añade al final de la cola de solicitudes para el elemento de datos, y se concede si es compatible con todos los bloqueos anteriores.

Los desbloqueos solicitados dan lugar a que la solicitud sea borrada y después se comprueban las solicitudes para ver si ahora se pueden conceder.

Si la transacción aborta se borran todas las solicitudes de la transacción, en espera o concedidas.

El gestor de bloqueos, para implementar esto eficientemente, puede mantener una lista de los bloqueos almacenados por cada transacción.

Tabla de bloqueo (II)



La tabla contiene bloqueos para cinco elementos de datos E4,E7,E23,E44 y E912.

A T23 se le han concedido bloqueos sobre E912 y E7 y está esperando para bloquear E4.

Control de la concurrencia

Ejecuciones concurrentes

Planificaciones

Protocolos basados en bloqueos

Control de concurrencia basado en marcas de tiempo

Granularidad de los ítems de datos

Protocolos basados en marcas temporales (I)

- Una marca de tiempo es un identificador único creado por el gestor de la BD para identificar una transacción.
- Los valores de las marcas de tiempo se asignan por el orden en el cual las transacciones son recibidas por el sistema.
- La marca de tiempo puede ser considerada como el inicio de una transacción.
- Las marcas de tiempo pueden ser generadas de varias formas:
  - Utilizando un **contador lógico** que se incrementa cada vez que su valor se asigna a una transacción. En este esquema las marcas de tiempo de las transacciones están numeradas 1, 2, 3 etc .

    Un contador de computador tiene un valor máximo finito, por lo que el sistema deberá restablecer periódicamente a cero el contador cuando haya un lapso corto de tiempo en el que ninguna transacción se esté ejecutando.
  - Utilizando **la hora del reloj del sistema** asegurando que no se generan dos marcas de tiempo durante el mismo tic de reloj.

Protocolos basados en marcas temporales (II)

- El método consiste en ordenar las transacciones según sus marcas de tiempo.

  - Si un transacción T_i ingresa al sistema recibe una marca de tiempo $MT(T_i)$.

  - Si luego entra una transacción T_j el sistema genera una marca de tiempo $MT(T_j)$, tal que $MT(T_i) < MT(T_j)$.

  - El sistema debe asegurar que se produzcan planificaciones equivalentes a una planificación secuencial en la cual T_i aparece antes que T_j.

Protocolos basados en marcas temporales (III)

Ordenación por marcas de tiempo

- Se permite que las transacciones se ejecuten libremente.

- Hay que asegurarse de que por cada ítem accedido por más de una transacción en el plan, el orden de acceso no viole la secuencialidad, para lo cual se asocian a cada ítem X de la BD dos marcas de tiempo:

  - MT_lectura(X): Es la mayor de todas las marcas de tiempo de las transacciones que han leído con éxito el elemento X; es decir MT lectura(X) = MT(T), donde T es la transacción más reciente que ha leído X de manera satisfactoria.
  - MT_escritura(X): Es la mayor de todas las marcas de tiempo de las transacciones que han escrito con éxito el elemento X; esto es, MT escritura (X) = MT (T), donde T es la transacción más reciente que ha escrito X de manera satisfactoria.

Protocolos basados en marcas temporales (IV)

- Estas marcas temporales se actualizan cada vez que se ejecuta una nueva operación leer o escribir.

- El algoritmo de control de concurrencia debe comprobar si la ordenación por marcas de tiempo es violada cuando la transacción **T** intenta realizar una operación de lectura o escritura.

Protocolos basados en marcas temporales (V)

La transacción T intenta una operación Escribir_item(X)	
a) Si MT_lectura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna transacción posterior a T ya leyó el ítem X antes de que T realice la escritura. Se viola la secuencialidad.
b) Si MT_escritura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna transacción posterior a T ya escribió el valor de X. Se evita el Escribir_item(X) de T para que no modifique el valor correcto con uno obsoleto.
SI no se cumplen a) y b)	Ejecutar la operación Escribir_item(X) de T y hacer MT_escritura ← MT(T). La Transacción T llega a tiempo para realizar la operación
La transacción T intenta una operación Leer_item(X)	
a) Si MT_escritura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna Transacción posterior a T ya sobrescribió el valor de X que T debió leer.
SI no se cumple a)	Ejecutar la operación Leer_item(X) de T y asignar a MT_lectura (X) el valor más grande entre MT(T) y MT_lectura(X).

Protocolos basados en marcas temporales (VI)

- El algoritmo de ordenación por marcas de tiempo detecta operaciones en conflicto que ocurren en orden incorrecto y rechaza la más reciente de las dos abortando la transacción que la emitió.

Protocolos basados en marcas temporales (VII)

- El protocolo de ordenación por marcas temporales garantiza la secuencialidad en cuanto a conflictos.
- **Se asegura la ausencia de interbloqueos** ya que **ninguna transacción queda bloqueada antes de realizar alguna operación.**
- Existe la posibilidad de inanición ya que transacciones largas puede sufrir de repetidos reinicios.
  - Si se detecta el reinicio repetido alguna transacción, el sistema debe bloquear las transacciones conflictivas para permitir que la transacción reiniciada termine.

Variaciones del protocolo de marcas de tiempo

- El protocolo de ordenación por marcas temporales garantiza la secuencialidad ya que todos los arcos del grafo de precedencia son del tipo:

Por lo tanto, no se producirán ciclos en el grafo de precedencia

- El protocolo asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar.
- Pero puede que la planificación no esté libre de cascadas e incluso, es posible que no sea recuperable.

Recuperabilidad y libertad de cascada

- **Problema** del protocolo de ordenación por marcas temporales:
  - Supóngase que T_i aborta, pero T_j ha leído un elemento de datos escrito por T_i.
  - A continuación T_j debe abortar; si a T_j se le ha permitido completar antes, la planificación no es recuperable.
  - Además, cualquier transacción que haya leído un elemento de datos escrito por T_j debe abortar.
  - Esto puede llevar a un retroceso en cascada, es decir, a una cadena de retrocesos.
- **Solución:**
  - Una transacción se estructura de modo que todas sus escrituras se llevan a cabo al final de su procesamiento.
  - Todas las escrituras de una transacción forman una acción atómica, es decir, no se puede ejecutar ninguna transacción mientras que se está escribiendo una transacción.
  - Una transacción que aborta se reinicia con una marca temporal nueva.

Regla de escritura de Thomas

- Versión modificada del protocolo de ordenación por marcas temporales en el que las operaciones escribir obsoletas se pueden ignorar bajo determinadas circunstancias.

- Cuando T_i intenta escribir el elemento de datos Q, si $MT(T_i)$ < marca_temporal-E(Q), entonces T_i está intentando escribir un valor obsoleto de {Q}. Por lo tanto, en vez de retroceder T_i como habría hecho el protocolo de ordenación por marcas temporales, esta operación {escribir} se puede ignorar.

- En cualquier otro caso este protocolo se comporta igual que el protocolo de ordenación por marcas temporales.

- La regla de escritura de Thomas permite una gran concurrencia potencial.

Control de la concurrencia

Ejecuciones concurrentes

Planificaciones

Protocolos basados en bloqueos

Control de concurrencia basado en marcas de tiempo

Granularidad de los ítems de datos

Granularidad de los ítems de datos

- Toda técnica de control de concurrencia supone que la base de datos está constituida por un conjunto de elementos de datos con nombre.

- Normalmente, un elemento de datos será uno de estos:
  - un valor de campo de un registro de la BD
  - un registro de la BD
  - una página (uno o varios bloques de disco)
  - un fichero
  - la BD completa

- Granularidad = tamaño del elemento de información
  - Granularidad fina ➡ elementos de tamaño pequeño
  - Granularidad gruesa ➡ elementos grandes

Granularidad de los ítems de datos

- Elección del tamaño adecuado del elemento de datos
- En el contexto de los métodos de bloqueo, el tamaño del elemento de datos afecta al grado de concurrencia:

  ⬇ tamaño(elemento) ⇨ ⬆ Grado de concurrencia

  Y también...

  ⇨ ⬆ número de elementos en la BD

  ⇨ ⬆ carga de trabajo para la gestión de bloqueos, y

  ⇨ ⬆ espacio ocupado por la información de bloqueo

- Pero... ¿Cuál es el tamaño adecuado para los elementos?

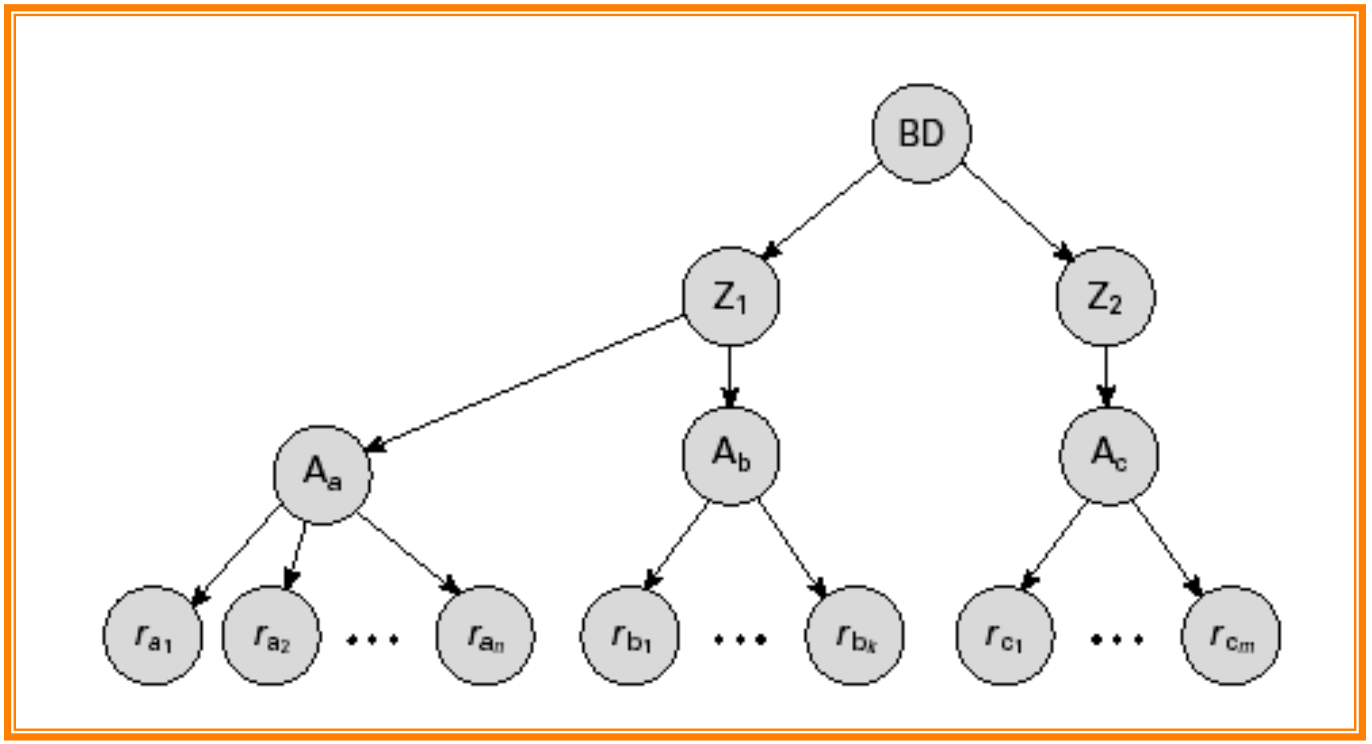
  Pues depende de la naturaleza de las transacciones:

  – Si una T representativa accede a pocos registros
    - elegir granularidad de registro
  – Si T accede a muchos registros de un mismo fichero
    - elegir granularidad de página o de fichero

Granularidad de los ítems de datos

- La granularidad determina el nivel de bloqueo de datos.
  - Pero también produce mucha sobrecarga en tiempo de ejecución y de espacio ocupado con información sobre los bloqueos en el Gestor de Concurrencia.
- Normalmente los DMBS no soportan el bloqueo de campos.
- Los datos en los niveles de granularidad disponibles pueden ser representados por un **árbol de granularidad**.
  - Cuando se bloquea una parte del árbol, automáticamente se bloquean los niveles descendientes.
  - De la misma forma no se puede bloquear un nodo del árbol si alguno de sus nodos ascendientes ya se encuentra bloqueado por otra entidad.
    - Se recorre el árbol desde la raíz hacia el nodo a bloquear si se encuentra un nodo ascendiente bloqueado en modo incompatible entonces se retrasa la petición de bloqueo.

Ejemplo de jerarquía de granularidad



El nivel más alto en la jerarquía del ejemplo es la base de datos completa.

Los niveles inferiores son del tipo *área*, *archivo* y *registro*, por ese orden.

INTEGRIDAD SEMÁNTICA Y CONFIDENCIALIDAD

Integridad semántica y confidencialidad

Introducción

Integridad semántica

Control de accesos

Cifrado

Introducción

En este tema estudiaremos las técnicas empleadas para proteger la base de datos contra personas que no estén autorizadas para acceder a una parte de la base de datos o a toda.

La seguridad de las bases de datos es un área amplia que abarca diferentes temas:

- **Cuestiones éticas y legales** relativas al derecho de tener acceso a cierta información.
- **Cuestiones de política gubernamental, institucional o corporativa,** relacionadas con las clases de información que no deben estar disponibles para el público.
- **Cuestiones relacionadas con el sistema**, como los niveles del sistema en los que deben realizarse las diversas funciones de seguridad; por ejemplo, si una determinada función de seguridad se debe tratar a nivel de hardware, a nivel de sistema operativo o a nivel del SGBD.
- La necesidad en algunas organizaciones de identificar **múltiples niveles de seguridad y de clasificar los datos y los usuarios según estos niveles**: Ej. (máximo secreto, secreto, confidencialidad)

Introducción

En un sistema de bases de datos multiusuario, el SGBD debe proveer de técnicas que permitan a ciertos usuarios o grupos de usuarios tener acceso a determinadas porciones de una base de datos sin tener acceso al resto.

Otro problema de seguridad consiste en controlar el acceso a una base de datos estadística, la cual sirve para proporcionar información estadística o resúmenes de valores basados en diversos criterios. La seguridad en las bases de datos estadísticas debe cuidar que la información sobre distintos individuos no sea accesible.

Otro aspecto de seguridad es el cifrado de datos, que sirve para proteger datos confidenciales, tales como números de tarjetas de crédito. Así mismo, el cifrado también puede proveer protección adicional a secciones confidenciales de una base de datos.

Los datos se codifican mediante algún algoritmo de cifrado que posteriormente habrá que descifrar.

Introducción

Puntos de vista en aspectos de seguridad de los datos de una base de datos:

- **Subsistemas de recuperación:** tratan de conservar la integridad de la información de la base de datos a pesar de los fallos del sistema.

- **Subsistemas de control de concurrencia:** tratan de proteger la base de datos contra anomalías potenciales que puede producir la ejecución simultánea de transacciones.

- Aborda el problema de la seguridad desde dos puntos de vista:
  - **Integridad semántica:** garantiza que la información que contiene la base de datos se corresponde con la realidad (correcta, válida o precisa).

  - **Control de accesos:** garantiza que el acceso a los datos sólo se permite a personas autorizadas.

Integridad semántica y confidencialidad

Introducción

Integridad semántica

Control de accesos

Cifrado

Integridad semántica

Restricciones de integridad

- Medio para asegurar que los cambios que los usuarios autorizados realizan a la base de datos no producen pérdida en la consistencia de los datos.
- Protegen la base de datos contra cambios accidentales.

Subsistema de integridad

- Detecta y corrige las operaciones semánticamente inconsistentes (las que violan las restricciones definidas por el DBA sobre los dominios, atributos, etc.).

Condiciones de integridad semántica que se prueban fácilmente (incluidas en la mayoría de los SGBD):

- Restricciones de dominio: cada atributo definido sobre un dominio de valores.
- Posibilidad de valores nulos para los atributos.
- Dependencias funcionales: limita las relaciones legales posibles.
- Restricciones de integridad referencial.

- Otras posibilidades especiales para expresar condiciones de integridad son las siguientes:

  - **AFIRMACIONES**

    Predicados que expresan una condición que debe satisfacer siempre la base de datos.

    Para cada afirmación el sistema prueba su validez. Si la afirmación es válida, cualquier modificación futura a la base de datos sólo se permite si no viola la afirmación.

    Ejemplo: (calif_j >= 0 and calif_j <= 10)

- **DISPARADORES**
  - Sentencia que el sistema ejecuta automáticamente como un efecto secundario de una modificación de la base de datos.

  - Se deben especificar las condiciones bajo las cuales se va a ejecutar un disparador y las acciones que se van a tomar cuando se ejecute.

  - Ejemplo: trigger que se ejecuta cuando se está tratando de modificar o insertar un valor para calif_j no comprendido entre 0 y 10.

Integridad semántica y confidencialidad

Introducción

Integridad semántica

Control de accesos

Cifrado

- Subsistemas de control: garantizan que a la base de datos sólo acceden usuarios autorizados.

- Mecanismos de control:

  - **VISTAS** (esquemas externos o subesquemas): permiten ocultar datos a los usuarios.

  - **CONTROL DE ACCESOS** (lenguaje de control de datos): permite asignar distintos privilegios de acceso y modificación a los usuarios.

- **Ventana o vista**

Selección predefinida de datos de una o más tablas, tratada como una tabla y construida dinámicamente.

- Ventajas que supone el uso de vistas:
  - SEGURIDAD: ocultan información a los usuarios.
  - FACILIDAD: selecciones predefinidas liberan al usuario de repetir condiciones de búsqueda.
  - INDEPENDENCIA: la tabla base puede cambiar.
- **Control de accesos**
  - Asignación de distintos privilegios de acceso a los usuarios

- **Vista**

- Una vista se puede definir como una tabla derivada de otras tablas. Estas otras pueden ser tabla de base o vistas previamente definidas.

- Las vistas no necesariamente existen de forma física, se les considera como tablas virtuales, en contraste con las tablas de base cuyas tuplas se almacenan realmente en la base de datos.

- **Vista**
- La instrucción para crear una vista es CREATE VIEW.

- La vista recibe un nombre de tabla (virtual) (o nombre de vista), una lista de nombre de atributos y una consulta para especificar el contenido de la vista. Si ninguno de los atributos de la vista es el resultado de aplicar funciones u operaciones aritméticas, no tendremos que especificar nombres de atributos para la vista, pues serán los mismos que los nombres de los atributos de las tablas de definición.

- v1: CREATE VIEW trabaja-en1
  AS SELECT nombre, apellido, nombrep, horas
  FROM empleado, proyecto, trabaja-en
  WHERE nss=nsse AND np=numerop;

- v2: CREATE VIEW info-depto(nombre-depto,
  num-de-emps, sal-total)
  AS SELECT nombred, COUNT(*), SUM(salario)
  FROM departamento, empleado
  WHERE numerod=nd
  GROUP BY nombred;

- En v1: no especificamos nuevos nombres de atributos para la vista trabaja-en1, aunque podríamos haberlo hecho, en este caso, trabaja-en1 hereda los nombres que tienen los atributos de las tablas empleado, proyecto y trabaja-en.

- La vista v2: especifica explícitamente nuevos nombres de atributos para la vista info-depto, por medio de una correspondencia uno a uno entre los atributos especificados en la claúsula CREATE VIEW y los especificados en la claúsula SELECT de la consulta que define la vista.

- Podríamos especificar consultas SQL en términos de una vista o tabla virtual de la misma manera que especificamos consultas en términos de tablas base.

- Por ejemplo si queremos obtener el apellido y el nombre de pila de todos los empleados que trabajan en el "Proyecto X" , podemos usar la vistas trabajaen-1 y especificar la consulta:

- CV1: SELECT nombre, apellido
  FROM trabaja-en1
  WHERE nombrep="ProyectoX";

- La misma consulta requeriría la especificación de dos reuniones si se especificara sobre las relaciones de base.

- Una de las principales ventajas de las vista es que simplifican la especificación de ciertas consultas, además de que pueden servir como mecanismos de seguridad y autorización.

- Las vistas sirven como mecanismo de autorización.

- Ej: Supóngase que el propietario A de una relación R desea que otra cuenta B pueda leer únicamente ciertos campos de R.

  - A puede crear una vistas V de R que incluya sólo esos atributos y después conceder el privilegio SELECT para V.

  - Lo mismo se aplica cuando se desea limitar a B la lectura de únicamente ciertas tuplas de R.

  - Se puede crear una vista V' definiéndola por medio de una consulta que seleccione sólo las tuplas de R que A desee poner al alcance de B.

- **Tipos de usuarios según los privilegios que posee**
  - Clasificación de los usuarios
    - **DBA:** posee privilegios sobre todas las operaciones del sistema, incluida la de conceder privilegios y establecer usuarios.
    - **Usuarios con privilegios** de crear, borrar o modificar objetos, que además pueden conceder privilegios a otros usuarios sobre los objetos que han creado. Estos privilegios permiten al usuario modificar el esquema de la base de datos, dependiendo del tipo de autorización: de índice, de recursos, de alteración o de eliminación.
    - **Usuarios con derecho a consultar** o actualizar datos de la base de datos, pero sin derecho a crear, borrar o modificar objetos. Un usuario puede tener asignados todos, ninguno o una combinación de los diferentes tipos de autorización (lectura, inserción, actualización y eliminación de información).

- El DBA es la autoridad central que gestiona un sistema de bases de datos. Entre las obligaciones del DBA están conceder privilegios a los usuarios que necesitan usar el sistema, clasificar los usuarios y los datos de acuerdo con la política de la organización.

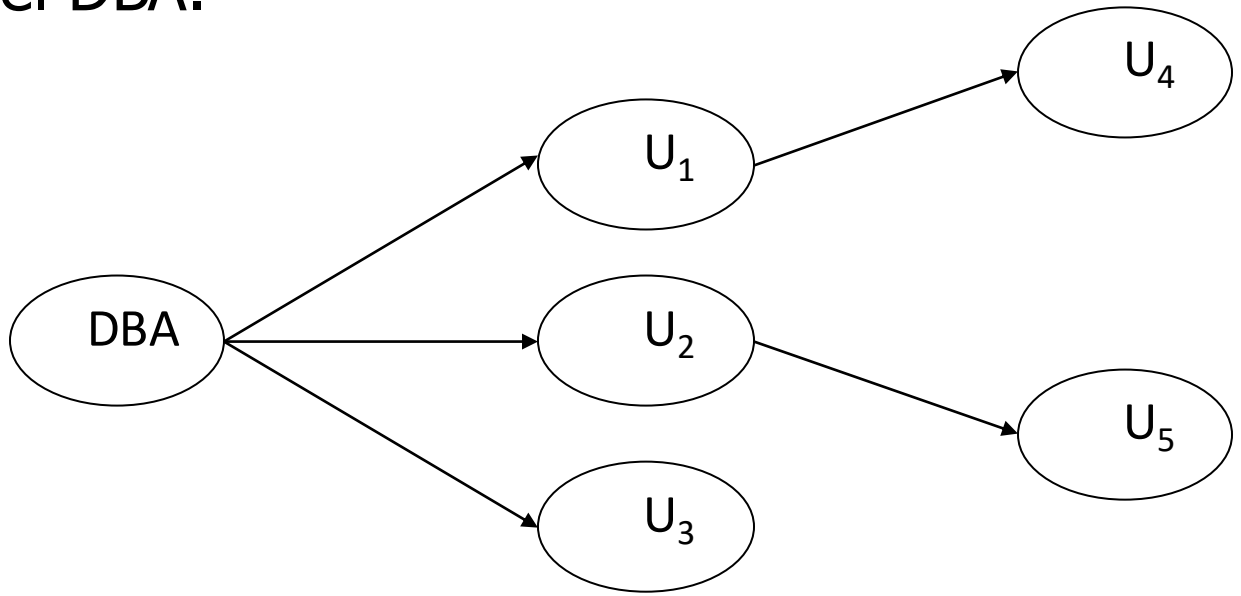
- El DBA tiene una cuenta de ABD en el SGBD, a veces denominada cuenta del sistema o de superusuario, que confiere capacidades extraordinarias no disponibles para las cuentas y los usuarios ordinarios de la base de datos (cuenta raíz (root), o superusuario (superuser)).

- Un usuario al que se le ha concedido una forma de autoridad puede tener permiso para conceder esa autoridad a otros usuarios.

- Las instrucciones privilegiadas del ABD incluyen órdenes para conceder y revocar privilegios a cuentas individuales, usuarios, o grupos de usuarios y para efectuar las siguientes tipos de acciones:

- **Creación de cuentas**: esta acción crea una nueva cuenta y contraseña para un usuario o grupo de usuarios, con el fin de que pueden tener acceso al SGBD.
- **Concesión de privilegios**: esta acción permite al ABD conceder ciertos privilegios a ciertas cuentas.
- **Revocación de privilegios**: esta acción permite al ABD revocar (cancelar) ciertos privilegios que se habrían concedido previamente a distintas cuentas.
- **Asignación de niveles de seguridad**: esta acción consiste en asignar cuentas de usuario al nivel de clasificación de seguridad apropiado.

- **Grafo de autorización**
- Representa la transferencia de autorización de un usuario a otro.
- Para asegurar que la autorización entre usuarios pueda ser anulada posteriormente se debe exigir que todas las aristas del grafo de autorización sean parte de una ruta que tenga su origen en el DBA.



- **DCL Lenguaje de control de datos**
- Permite definir diferentes niveles de acceso sobre diferentes subconjuntos de las tablas de la base de datos.
- Sentencias SQL para definir derechos de acceso a los usuarios:
- Para **conceder privilegios**:
  - **grant <privilegio> on <objeto> to <usuario> [with grant option];**
- Para **anular privilegios**:
  - **revoke <privilegio> on <objeto> from <usuario>;**

- Todos los derechos controlados por grant y revoke son almacenados en el catálogo.
- Privilegios que se pueden aplicar sobre tablas y vistas: select, update, insert, delete.
  - all privileges indica colección de todos los privilegios.
  - public indica que se concede el derecho a todos los usuarios.

- Siempre que el propietario A de una relación R concede el privilegio de R a otra cuenta B, el privilegio puede darse a B con opción de concesión (GRAN OPTION) o sin ella.

- Si se da con esta opción, significa que B también podra conceder este privilegio de R a otras cuentas.

- Supongamos que A concede a B la opción de concesión (GRANT OPTION) y que B concede luego el privilegio de R a una tercera cuenta C, también con la opción de concesión.

- De esa manera los privilegios de R se pueden propagar a otras cuentas sin que lo sepa el propietario de R.

- Si la cuenta propietario de A revoca el privilegio concedido a B, el sistema deberá revocar automáticamente todos los privilegios que B propagó basándose en ese privilegio.

- Supongamos que el administrador de la base de datos crea cuatro cuentas: A1,A2,A3,A4 y desea que sólo A1 pueda crear relaciones de base, el administrador de la base deberá emitir la siguiente instrucción:
  - GRANT CREATETAB to A1;
  - El privilegio CREATETAB (crear tabla) confiere a la cuenta A1 la capacidad de crear nuevas tablas (relaciones de base) de la base de datos, y es por tanto un privilegio de cuenta.
  - También lo podemos hacer según las versión del SQL que manejemos a través de:
    - CREATE SCHEMA ejemplo AUTORIZATION A1;
  - Ahora la cuenta A1 puede crear tablas dentro del esquema llamado ejemplo.

- Supongamos que A1 crea las dos relaciones de base empleado y departamento, con ello A1, es propietario de estas dos relaciones y por lo tanto tiene todos los privilegios de relación para ambas.

- A continuación, supongamos que:

- La cuenta A1 desea conceder a la cuenta A2 el privilegio de insertar y eliminar tuplas en estas relaciones.

- Sin embargo A1 no desea que A2 pueda propagar estos privilegios a cuentas adicionales.

- A1 puede emitir la siguiente instrucción:

    - GRANT INSERT, DELETE ON empleado, departamento TO A2;

- La cuenta propietario A1 de una relación tiene automáticamente la opción de concesión (GRANT OPTION), y puede conceder privilegios para la relación a otras cuentas.

- Sin embargo, la cuenta A2 no puede conceder privilegios INSERT y DELETE para las tablas empleado y departamento, porque no recibió la concesión en la instrucción anterior.

- Ahora supongamos que A1 desea permitir a la cuenta A3 obtener información de cualquiera de las dos tablas y también propagar el privilegio SELECT a otras cuentas.

- A1 puede emitir la siguiente instrucción:

GRANT SELECT ON empleado, departamento TO A3 WITH GRANT OPTION;

La claúsula GRANT OPTION significa que A3 puede propagar el privilegio a otras cuentas empleando GRANT.

- Por ejemplo A3 puede conceder a A4 el privilegio SELECT para la relación empleado mediante la instrucción:

GRANT SELECT ON empleado TO A4;

- A4 no puede conceder el privilegio SELECT a otras cuentas, porque no se le concedió GRANT OPTION.

- Supongamos que ahora A1 decide revocar el privilegio SELECT que A3 tienen para la relación empleado; A1 podría emitir la instrucción:

    - REVOKE SELECT ON empleado FROM A3;

- EL SGBD deberá ahora revocar automáticamente el privilegio SELECT para empleado que tiene A4, porque A3 le concedió ese privilegio a A4 pero A3 ya no lo tiene.

- A continuación supongamos que A1 quiere devolver a A3 la capacidad SELECT limitada para la relación empleado y desea que A3 pueda propagar ese privilegio.

- La limitación es que sólo podrá obtener los atributos, nombre, fecha-ncto, y dirección y sólo la tupla con nd=5.

- A1 podrá crear esta vista:

- CREATE VIEW A3empleado AS
  SELECT nombre, fecha-ncto, dirección
  FROM empleado WHERE nd=5;

- Una vez creada la vista, A1 puede conceder a A3 el privilegio SELECT para la vista A3empleado:

- GRANT SELECT ON A3empleado TO A3 WITH GRANT OPTION;

- Por último supongamos que A1 desea dar permiso a A4 para actualizar sólo el atributo salario de Empleado; A1 puede emitir la siguiente instrucción:

- GRAN UPDATE ON empleado(salario) to A4;

- El privilegio UPDATE o INSERT puede especificar atributos concretos que puede ser actualizados o insertados en una relación. Otros pivilegios (select, delete) no pueden limitarse a ciertos atributos.

Integridad semántica y confidencialidad

Introducción

Integridad semántica

Control de accesos

Cifrado

- Se utiliza cuando las precauciones que tome el SGBD para evitar el acceso sin autorización no sean suficientes para proteger datos muy importantes.

- Propiedades de buenas técnicas de cifrado:

  - Para los usuarios autorizados es sencillo cifrar y descifrar datos.

  - El esquema de cifrado no depende de mantener en secreto el algoritmo de cifrado, sino de un parámetro del algoritmo llamado clave de cifrado.

  - Para un intruso es muy difícil determinar cual es la clave de cifrado.

- **Cifrado de clave pública:**
  - Cada usuario tiene dos claves: pública y privada.
  - Todas las claves públicas se publican de manera que las conozca todo el mundo.
  - La clave privada sólo la conoce el usuario al que pertenece.
  - Si un usuario U1 desea enviar información confidencial a U2, utiliza el algoritmo con la clave pública de U2 y sólo la podrá descifrar éste con su clave privada.

BASES DE DATOS DISTRIBUIDAS

Bases de datos distribuidas

Conceptos de bases de datos distribuidas

Ventajas de las bases de datos distribuidas

Funciones adicionales de las bases de datos distribuidas

Técnicas de fragmentación, replicación y asignación de datos para el diseño de bases de datos distribuidas.

Conceptos de bases de datos distribuidas

- Un sistema de computación distribuido consiste en un conjunto de elementos de procesamiento, no necesariamente homogéneos, que están interconectados por una red de computadores, y que cooperan en la ejecución de ciertas tareas asignadas.

- El objetivo general de los sistemas de computación distribuidos es dividir un problema grande e inmanejable en piezas más pequeñas que permitan resolver el problema de forma coordinada.

- De manera que:
  - Se aprovecha más potencia del ordenador para resolver tareas complejas.
  - Cada elemento de procesamiento autónomo se puede gestionar independientemente y puede desarrollar sus propias aplicaciones.

Introducción

Podemos definir **una base de datos distribuida** (BDD) como una colección de múltiples bases de datos interrelacionadas lógicamente, distribuidas por una red de computadores, y un **sistema de gestión de bases de datos distribuido** (SGBDD) como un sistema software que maneja una base de datos distribuida haciendo la distribución transparente para el usuario.

Bases de datos distribuidas

Conceptos de bases de datos distribuidas

Ventajas de las bases de datos distribuidas

Funciones adicionales de las bases de datos distribuidas

Técnicas de fragmentación, replicación y asignación de datos para el diseño de bases de datos distribuidas.

Ventajas de las bases de datos distribuidas

1.- **Gestión de datos distribuidos con diferentes niveles de transparencia:** un SGBDD debería ofrecer transparencia en la distribución en el sentido de que debería ocultar los detalles de dónde se almacena físicamente cada fichero (tabla, relación).

Son posibles los siguientes tipos de transparencias:

- **Transparencia de red o de distribución**: hace referencia a la liberación del usuario de los detalles operacionales de la red.
- **Transparencia de réplica**: las copias de los datos se deben almacenar en varios sitios para mejorar su disponibilidad, rendimiento y fiabilidad. La transparencia de réplica hace que el usuario desconozca la existencia de copias.

Ventajas de las bases de datos distribuidas

- **Transparencia de fragmentación**: son posibles dos tipos de fragmentación.
  - **Fragmentación horizontal:** distribuye una relación en conjuntos de tuplas (filas).
  - **Fragmentación vertical:** distribuye una relación en subrelaciones, donde cada subrelación se define por un subconjunto de las columnas de la relación original.

Ventajas de las bases de datos distribuidas

2**.- Incremento de la fiabilidad y disponibilidad**: estas son dos de las más comunes ventajas citadas para las bases de datos distribuidas.

- **Fiabilidad:** Probabilidad de que un sistema esté en marcha (no caído) en un punto de tiempo determinado.
- **Disponibilidad:** probabilidad de que un sistema esté disponible continuamente durante un intervalo de tiempo.

Cuando los datos y el software del SGBD están distribuidos por varios sitios, un sitio puede fallar mientras otros continúan operando.

Solamente son inaccesibles los datos y el software que hay en el sitio que ha fallado.

Esto mejora tanto la fiabilidad como la disponibilidad.

Ventajas de las bases de datos distribuidas

3.- **Mejora del rendimiento**: un SGBD distribuido fragmenta la base de datos manteniendo los datos cerca de donde más se necesitan.

- Cuando se distribuyen una gran base de datos por múltiples sitios, hay bases de datos más pequeñas en cada sitio. Como resultado, las consultas locales y las transacciones que acceden a datos de un único sitio tienen mejor rendimiento porque las bases de datos locales son más pequeñas.
- Además cada sitio tienen un menor número de transacciones ejecutándose que si todas las transacciones refenciasen a una base de datos centralizada.
- El paralelismo entre intraconsulta e interconsulta puede conseguirse ejecutando varias consultas en sitios diferentes, o descomponiendo una consulta en subconsultas que puedan ejecutarse en paralelo.

Ventajas de las bases de datos distribuidas

4.- **Expansión más sencilla**: en un entorno distribuido, la expansión del sistema mediante la **adicción de datos**, el **aumento del tamaño de la base de datos**, o la **adicción de más procesadores** es mucho más sencilla**.**

Bases de datos distribuidas

Conceptos de bases de datos distribuidas

Ventajas de las bases de datos distribuidas

Funciones adicionales de las bases de datos distribuidas

Técnicas de fragmentación, replicación y asignación de datos para el diseño de bases de datos distribuidas.

Funciones adicionales de las bases de datos distribuidas

La distribución conlleva un incremento de la complejidad en el diseño e implementación del sistema.

El software del SGBDD debe poder proporcionar las siguientes funciones además de las proporcionadas por los SGBD centralizados:

- **Mantenimiento de la pista de datos**: la capacidad para seguir la pista a la distribución de datos, la fragmentación y la réplica expandiendo el catálogo del SGBDD.
- **Procesamiento de consultas distribuidas**: la capacidad para acceder a sitios remotos y transmitir consultas y datos a través de varios sitios utilizando una red de comunicación.
- **Gestión de transacciones distribuidas**: la capaciad de idear estrategias de ejecución para consultas y transacciones que acceden a los datos desde más de un sitio y de sincronizar el acceso a datos distribuidos y mantener integridad sobre toda la base de datos.

Funciones adicionales de las bases de datos distribuidas

- **Gestión de datos replicados**: la capacidad de decidir a qué copia de un elemento de datos replicado acceder y de mantener la consistencia de copia de los elementos de datos replicados.
- **Recuperación de base de datos distribuidas**: la capacidad de recuperarse de caídas de sitios individuales y de nuevos tipos de fallos como el de enlaces de comunicación.
- **Seguridad**: las transacciones distribuidas se deben ejecutar con una gestión apropiada de seguridad de datos y los privilegios de autorización/acceso de los usuarios.
- **Gestión de directorio (catálogo) distribuido**: un directorio contiene información (metadatos) de los datos de la base de datos. El directorio debe ser global para toda la BDD, o local para cada sitio.

Funciones adicionales de las bases de datos distribuidas

- **En nivel físico de hardware**, un **sistema centralizado se distingue de un SGBDD** por:
  - Existen varios computadores llamados sitios o nodos.
  - Estos sitios se deben conectar a través de algún tipo de red de comunicación para transmitir datos o instrucciones a lo largo de los diferentes sitios.

- Los sitios podrían estar localizados físicamente próximos, es decir dentro del mismo edificio o grupo de edificios adyacentes, y conectados mediante una red de área local, ó podrían estar distribuidos geográficamente a gran distancia unos de otros y conectados mediante una red de área ancha o de larga distancia.

Bases de datos distribuidas

Conceptos de bases de datos distribuidas

Ventajas de las bases de datos distribuidas

Funciones adicionales de las bases de datos distribuidas

Técnicas de fragmentación, replicación y asignación de datos para el diseño de bases de datos distribuidas.

Fragmentación de datos: consiste en dividir la base de datos en unidades lógicas llamados fragmentos, cuyo almacenamiento pueda asignarse a varios sitios.

Las unidades lógicas más simples son las propias relaciones, es decir, cada relación completa se almacenará en un sitio específico.

Una relación fragmentada está dividida en varios fragmentos, cada uno de los cuales se almacena en una localidad:
r = {r1, r2, r3, ... rn}, a partir de los cuales es posible reconstruir la relación original.

Tipos de fragmentación: **horizontal** (cada tupla de r se asigna a uno o varios fragmentos), **vertical** (el esquema de r se divide en varios esquemas más pequeños) y **mixta**.

Ejemplo:

Fragmentación horizontal: r = {r1, r2}

ALUMNOS				
DNI	NOMBRE	APELLIDOS	DIRECCIÓN	LOCALIDAD
01111111	PEDRO	LOPEZ CUBERO	C/ OLIVILLAS 4	ZAMORA
02222222	LUIS	PÉREZ GONZÁLEZ	C/ MAYOR BJ	SALAMANCA
03333333	JUAN	MARTÍN MARTÍN	C/ OLMOS 6	ZAMORA
04444444	JORGE	GONZÁLEZ GIL	C/ DEL RÍO 3, 4ºA	ZAMORA

$r1 = \sigma_{\text{LOCALIDAD="ZAMORA"}}(\text{ALUMNOS})$

DNI	NOMBRE	APELLIDOS	DIRECCIÓN	LOCALIDAD
01111111	PEDRO	LOPEZ CUBERO	C/ OLIVILLAS 4	ZAMORA
03333333	JUAN	MARTÍN MARTÍN	C/ OLMOS 6	ZAMORA
04444444	JORGE	GONZÁLEZ GIL	C/ DEL RÍO 3, 4ºA	ZAMORA

$r2 = \sigma_{\text{LOCALIDAD="SALAMANCA"}}(\text{ALUMNOS})$

DNI	NOMBRE	APELLIDOS	DIRECCIÓN	LOCALIDAD
02222222	LUIS	PÉREZ GONZÁLEZ	C/ MAYOR BJ	SALAMANCA

Fragmentación vertical: r = {r1, r2}

$r_1 = \pi_{DNI,\ NOMBRE,\ APELLIDOS,\ IDTUPLA}\ (ALUMNOS)$

DNI	NOMBRE	APELLIDOS	IDTUPLA
01111111	PEDRO	LOPEZ CUBERO	1
02222222	LUIS	PÉREZ GONZÁLEZ	2
03333333	JUAN	MARTÍN MARTÍN	3
04444444	JORGE	GONZÁLEZ GIL	4

$r_2 = \pi_{DIRECCIÓN,\ LOCALIDAD,\ IDTUPLA}\ (ALUMNOS)$

DIRECCIÓN	LOCALIDAD	IDTUPLA
C/ OLIVILLAS 4	ZAMORA	1
C/ MAYOR BJ	SALAMANCA	2
C/ OLMOS 6	ZAMORA	3
C/ DEL RÍO 3, 4ºA	ZAMORA	4

Fragmentación horizontal: es un subconjunto de las tuplas de esa relación.

Las tuplas que pertenecen al fragmento horizontal se especifican mediante una condición sobre uno o más de los atributos de la relación.

La fragmentación horizontal divide una relación horizontalmente agrupando filas para crear subconjuntos de tuplas, donde cada subconjunto tiene un cierto significado lógico.

Ningún sitio tiene porqué necesitar todos los atributos de una relación, lo que indica la necesidad de otro tipo diferente de fragmentación.

Fragmentación vertical: La fragmentación vertical de una relación mantiene sólo ciertos atributos de la relación.

Fragmentación mixta: Podemos entremezclar los dos tipos de fragmentación para obtener la fragmentación mixta.

El esquema de fragmentación de una base de datos es una definición de un conjunto de fragmentos que incluye todos los atributos y tuplas de la base de datos y satisface la condición de que la base de datos completa se puede reconstruir a partir de los fragmentos mediante una secuencia de operaciones de unión externa o reunión externa.

El esquema de asignación describe la asignación de fragmentos entre los sitios del SBDD, por tanto, es una correspondencia que especifica el sitio o sitios donde se almacena cada fragmento. Si un fragmento se almacena en más de un sitio, se dice que está replicado.

La replicación resulta útil para mejorar la disponibilidad de los datos. El caso más extremo es la replicación de toda la base de datos en todos los sitios del sistema distribuido, creando así **una base de datos totalmente replicada**.

También mejora el rendimiento de la recuperación en consultas globales, porque el resultado de la consulta se puede obtener en cualquier sitio.

La desventaja de la replicación completa es que puede reducir drásticamente la rapidez de las operaciones de actualización, pues una sola actualización lógica se deberá ejecutar en todas y cada una de las copias de la base de datos a fin de mantener la consistencia.

Cada fragmento o copia de un fragmento se debe asignar a un sitio determinado en el sistema distribuido. Este proceso se denomina **distribución de los datos (o asignación de los datos).**