Bases de datos relacionales y SQL

José R.R. Viqueira

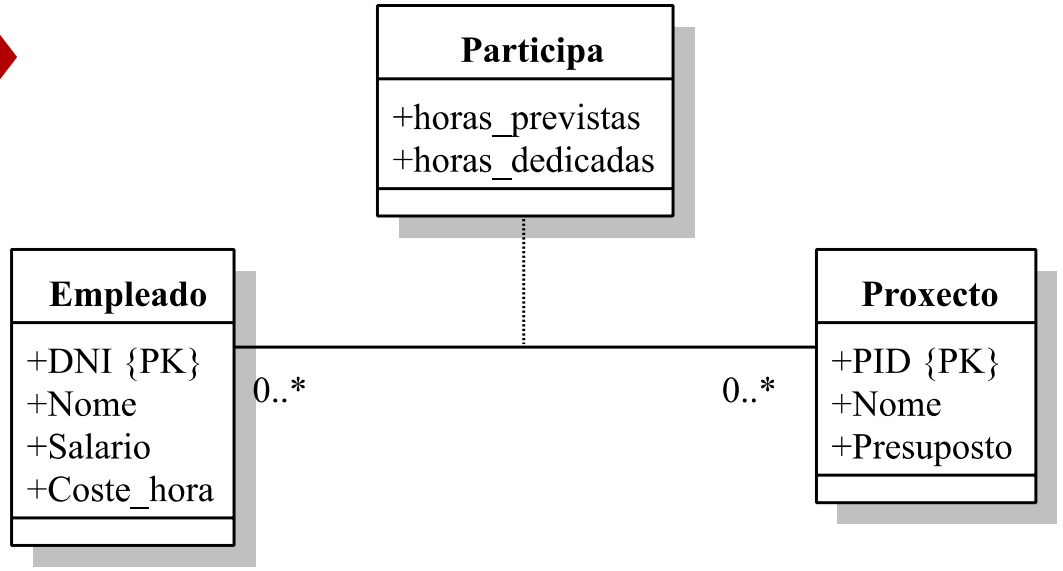
Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: http://citius.usc.es/equipo/persoal-adscrito/jrr.viqueira

Curso 2021/2022

Guion

- Modelo relacional y creación del esquema
- Consulta de datos
  - Lenguajes formales: Álgebra y Cálculo Relacional
  - Sintaxis básica
  - Valores nulos
  - Agregación
  - Subconsultas
  - Operadores de conjunto
  - Joins
- Transacciones
- Otras funcionalidades

Modelo relacional y creación del esquema



Proxecto

PID	Nome	Presuposto
1	Tráfico	198000
2	Xardinería	97000
3	Festas	86000

Empleado

DNI	Nome	Salario	Coste_hora
23456238	Alfredo	35000	45
25368964	Sofía	43000	60
58325647	Ricardo	29500	30
78878965	Elena	40500	55
78532564	Ernesto	41000	56

Participa

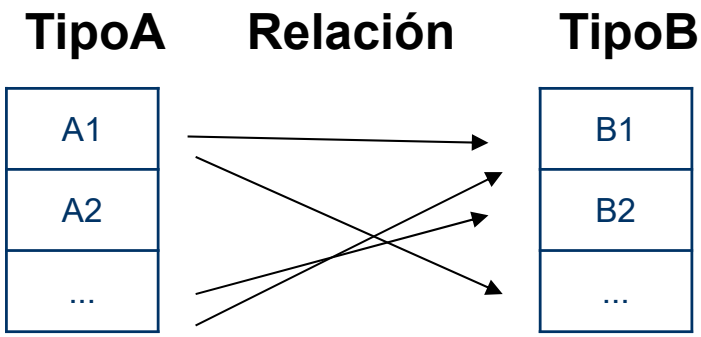
Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	1	1800	95
25368964	1	800	30
23456238	2	600	10
78878965	2	800	5
78532564	2	300	100

Modelo relacional y creación del esquema

Tipos de Dato

- char(n): tamaño fijo
- varchar(n): tamaño variable
- int, smallint
- numeric(p, d): punto fijo
- real, double precision: Punto flotante
- float (n): mínima precisión de n dígitos
- date, time, timestamp, interval
- Etc.

Modelo relacional y creación del esquema



Relación

A: TipoA	B: TipoB
A1	B1
A1	B12
. . .	. . .

Tipos no necesariamente distintos

NombreR

A1: T1	A2: T2	. . .	AN: TN
A1x	A2u	. . .	ANr
A1y	A2v	. . .	ANs
. . .	. . .	. . .	. . .

Esquema

Nombre de la relación + cunjunto de pares (nomre de atributo, tipo de datos)

NombreR (A1:T1, A2:T2, ..., AN:TN)

Contenido

Subconjunto del producto cartesiano de los N tipos de datos

Modelo relacional y creación del esquema

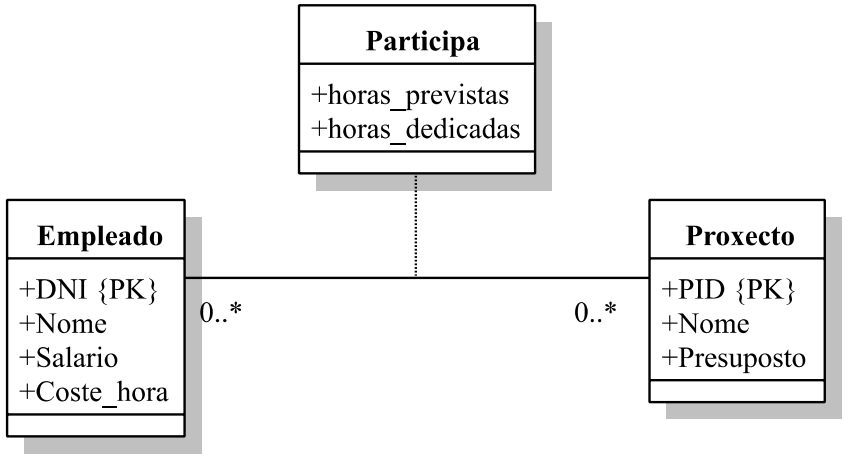
Lenguaje de definición de datos (DDL): Sintaxis básica

- CREATE TABLE Empleado (
  dni char(8),
  nome varchar(500) NOT NULL,
  salario decimal(8,2),
  coste_hora decimal(6,2) NOT NULL DEFAULT 40,
  primary key (dni));

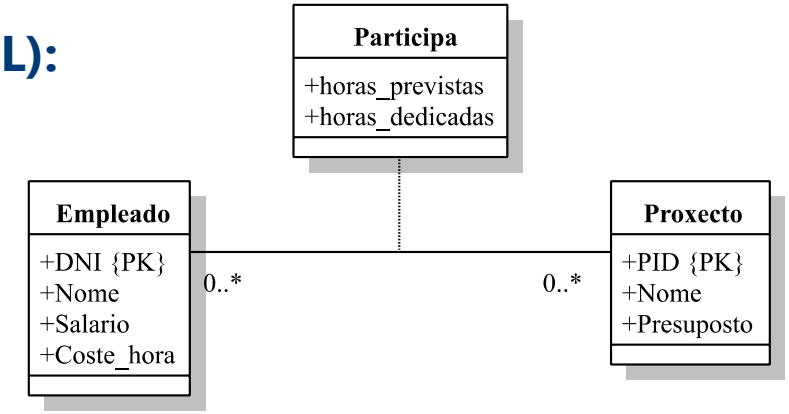
Esquema y Restricciones

- CREATE TABLE Proxecto(
  pid int primary key,
  nome varchar(500) NOT NULL,
  presuposto decimal(12, 2));

Se puede usar **UNIQUE (a1, a2, ...)** para definir otras claves candidatas



Lenguaje de definición de datos (DDL): Restricciones

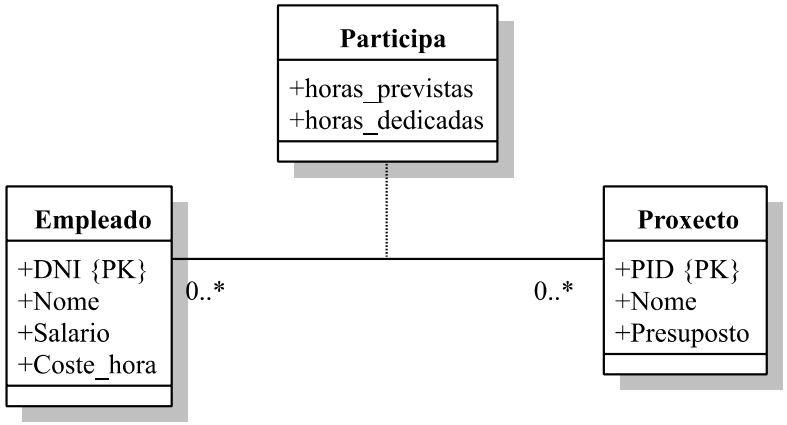



```
CREATE TABLE participa (
  empleado char(8),
  proxecto int,
  horas_previstas int NOT NULL,
  horas_dedicadas int DEFAULT 0,
  primary key (empleado, proxecto),
  foreign key (empleado) references Empleado
                         on delete restrict
                         on update cascade,
  foreign key (proxecto) references Proxecto
                         on delete cascade
                         on update cascade,
  check (horas_previstas >= horas_dedicadas));
```

- restrict
- cascade
- set null
- set default

Lenguaje de definición de datos (DDL): Eliminación y modificación del esquema

- DROP TABLE participa;
- ALTER TABLE Empleado
  ADD email varchar(500);
- ALTER TABLE Empleado
  DROP COLUMN email;
- ETC.



Lenguajes formales: Álgebra y Cálculo relacional

- **Álgebra relacional: Operaciones primitivas**
  - Selección: $\sigma_p(R)$
  - Proyección (proyección generalizada): $\pi_{A1,A2,...,An}(R)$
  - Producto cartesiano: $R1 \times R2$
  - Unión: $R1 \cup R2$
  - Diferencia: $R1 - R2$
- **Álgebra relacional: Operaciones derivadas**
  - Intersección: $R1 \cap R2$
  - Join
    - Theta Join: $R1 \bowtie_p R2 \equiv \sigma_p (R1 \times R2)$
    - Natural Join: $R1 \bowtie R2$

Lenguaje Procedimental

Lenguajes formales: Álgebra y Cálculo relacional

- **Cálculo de predicados (lógica de predicados)**
  - **Constantes, variables, predicados, funciones**
  - **Conectores:** $\vee, \wedge, \rightarrow, \leftrightarrow, \neg$
  - **Cuantificadores:** $\forall, \exists$
  - **Expresiones:**
    - $numero(x) \wedge \neg (\exists y(numero(y) \wedge mayor(x, y)))$

Lenguajes formales: Álgebra y Cálculo relacional

- **Cálculo relacional (de tuplas)**
  - Expresiones: {t | P(t)}
  - Variables referencian tuplas: t
  - Función que accede al valor de un atributo: t[A]
  - Constantes son valores de los tipos de datos
  - Predicados
    - Relaciones: Empleado(t)
    - Otros predicados implementados en el sistema: x > y, x = y, etc.
  - Conectores: $\vee$, $\wedge$, $\rightarrow$, $\neg$
  - Cuantificadores: $\exists x(\text{Empleado}(x) \wedge x[\text{salario}] > y[\text{salario}])$

Lenguaje Declarativo

Lenguajes formales: Álgebra y Cálculo relacional

Cálculo relacional (de tuplas)

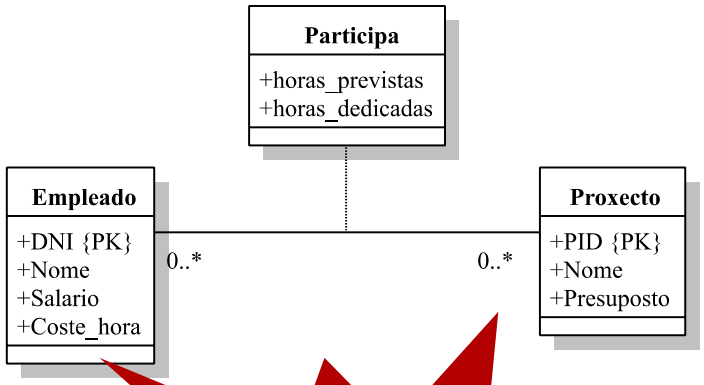
Expresiones Seguras

- Una expresión podría generar un resultado infinito
  - $\{ t \mid \neg \text{Empleado}(t) \}$
- El lenguaje debería de evitar que se puedan construir expresiones como esas
- Por ejemplo, forzar a que todas las variables tengan que estar referenciadas dentro de un predicado de relación sin negar.
  - $\{ t \mid \neg \text{Empleado}(t) \wedge \text{cliente}(t)\}$

Lenguajes formales: Álgebra y Cálculo relacional

Procesamiento de consultas

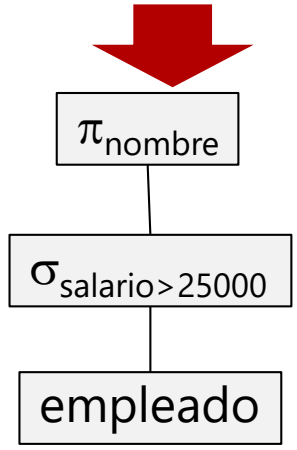
- Nombre de los empleados con un salario mayor de 25000



$\{t[nome] \mid empleado(t) \wedge t[salario] > 25000\}$

Lenguaje Declarativo

$\pi_{nombre}(\sigma_{salario>25000}(Empleado))$



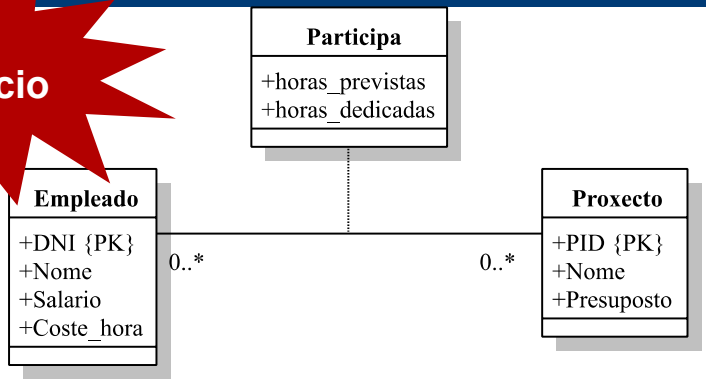
Lenguaje Procedimental

Lenguajes formales: Álgebra y Cálculo relacional

Procesamiento de consultas

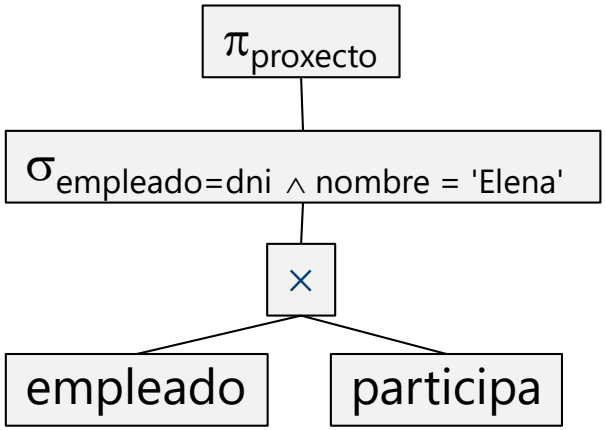
Ejercicio

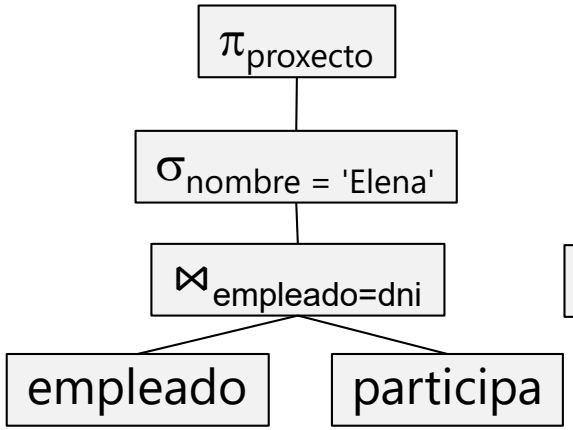
- Identificadores de proyecto de los proyectos en los que ha participado 'Elena'

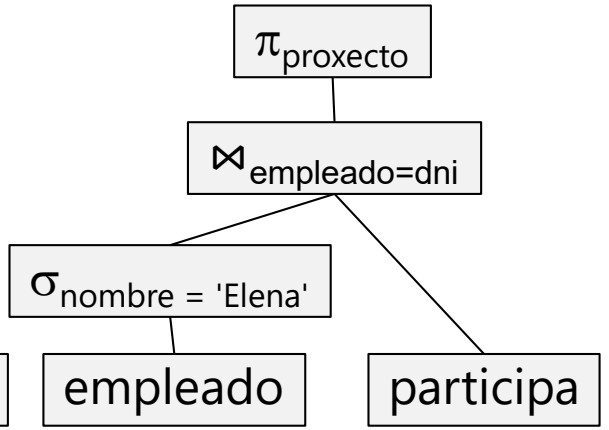


$$\{ p[proxecto] \mid participa(p) \wedge empleado(e) \wedge p[empleado]=e[dni] \wedge e[nombre]='Elena' \}$$

$$\{ p[proxecto] \mid participa(p) \wedge \exists e(empleado(e) \wedge p[empleado]=e[dni] \wedge e[nombre]='Elena') \}$$







Sintaxis básica

- **Sintaxis amigable (parecido a expresarlo en Inglés)**
- **Expresiones seguras del cálculo relacional de tuplas**
- **Generación de árboles de ejecución basados en álgebra relacional**
- **Uso de multi conjuntos en lugar de conjuntos**

③ SELECT $e_1, e_2, \ldots, e_n$ → $\pi_{e1, e2, \ldots, en}$

① FROM R_1 as r1, R_2 as r2, ..., R_m as rm → $R_1 \times R_2 \times \ldots \times R_m$

② WHERE p → σ_p

$$\{ e_1, e_2, \ldots, e_n \mid R_1(r1) \wedge R_2(r2) \wedge \ldots \wedge R_m(rm) \wedge p\ (r1, r2, \ldots, rm)\}$$

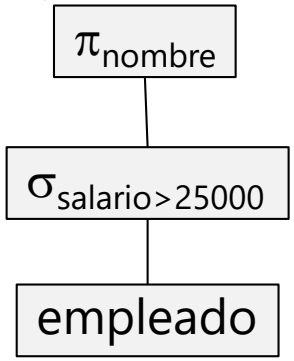
Sintaxis básica

- **Nombre de los empleados con un salario mayor de 25000**

Ejercicio

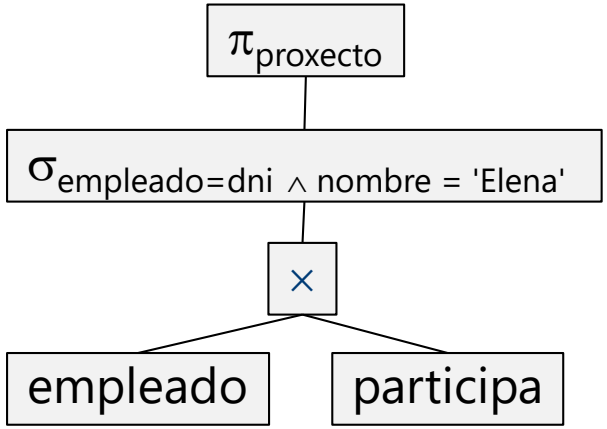
```
SELECT nome
FROM empleado as e
WHERE e.salario > 25000
```

```
SELECT nome
FROM empleado
WHERE salario > 25000
```



- **Identificadores de los proyectos en los que ha participado 'Elena'**

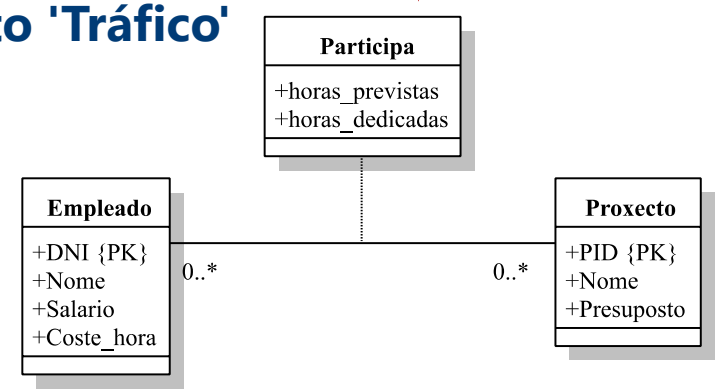
```
SELECT p.proxecto
FROM empleado as e, participa as p
WHERE e.dni=p.empleado and
      e.nombre='Elena'
```



Sintaxis básica

Ejercicio

- **Nombres y salarios de los empleados que ya han dedicado entre 20 y 40 horas al proyecto 'Tráfico'**
- **Ordena el resultado por salario de forma descendente y después por nombre de forma ascendente**




```
(3) SELECT e.nome, e.salario
(1) FROM empleado as e, participa pa, proxecto pr
(2) WHERE e.dni=pa.empleado
        and pr.pid=pa.proxecto
        and pa.horas_dedicadas  between 20 and 40
        and pr.nombre = 'Tráfico'
(4) ORDER BY e.salario DESC, e.nombre ASC
```

Ya no es una relación, es un listado

Sintaxis básica

- **Nombres de los empleados que han participado en algún proyecto con un presupuesto mayor de 90000**

Ejercicio

Proxecto

PID	Nome	Presuposto
1	Tráfico	198000
2	Xardinería	97000
3	Festas	86000

Empleado

DNI	Nome	Salario	Coste_hora
23456238	Alfredo	35000	45
25368964	Sofía	43000	60
58325647	Ricardo	29500	30
78878965	Elena	40500	55
78532564	Ernesto	41000	56

Participa

Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	1	1800	95
25368964	1	800	30
23456238	2	600	10
78878965	2	800	5
78532564	2	300	100

```
SELECT e.nome
FROM empleado as e, participa pa,
        proxecto pr
WHERE e.dni=pa.empleado
    and pr.pid=pa.proxecto
    and pr.presuposto > 90000
```

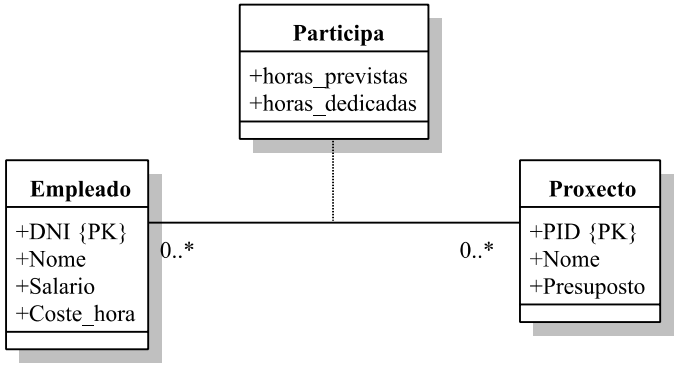
Nome
Alfredo
Sofía
Alfredo
Ricardo
Elena
Ernesto

```
SELECT DISTINCT e.nome
FROM empleado as e, participa pa,
        proxecto pr
WHERE e.dni=pa.empleado
    and pr.pid=pa.proxecto
    and pr.presuposto > 90000
```

Nome
Alfredo
Sofía
Ricardo
Elena
Ernesto

Valores Nulos

- **Cada tipo de dato tiene un valor especial NULL**
  - Valor no conocido, todavía no insertado, etc.
  - También el tipo de datos Boolean tiene el valor NULL. Las siguientes expresiones booleanas evalúan a NULL
    - **NULL = NULL**
    - **A = NULL**
  - Clausula WHERE obtiene solo las tuplas para las que el predicado se evalúa a TRUE



Todos los atributos

```
SELECT *
FROM empleado
WHERE salario = NULL
```

```
SELECT *
FROM empleado
WHERE salario IS NULL
```

```
SELECT *
FROM empleado
WHERE salario IS NOT NULL
```

Agregación

- **Obtener la media del salario de todos los empleados**

Función de Agregado

```
SELECT AVG(salario)
FROM empleado
```

Proxecto

PID	Nome	Presuposto
1	Tráfico	198000
2	Xardinería	97000
3	Festas	86000

Empleado

DNI	Nome	Salario	Coste_hora
23456238	Alfredo	35000	45
25368964	Sofía	43000	60
58325647	Ricardo	29500	30
78878965	Elena	40500	55
78532564	Ernesto	41000	56

Participa

Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	1	1800	95
25368964	1	800	30
23456238	2	600	10
78878965	2	800	5
78532564	2	300	100

↓

AVG = 37800

```
SELECT AVG(salario), MIN(salario), MAX(Salario), COUNT(*), COUNT(DISTINCT coste_hora)
FROM empleado
```

~~SELECT AVG(salario), nome~~
~~FROM empleado~~

Agregación

- **Para cada proyecto, obtener su nombre y el número de empleados que participan**

Proxecto

PID	Nome	Presuposto
1	Tráfico	198000
2	Xardinería	97000
3	Festas	86000

4. SELECT pr.nome, COUNT(*)
1. FROM proxecto pr, participa pa
2. WHERE pr.pid = pa.proxecto
3. GROUP BY pr.pid, pr.nome

Empleado

DNI	Nome	Salario	Coste_hora
23456238	Alfredo	35000	45
25368964	Sofía	43000	60
58325647	Ricardo	29500	30
78878965	Elena	40500	55
78532564	Ernesto	41000	56

Participa

Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	1	1800	95
25368964	1	800	30
23456238	2	600	10
78878965	2	800	5
78532564	2	300	100

Empleado	Proxecto	horas_previsas	horas_dedicadas	PID	Nome	Presuposto
23456238	1	1800	95	1	Tráfico	198000
25368964	1	800	30	1	Tráfico	198000
23456238	2	600	10	2	Xardinería	97000
78878965	2	800	5	2	Xardinería	97000
78532564	2	300	100	2	Xardinería	97000

Nome	count
Tráfico	2
Xardinería	3

Agregación

- **Para cada empleado, obtener la suma de horas que ha dedicado ya a sus proyectos**

Proxecto

PID	Nome	Presuposto
1	Tráfico	198000
2	Xardinería	97000
3	Festas	86000

Empleado

DNI	Nome	Salario	Coste_hora
23456238	Alfredo	35000	45
25368964	Sofía	43000	60
58325647	Ricardo	29500	30
78878965	Elena	40500	55
78532564	Ernesto	41000	56

Participa

Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	1	1800	95
25368964	1	800	30
23456238	2	600	10
78878965	2	800	5
78532564	2	300	100

```
SELECT e.dni, e.nombre, SUM(pa.horas_dedicadas)
FROM empleado e, participa p
WHERE e.dni = p.empleado
GROUP BY e.dni
```

Agregación

- **Para cada empleado que participe en dos o más proyectos, obtener el número de horas que tiene previsto dedicar en media a cada proyecto**

Proxecto

PID	Nome	Presuposto
1	Tráfico	198000
2	Xardinería	97000
3	Festas	86000

```
5  SELECT e.dni, e.nombre,
                 AVG(horas_previstas)
1  FROM empleado e, participa p
2  WHERE e.dni = p.empleado
3  GROUP BY e.dni
4  HAVING count(*) > 2
```

Empleado

DNI	Nome	Salario	Coste_hora
23456238	Alfredo	35000	45
25368964	Sofía	43000	60
58325647	Ricardo	29500	30
78878965	Elena	40500	55
78532564	Ernesto	41000	56

Participa

Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	1	1800	95
25368964	1	800	30
23456238	2	600	10
78878965	2	800	5
78532564	2	300	100

dni	Nome	Salario	coste_hora	Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	Alfredo	35000	45	23456238	1	1800	95
23456238	Alfredo	29500	30	23456238	2	600	10
~~25368964~~	~~Sofía~~	~~43000~~	~~60~~	~~25368964~~	~~1~~	~~800~~	~~30~~
~~78878965~~	~~Elena~~	~~40500~~	~~55~~	~~78878965~~	~~2~~	~~800~~	~~5~~
~~78532564~~	~~Ernesto~~	~~41000~~	~~56~~	~~78532564~~	~~2~~	~~300~~	~~100~~

dni	Nome	horas_previsas
23456238	Alfredo	1200

Agregación

- **Para cada proyecto que tenga más de dos empleados, obtener su identificador, nombre, presupuesto total, presupuesto previsto para personal y presupuesto previsto para otros gastos**

Proxecto

PID	Nome	Presuposto
1	Tráfico	198000
2	Xardinería	97000
3	Festas	86000

Empleado

DNI	Nome	Salario	Coste_hora
23456238	Alfredo	35000	45
25368964	Sofía	43000	60
58325647	Ricardo	29500	30
78878965	Elena	40500	55
78532564	Ernesto	41000	56

Participa

Empleado	Proxecto	horas_previsas	horas_dedicadas
23456238	1	1800	95
25368964	1	800	30
23456238	2	600	10
78878965	2	800	5
78532564	2	300	100


```
SELECT pr.pid, pr.nome, pr.presupuesto,
            SUM(pa.horas_previstas*e.coste_hora) AS personal,
            pr.presupuesto - SUM(pa.horas_previstas*e.coste_hora) AS otros_gastos
FROM proxecto pr, participa pa, empleado e
WHERE pr.pid=pa.proxecto and e.dni=pa.empleado
GROUP BY pr.pid
HAVING count(*) > 2
```

Subconsultas

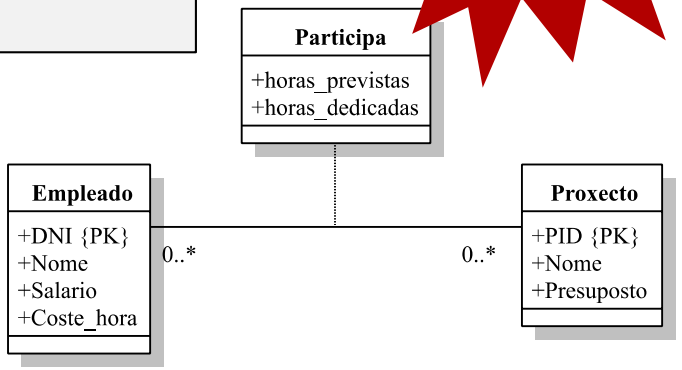
Subconsultas en la clausula FROM

Devuelve una tabla

```
SELECT e1, e2, ..., en
FROM R1 as r1, (SELECT ...) as r2, ..., Rm as rm
WHERE p
```

- Permiten implementar estrategias de tipo divide y vencerás
- Para cada proyecto, obtén el número de empleados con un coste por hora mayor de 40 y el número de empleados con un coste por hora menor o igual que 40

Ejercicio




```
SELECT p.pid, p.nome, Tmas40.mas40, Tmenos40.menos40
FROM proyectos p,
        (SELECT p.proxecto, count(*) as mas40
         FROM participa p, empleado e
         WHERE p.empleado=e.dni and coste_hora > 40
         GROUP BY p.proxecto) as Tmas40,
        (SELECT p.proxecto, count(*) as menos40
         FROM participa p, empleado e
         WHERE p.empleado=e.dni and coste_hora <= 40
         GROUP BY p.proxecto) as Tmenos40
WHERE p.pid = Tmas40.proxecto and p.pid = Tmenos40.proxecto
```

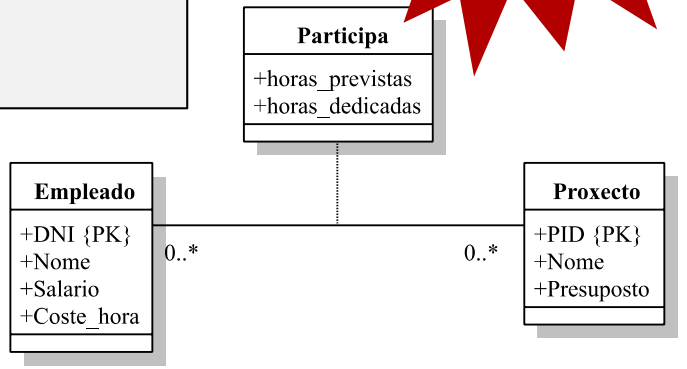
Subconsultas

Subconsultas en otras subclausulas

Devuelve un único valor

```
SELECT e1, (SELECT ...) as e2, ..., en
FROM R1 as r1, R2 as r2, ..., Rm as rm
WHERE ... (SELECT ...) ...
GROUP BY ...
HAVING ... (SELECT ...) ...
```

Ejercicio



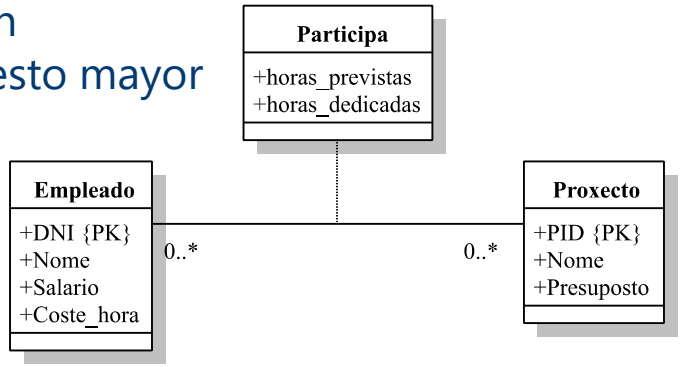
- Obtén todos los datos del empleado con el máximo salario

```
SELECT *
FROM empleado
WHERE salario = (SELECT MAX(salario) FROM empleado)
```

Subconsultas

- **Predicados de conjuntos y cuantificadores**
  - Se usan en la clausulas WHERE y HAVING
    - IN
    - Modificadores de comparaciones (ANY, SOME, ALL)
    - EXISTS
  - Obtener los datos de los empleados que han participado en algún proyecto con presupuesto mayor de 90000

Ejercicio




```
SELECT DISTINCT e.*
FROM empleado e, participa pa, proxecto pr
WHERE e.dni = pa.empleado
    and pr.pid = pa.proxecto
    and pr.presuposto > 90000
```

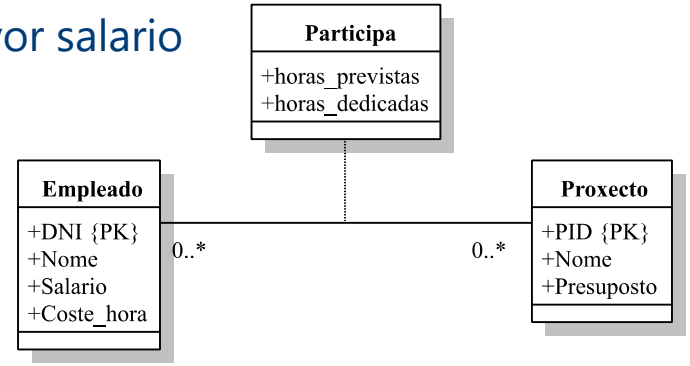
```
SELECT *
FROM empleado
WHERE dni IN
        (SELECT empleado
         FROM participa pa, proxecto pr
         WHERE pa.proxecto=pr.pid
              and pr.presuposto > 90000)
```

```
SELECT *
FROM empleado
WHERE dni = ANY
        (SELECT empleado
         FROM participa pa, proxecto pr
         WHERE pa.proxecto=pr.pid
              and pr.presuposto > 90000)
```

Subconsultas

- **Predicados de conjuntos y cuantificadores**
  - Se usan en la clausulas WHERE y HAVING
    - IN
    - Modificadores de comparaciones (ANY, SOME, ALL)
    - EXISTS
  - Obtener los datos del empleado con el mayor salario

Ejercicio




```
SELECT *
FROM empleado
WHERE salario = (SELECT MAX(salario)
                 FROM empleado)
```

```
SELECT *
FROM empleado
WHERE salario > = ALL (SELECT salario
                       FROM empleado)
```

```
SELECT *
FROM empleado e1
WHERE NOT EXISTS (SELECT *  FROM empleado e2
                  WHERE e1.salario < e2.salario)
```

$$\{e1 \mid empleado(e1) \wedge \neg\exists e2(empleado(e2) \wedge e1[salario] < e2[salario])\}$$

Operadores de Conjunto

Union, intersección y diferencia de conjuntos

```
SELECT ...
FROM ...
...
UNION
SELECT ...
FROM ...
...
ORDER BY ...
```

```
SELECT ...
FROM ...
...
UNION ALL
SELECT ...
FROM ...
...
ORDER BY ...
```

```
SELECT ...
FROM ...
...
EXCEPT
SELECT ...
FROM ...
...
ORDER BY ...
```

```
SELECT ...
FROM ...
...
INTERSECT
SELECT ...
FROM ...
...
ORDER BY ...
```

Recordar que las tablas deben ser compatibles

Por defecto se eliminan duplicados

Joins

- **INNER Theta Join (cualquier predicado)**
  - Mostrar los datos de los empleados junto con los datos de su departamento

```
SELECT *
FROM empleado e, departamento d
WHERE e.depId = d.depId
```

Empleado

DNI	Nome	Salario	Coste_hora	depId
23456238	Alfredo	35000	45	1
25368964	Sofía	43000	60	1
58325647	Ricardo	29500	30	2
78878965	Elena	40500	55	2
78532564	Ernesto	41000	56	

Departamento

depId	nom_dept
1	Marketing
2	Ventas
3	Compras

DNI	Nome	Salario	Coste_hora	depId	depId	nom_dept
23456238	Alfredo	35000	45	1	1	Marketing
25368964	Sofía	43000	60	1	1	Marketing
58325647	Ricardo	29500	30	2	2	Ventas
78878965	Elena	40500	55	2	2	Ventas

Joins

- **INNER Theta Join (cualquier predicado)**
  - Mostrar los datos de los empleados junto con los datos de su departamento

Tipo de Join por defecto

Se puede eliminar

```
SELECT *
FROM empleado e INNER JOIN departamento d
        ON (e.depId = d.depId)
```

Empleado

DNI	Nome	Salario	Coste_hora	depId
23456238	Alfredo	35000	45	1
25368964	Sofía	43000	60	1
58325647	Ricardo	29500	30	2
78878965	Elena	40500	55	2
78532564	Ernesto	41000	56	

Departamento

depId	nom_dept
1	Marketing
2	Ventas
3	Compras

DNI	Nome	Salario	Coste_hora	depId	depId	nom_dept
23456238	Alfredo	35000	45	1	1	Marketing
25368964	Sofía	43000	60	1	1	Marketing
58325647	Ricardo	29500	30	2	2	Ventas
78878965	Elena	40500	55	2	2	Ventas

Joins

- **INNER Equi Join (predicado =)**
  - Mostrar los datos de los empleados junto con los datos de su departamento

Mismo nombre de columna

Se elimina una copia del resultado

```
SELECT *
FROM empleado e JOIN departamento d
     USING (depId)
```

Empleado

DNI	Nome	Salario	Coste_hora	depId
23456238	Alfredo	35000	45	1
25368964	Sofía	43000	60	1
58325647	Ricardo	29500	30	2
78878965	Elena	40500	55	2
78532564	Ernesto	41000	56	

Departamento

depId	nom_dept
1	Marketing
2	Ventas
3	Compras

DNI	Nome	Salario	Coste_hora	depId	nom_dept
23456238	Alfredo	35000	45	1	Marketing
25368964	Sofía	43000	60	1	Marketing
58325647	Ricardo	29500	30	2	Ventas
78878965	Elena	40500	55	2	Ventas

Joins

- **INNER NATURAL JOIN**
  - Mostrar los datos de los empleados junto con los datos de su departamento

Equi Join por todos los atributos con el mismo nombre

```
SELECT *
FROM empleado e NATURAL JOIN departamento d
```

Empleado

DNI	Nome	Salario	Coste_hora	depId
23456238	Alfredo	35000	45	1
25368964	Sofía	43000	60	1
58325647	Ricardo	29500	30	2
78878965	Elena	40500	55	2
78532564	Ernesto	41000	56	

Departamento

depId	nom_dept
1	Marketing
2	Ventas
3	Compras

DNI	Nome	Salario	Coste_hora	depId	nom_dept
23456238	Alfredo	35000	45	1	Marketing
25368964	Sofía	43000	60	1	Marketing
58325647	Ricardo	29500	30	2	Ventas
78878965	Elena	40500	55	2	Ventas

Joins

- **OUTER JOINS**
  - Se pierden los datos de Ernesto y del departamento de Compras

```
SELECT *
FROM empleado e
        NATURAL LEFT OUTER JOIN
        departamento d
```

```
SELECT *
FROM empleado e NATURAL RIGHT JOIN departamento d
```

```
SELECT *
FROM empleado e NATURAL FULL JOIN departamento d
```

Empleado

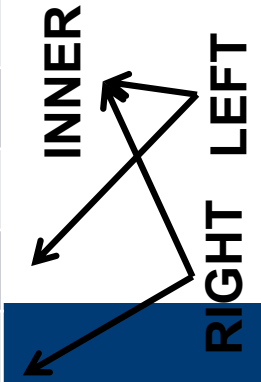
DNI	Nome	Salario	Coste_hora	depId
23456238	Alfredo	35000	45	1
25368964	Sofía	43000	60	1
58325647	Ricardo	29500	30	2
78878965	Elena	40500	55	2
78532564	Ernesto	41000	56	

Departamento

depId	nom_dept
1	Marketing
2	Ventas
3	Compras

FULL

DNI	Nome	Salario	Coste_hora	depId	nom_dept
23456238	Alfredo	35000	45	1	Marketing
25368964	Sofía	43000	60	1	Marketing
58325647	Ricardo	29500	30	2	Ventas
78878965	Elena	40500	55	2	Ventas
78532564	Ernesto	41000	56		
				3	Compras



Lenguaje de Manipulación de datos: Inserción de datos

```
INSERT INTO nombre_tabla (A1, A2, ..., An)
VALUES (v11, v12, ..., v1n),
       (v21, v22, ..., v2n),
       . . .
       (vm1, vm2, ..., vmn);
```

```
INSERT INTO nombre_tabla (A1, A2, ..., An)
SELECT e1, e2, ..., en
FROM ...
```

Si no se especifican se consideran todos los atributos de la tabla

Los no especificados se rellenan con valor por defecto o con nulos

Si no tienen valor por defecto y no admiten nulos se genera una excepción

Transacciones

Borrado de datos

```
DELETE FROM nombre_tabla
WHERE p;
```

Modificación de datos

```
UPDATE nombre_tabla
 SET a1 = e1, a2 = e2, ..., an = en
WHERE p;
```

Transacciones

TRANSACCIÓN

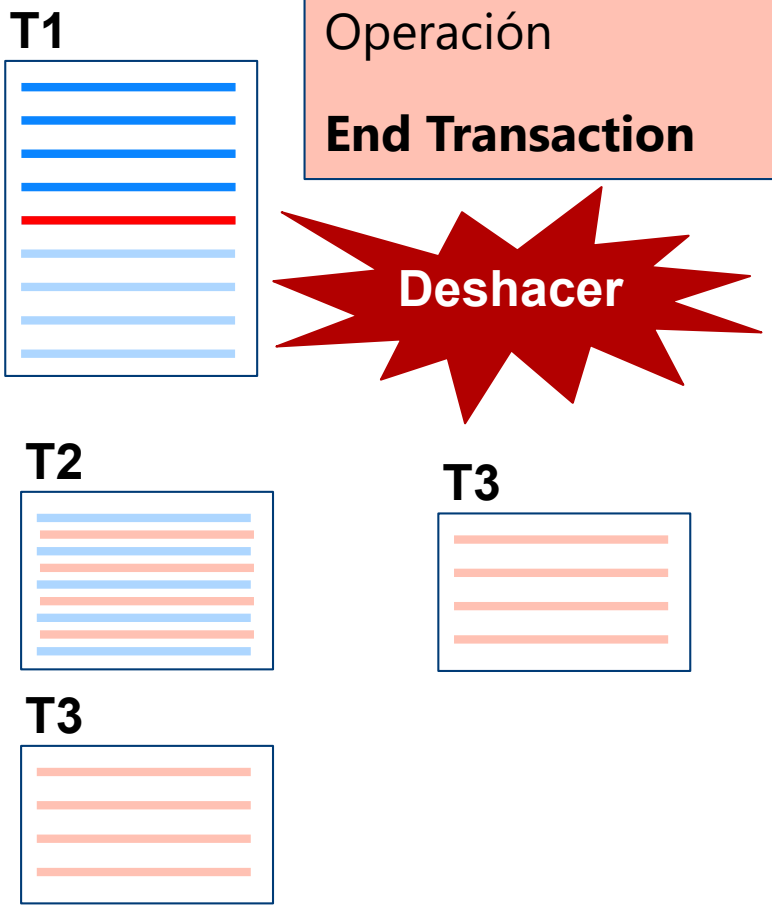
Unidad de ejecución de un programa que accede y posiblemente modifica varios elementos de datos

Begin Transaction

Operación
Operación
Operación
....
Operación

End Transaction

- Unidad simple e indivisible para el usuario (**Atomicidad**)
  - Complejo: Datos en memoria y en disco
- Operan sin interferencia de las operaciones de otras transacciones (**Aislamiento**)
- Sus efectos perduran a pesar de caídas del sistema (**Durabilidad**)
- Su ejecución aislada y atómica deja la base de datos en un estado consistente (**Consistencia**)
  - Dependiente de la aplicación



Transacciones

- Propiedades **ACID**
  - **Atomicidad** (**Atomicity**)
    - Se ejecutan todas las operaciones o no se ejecuta ninguna
  - **Consistencia** (**Consistency**)
    - La ejecución aislada de una transacción preserva la consistencia de la base de datos
  - **Aislamiento** (**Isolation**)
    - Aunque varias transacciones se ejecuten de forma concurrente, el sistema garantiza que cada par de transacciones T_i y T_j, para T_i da la impresión de que o bien T_j terminó su ejecución antes de iniciarla T_i o T_j inició su ejecución después de terminarla T_i.
  - **Durabilidad** (**Durability**)
    - Después de que una transacción termine con éxito, los cambio que ha hecho en la base de datos persisten, incluso si hay fallos en el sistema.

Asegurar el aislamiento puede tener un impacto grande en el rendimiento del sistema

→ Necesario?

Depende la aplicación

Transacciones

- **Consistencia**
  - Suma de A y B debe permanecer inalterada en la transacción
    - T1 no debe crear ni destruir dinero
  - Responsabilidad del programador
  - Comprobación automática de restricciones de integridad puede ayudar a cumplir esta propiedad
    - PRIMARY KEY, FOREIGN KEY, CHECK, etc.
- **Atomicidad**
  - Estado inconsistente no provocado por la mala praxis del programador
  - Estado inconsistente por el que pasa la transacción es temporal
    - No se dará si se ejecuta de forma atómica
  - Se mantiene un registro (log) de valores antiguos
    - Restaurar valores antiguos en caso de fallo
  - Responsabilidad de SGBDs
    - Sistema de recuperación

T1

```
leer(A);
A:= A – 50;
escribir(A);
leer(B);
B:=B+50;
escribir(B);
```

```
leer(A);
A:= A – 50;
escribir(A);
Fallo
```

Registro Histórico

Durabilidad

- Una vez ha terminada la transacción, los cambios perduran en la base de datos, incluso si hay un fallo en el sistema después de completarse la transacción.
- Para garantizar la durabilidad
  - La información sobre las modificaciones de la transacción se guarda en el disco
  - Dicha información permite al SGBD reconstruir las modificaciones cuando el sistema se reinicia después de un fallo.
- Responsabilidad del SGBDs
  - Sistema de recuperación
    - También responsable de la atomicidad.

T1

```
leer(A);
A:= A – 50;
escribir(A);
leer(B);
B:=B+50;
escribir(B);
```

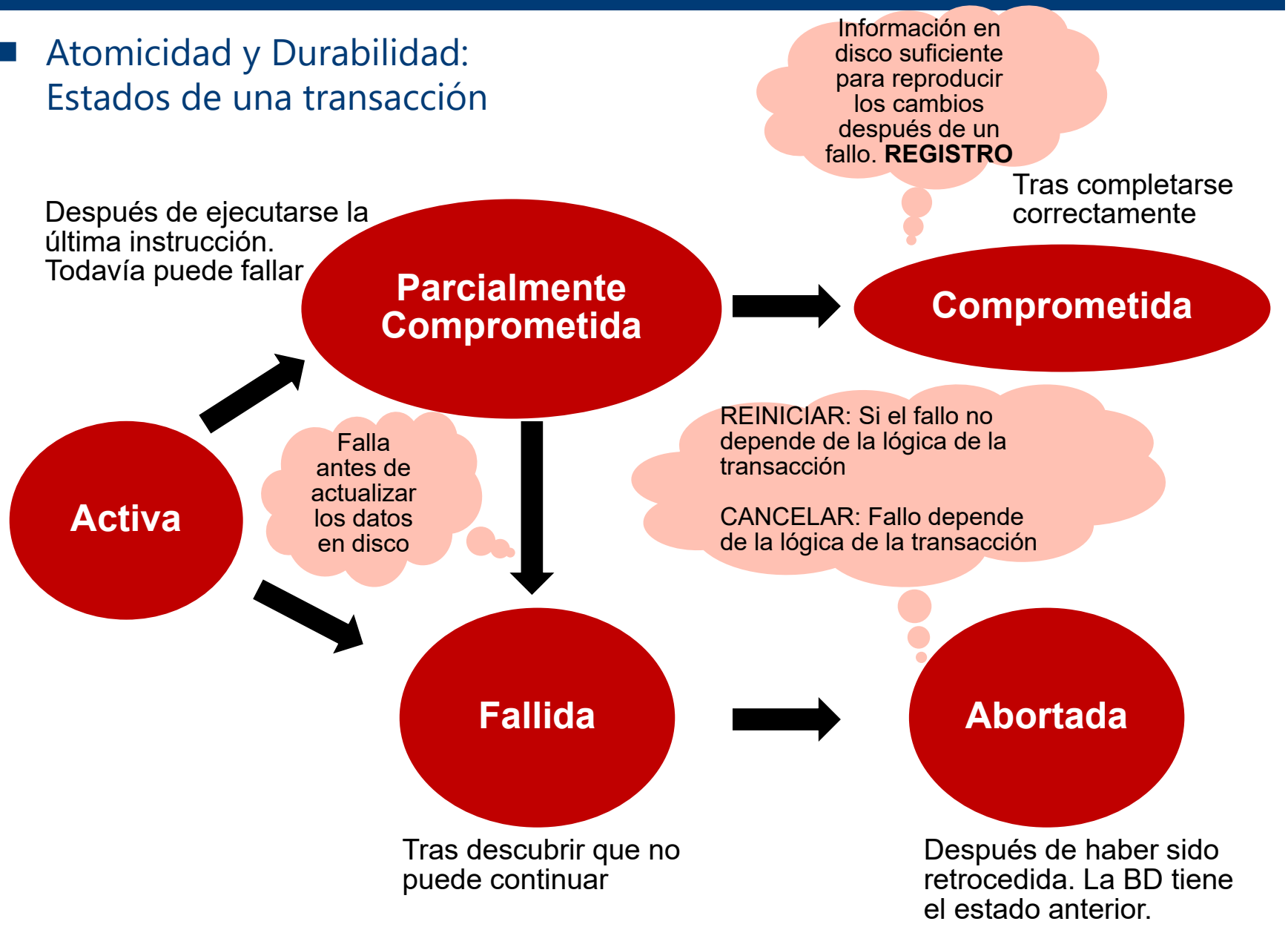
Transacciones

- **Aislamiento**
  - Incluso si la ejecución aislada de una transacción es Consistente, la ejecución concurrente con otra puede llevar a la base de datos a un estado inconsistente
    - El estado inconsistente puede mantenerse después del final de ambas transacciones
  - Solución:
    - Ejecución secuencial de las transacciones
    - Solución poco eficiente en cuanto a rendimiento del sistema
  - Esta propiedad garantiza que la ejecución concurrente de varias transacciones produce un resultado equivalente a la ejecución de las transacciones una detrás de la otra (secuencial) en algún orden
  - Responsabilidad del SGBDs
    - Sistema de control de concurrencia

T1	T2
leer(A);	
A:= A – 50;	
escribir(A);	
	leer(A);
	Leer (B);
	S:=A+B
	escribir(S);
leer(B);	
B:=B+50;	
escribir(B);	

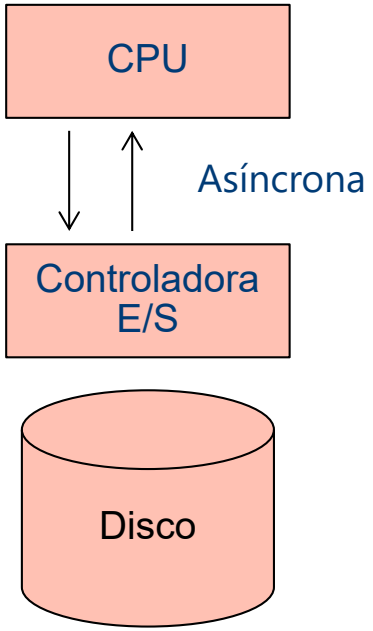
Transacciones

Atomicidad y Durabilidad: Estados de una transacción



Transacciones

- Aislamiento
  - Ejecución concurrente de varias transacciones puede generar inconsistencias
  - Razones para no limitarse a la ejecución secuencial
    - Mejora del rendimiento y uso de los recursos
      - Ejecución paralela de instrucciones E/S y CPU
        - Disminuye tiempo total de ejecución, aumenta uso CPU y E/S
    - Reducción tiempo medio de respuesta
      - Transacciones cortas no tienen que esperar a que termine una larga
  - Necesarios planificadores que garanticen aislamiento



- Aislamiento y Atomicidad
  - ¿Qué ocurre si falla una transacción durante la ejecución concurrente de varias?
  - Si T falla, es necesario deshacer sus efectos (atomicidad)
    - Cada Ti, que ha leído datos escritos por T debe deshacerse también!
      - Abortar transacciones que no han fallado
      - Si se han comprometido ya, nos e pueden abortar!

Transacciones

- Niveles de aislamiento en el estándar SQL
  - **Secuenciable**
    - Normalmente asegura la ejecución secuenciable de la transacción
      - Algunos SGBDs utilizan protocolos que no lo garantizan en todos los casos
  - **Lectura repetible**
    - Sólo se pueden leer datos comprometidos (escritos por transacciones comprometidas)
    - Entre dos lecturas del mismo dato en una transacción, otra no puede modificarlo
    - No garantiza la secuencialidad
  - **Lectura comprometida**
    - Sólo permite leer datos comprometidos
    - **NO** garantiza lecturas repetibles
  - **Lectura no comprometida**
    - Permite leer datos no comprometidos
    - Nunca escrituras sucias
      - Escribir sobre un dato escrito por otra transacción no comprometida

T I	T J
SELECT count(*) FROM empleado WHERE dept = 'Ventas'	INSERT INTO empleado (1, 'Juán', 23k, 'Ventas')

Las instrucciones de TI y TJ no deberían de tener conflicto ya que no actúan sobre los mismos elementos de datos

Sin embargo el orden de ejecución importa

HAI CONFLICTO

Fenómeno Fantasma

Transacciones

- Protocolos de implementación del aislamiento
  - ▷ Estrategias Pesimistas
    - Toman medidas que fuerzan esperas o retrocesos de transacciones como prevención
    - Basados en bloqueos
      - Protocolo de bloqueo en dos fases
    - Basados en marcas de tiempo
  - ▷ Estrategias optimistas
    - Permiten la ejecución normal y solo al final comprueban si hay problemas
    - Basados en validación
    - Basados en versiones múltiples
      - Aislamiento de instantáneas

No garantiza la secuencialidad

Pueden ser necesarias medidas adicionales

Cláusula **FOR UPDATE** (trata las lecturas como escrituras)

Otras Funcionalidades

- **Vistas**

```
CREATE VIEW nombre_vista AS
SELECT ...
FROM ...;
```

- **Creación de índices**
- **Tipos, funciones, procedimientos definidos por el usuario**
- **Disparadores**
- **Seguridad**
- **Acceso a la Base de datos desde un lenguaje de programación**
- **Consultas Recursivas**
- **Agregación avanzada y OLAP (... Inteligencia de Negocio)**

Bases de datos relacionales y SQL

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: http://citius.usc.es/equipo/persoal-adscrito/jrr.viqueira

Curso 2021/2022