Arquitectura de un sistema de bases de datos

Clase 01

IIC 3413

Prof. Cristian Riveros

Outline

Modelo Relacional

SQL

Algebra Relacional

Base de datos de grafos

Arquitectura

Outline

Modelo relacional

Definición

Una **relación** R esta compuesto por:

1. Esquema:

$$\texttt{name}(\texttt{att}_1 : D_1, \ldots, \texttt{att}_n : D_n)$$

   - $\texttt{name}$:= nombre de la relación R.
   - $\texttt{att}_i$:= nombre del atributo i.
   - D_i := dominio del atributo i.
   - n := aridad o cantidad de atributos de R.

2. Instancia: subconjunto finito de $D_1 \times \ldots \times D_n$.

Dada una relación R, usualmente denotaremos

- R: como el nombre y instancia de R (si no hay confusión).
- schema(R): como el esquema de R.
- arity(R): como la aridad de R.
- dom($\texttt{att}_i$): como el dominio D_i del atributo $\texttt{att}_i$.

Propiedades de una relación

El dominio de un atributo puede ser cualquier tipo primitivo:

- Números ($\mathbb{N}$, $\mathbb{Q}$, etc),
- Strings,
- Fechas, etc.

Una relación R se ve como una tabla con filas y columnas:

$\mathbf{att}_1$	$\mathbf{att}_2$	$\cdots$	$\mathbf{att}_n$
v_1	v_2	$\cdots$	v_n
v_1'	v_2'	$\cdots$	v_n'
$\vdots$	$\vdots$		$\vdots$

donde $(v_1, \ldots, v_n)$ o $(v_1', \ldots, v_n')$ son tuplas de R, PERO...

- El orden de las filas no es importante.
- Todas las filas son distintas.

Base de Datos (BD) relacional

Definición

1. Una **BD relacional** $\mathcal{D}$ es un conjunto finito de relaciones $R_1, \ldots, R_m$ cada una con un nombre distinto.

$$\mathcal{D} \quad = \quad \{R_1, R_2, \ldots, R_m\}$$

2. Un **esquema relacional** $\mathcal{S}$ de una BD $\mathcal{D}$ es el conjunto de esquemas de las relaciones $R_1, \ldots, R_m$.

$$\text{schema}(\mathcal{D}) \quad = \quad \{\text{schema}(R_1), \text{schema}(R_2), \ldots, \text{schema}(R_m)\}$$

Otras definiciones importantes:

- $\text{dom}(\mathcal{S})$ = todas las BD que tienen como esquema a $\mathcal{S}$.
- $|R|$ = numero de tuplas en R (o cardinalidad).

Base de Datos (BD) relacional

Ejemplo

- Esquema:

$$\text{Players}(\textit{pName} : \texttt{CHAR},\ \textit{pTeam} : \texttt{CHAR},\ \textit{pBirth} : \texttt{DATE},\ \textit{pPosition} : \texttt{INT})$$

$$\text{Teams}(\textit{tName} : \texttt{CHAR}, \textit{tFans} : \texttt{INT},\ \textit{tBirth} : \texttt{DATE})$$

- Instancia:

pName	**pTeam**	**pBirth**	**pPosition**
Alexis Sanchez	Inter de Milan	Dec 19, 1988	7
Gary Medel	Bologna	Ago 3, 1987	5

tName	**tFans**	**tBirth**
Bologna	38M	1909
Inter de Milan	80M	1908

Restricciones de Integridad

Definición

Restricciones de integridad = condición φ que restringen los datos que pueden ser almacenados en una BD relacional.

Una BD $\mathcal{D}$ es **valida** con respecto a φ si satisface las restricción φ.

$$\mathcal{D} \vDash \varphi$$

Ejemplos

- Restricciones de dominio.
- Llaves (Keys).
  - Primary Key.
  - Foreign Key.
- Condiciones generales.

Solo se permiten BD validas según las restricciones de integridad.

Extracción de datos basado en consultas

Consulta: función $f : \text{dom}(\mathcal{S}) \to D$

- $\mathcal{S}$ es una esquema relacional,
- D es un dominio cualquiera ($\mathbb{N}$, relaciones, etc).

Lenguaje de consultas: conjunto de expresiones **sintácticas** que definen consultas por medio de una **semántica**.

Dos lenguajes de consultas importantes para nosotros:

1. SQL
2. Algebra relacional

Outline

SQL (Structured Query Language)

- Standard mundial para consultas BD relacional.
- Lenguaje de consultas declarativo.
- Basado en calculo relacional (lógica de primer orden).
- Multiples componentes del lenguaje:
  - Data definition language (DDL).
    Ejemplo: `CREATE`, `ALTER`, etc...
  - Data manipulation language (DML).
    Ejemplo: `SELECT`, `INSERT`, `DELETE`, etc...
  - Data control language (DCL).
    Ejemplo: `GRANT`, `REVOKE`, etc...

Consultas SQL

Forma basica:

`SELECT`	< atributos >
`FROM`	< relaciones >
`WHERE`	< condiciones >

< atributos >: lista de atributos

- atributo
- Relacion . atributo
- Relacion . atributo `AS` nuevo_nombre

< relaciones >: lista de relaciones

< condiciones >: lista de condiciones booleanas

- φ_1 `AND` φ_2, φ_1 `OR` φ_2, `NOT` φ
- atributo$_1$ $\sim$ atributo$_2$ donde $\sim \in \{=, \leq, \geq, \ldots\}$.
- atributo$_1$ $\sim$ *constante* donde $\sim \in \{=, \leq, \geq, \ldots\}$, etc...

Ejemplos de consultas SQL

Ejemplos

Players (P):

pId	pName	pYear	pGoals
1	x	1987	100
2	y	1990	59
3	y	1985	88
NULL	z	1983	110

Matches (M):

mId	mStadium	mGoals
1	Nacional	3
2	Nacional	2
2	San Carlos	3
5	Monum.	4

- `SELECT` pName, pYear
  `FROM` Players
  `WHERE` pGoals ≥ 100
- `SELECT` P.pName `AS` Jugador, P.pYear `AS` Año
  `FROM` Players `AS` P
  `WHERE` pYear = 1990

Ejemplos de consultas SQL

Ejemplos

Players (P):

pId	pName	pYear	pGoals
1	x	1987	100
2	y	1990	59
3	y	1985	88
NULL	z	1983	110

Matches (M):

mId	mStadium	mGoals
1	Nacional	3
2	Nacional	2
2	San Carlos	3
5	Monum.	4

- ```
  SELECT   DISTINCT pName AS Nombre, mStadium AS Estadio
  FROM     Players, Matches
  WHERE    pId = mId
  ```

- ```
  SELECT      pName, AVG(mGoals)
  FROM        Players, Matches
  WHERE       pId = mId
  GROUP BY    pId, pName
  ```

Consultas anidadas y/o correlacionadas

Consultas anidada: consulta que contienen una subconsulta embebida en:

- `WHERE`
- `FROM`
- `HAVING`

Ejemplos

- ```
  SELECT  pName, pGoals
  FROM    Players
  WHERE   pGoals = ( SELECT MAX(pGoals)
                     FROM   Players )
  ```

- ```
  SELECT  DISTINCT pName, pYear
  FROM    Players, ( SELECT mId
                     FROM   Matches
                     WHERE  mGoals ≥ 3 )
  WHERE   pId = mId
  ```

Consultas anidadas y/o correlacionadas

Consulta correlacionada: consulta anidada que contienen una subconsulta embebida con una referencia a la consulta externa.

Ejemplo

```
SELECT   pName, pYear
FROM     Players AS P
WHERE    1 ≤ (  SELECT   AVG(mGoals)
                FROM     Matches AS M
                WHERE    M.mId = P.pId  )
```

¿cuál de las consultas anteriores
puede ser definida con una consulta NO anidada?

Outline

Algebra relacional

- Lenguaje de consultas procedural.
- Basado en operadores relacionales (algebra).

¿cuál es la utilidad del algebra relacional en BD?

Ventajas:

- Fácil de componer.
- Fácil de optimizar.

Operadores relacionales

Selección: $\sigma_\theta(R)$.

- θ es una combinación booleana $(\wedge, \vee)$ de terminos:

$$\begin{array}{rcl} \textit{attributo}_1 & \text{op} & \textit{attributo}_2 \\ \textit{attributo} & \text{op} & \textit{constante} \end{array}$$

con op $\in \{=, \leq, \geq, <, >\}$.

Proyección: $\pi_L(R)$.

- L = lista de atributos.

Operaciones de conjunto:

- Union: $R_1 \bigcup R_2$
- Intersección: $R_1 \bigcap R_2$
- Diferencia: $R_1 - R_2$

Operadores relacionales

Joins:

- Producto cruz: $R_1 \times R_2$
- θ-join: $R_1 \bowtie_\theta R_2 = \sigma_\theta(R_1 \times R_2)$
  - θ es una combinación booleana $(\wedge, \vee)$ de terminos:

$$\begin{array}{rcl} \mathit{attributo}_1 & \text{op} & \mathit{attributo}_2 \\ \mathit{attributo} & \text{op} & \mathit{constante} \end{array}$$

con op $\in \{=, \leq, \geq, <, >\}$.

- Equi-join: $R_1 \bowtie_\phi R_2 = \sigma_\phi(R_1 \times R_2)$
  - ϕ solo contine igualdades.
- Natural-join: $R_1 \bowtie R_2 = \sigma_\phi(R_1 \times R_2)$
  - $\phi = \bigwedge_{a \in \text{att}(R_1) \cap \text{att}(R_2)} R_1.a = R_2.a.$

Ejemplo de consultas en algebra relacional

Ejemplos

Players (P):

id	name	year	goals
1	x	1987	100
2	y	1990	59
3	y	1990	88
4	z	1983	110

Matches (M):

id	stadium	mgoals
1	Nacional	3
2	Nacional	2
2	San Carlos	3
NULL	Monum.	4

- $\pi_{\text{name,year}}(\text{P})$
- $\sigma_{\text{goals}\geq 100 \,\vee\, \text{goals}\leq 60}(\text{P})$
- $\pi_{\text{goals}}\left(\ \sigma_{\text{year}=1990}(\text{P})\ \right)$
- $\text{P} \bowtie_{P.id=M.id} \text{M}$
- $\text{P} \bowtie \text{M}$

Semántica algebra relacional

Players (P):

id	name	year	goals
1	x	1987	100
2	y	1990	59
3	y	1990	88
4	z	1983	110

Matches (M):

id	stadium	mgoals
1	Nacional	3
2	Nacional	2
2	San Carlos	3
`NULL`	Monum.	4

¿cuál es el resultado de la consulta $\pi_{\text{name,year}}(\text{P})$?

Dos tipos de semánticas para algebra relacional:

- Set semantics.
- Bag semantics (`SQL`).

¿por qué usar set semantics o bag semantics?

Algebra relacional extendida

Rename: $\rho_{\text{old\_att} \to \text{new\_att}}(R)$

Eliminación de duplicados: $\delta(R)$

Group-by con aggregación: $\gamma_{G,A}(R)$

- G: lista de atributos ha agrupar.
- A: lista de elementos de la forma:

$$f(\text{agg\_att}) \to \text{new\_att}$$

con $f \in \{\texttt{MIN}, \texttt{MAX}, \texttt{SUM}, \texttt{AVG}, \ldots\}$.

Sorting (ordenar): $\tau(R)$

Algebra relacional extendida

Ejemplos

Players (P):

id	name	year	goals
1	x	1987	100
2	y	1990	59
3	y	1990	88
4	z	1983	110

Matches (M):

id	stadium	mgoals
1	Nacional	3
2	Nacional	2
2	San Carlos	3
NULL	Monum.	4

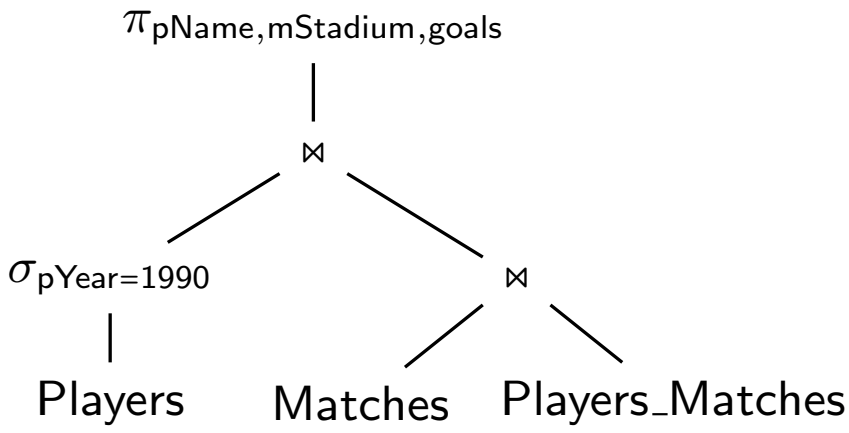
- $P \bowtie \rho_{mgoals \to goals}(M)$.
- $\delta(\pi_{\text{name,year}}(P))$.
- $\gamma_{\text{year,AVG(goals)} \to \text{GperY}}(P)$.

¿cómo evaluarían esta consulta Q?

Players(pId, pName, pYear)
Matches(mId, mStadium, mDate)
Players_Matches(pId, mId, goals)

$$\pi_{\text{pName,mStadium,goals}}(\sigma_{\text{pYear=1990}}(\text{Players}) \bowtie (\text{Matches} \bowtie \text{Players\_Matches}))$$

Plan lógico (= árbol de parsing) de Q:



El **plan lógico** será nuestro punto inicial para la **evaluación** de la consulta.

¿cómo convertirían esta consulta a algebra relacional?

```
SELECT   pName, mStadium, goals
FROM     Players AS P, Matches AS M, Players_Matches AS PM
WHERE    P.pId = PM.pId AND PM.mId = M.mId AND
         P.pYear ≥ 1985
```

Desde SQL a Algebra relacional

Idea (intuitiva)

$$\begin{array}{ll} \texttt{SELECT} & \text{att}_1, \ldots, \text{att}_n \\ \texttt{FROM} & R_1, \ldots, R_m \\ \texttt{WHERE} & \varphi \end{array}$$

1. Combine las relaciones en el `FROM` con **productos cruz**:

$$R_1 \times \ldots \times R_n$$

2. Aplique **selección** con la condición dada en el `WHERE`:

$$\sigma_{\varphi}(R_1 \times \ldots \times R_n)$$

3. Aplique **proyección** con las atributos dados en el `SELECT`:

$$\pi_{\text{att}_1,\ldots,\text{att}_n} (\sigma_{\varphi}(R_1 \times \ldots \times R_n))$$

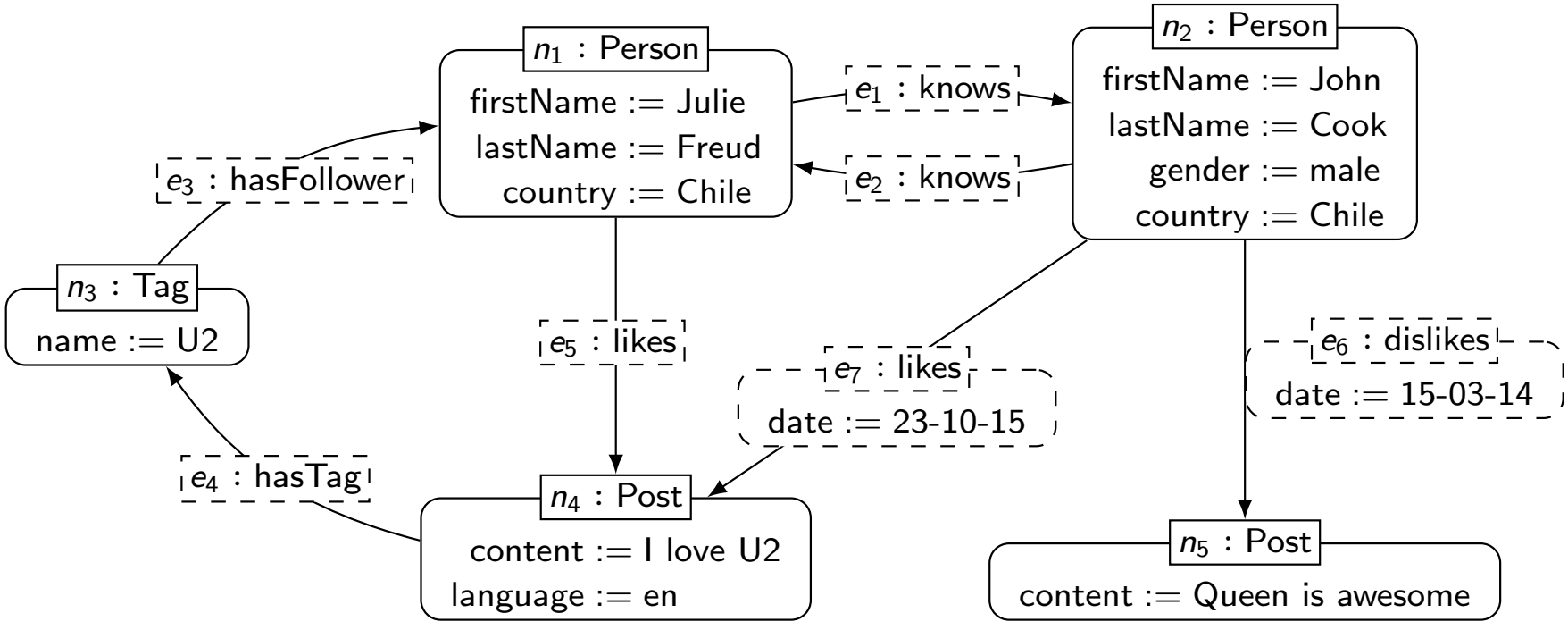
¿es este resultado un "buen" plan de evaluación de la consulta?

Outline

Base de datos de grafos

- Modelo alternativo a relacional.
- Para datos con estructura “naturalmente de grafo”.
- Lenguaje de consultas orientado a grafos.
- Algunos sistemas:
  1. Neo4J
  2. Microsoft Azure Cosmos DB
  3. OrientDB
  4. ArangoDB
- Modelo de datos: property graph.

Modelo de datos: property graph



Modelo de datos: property graph

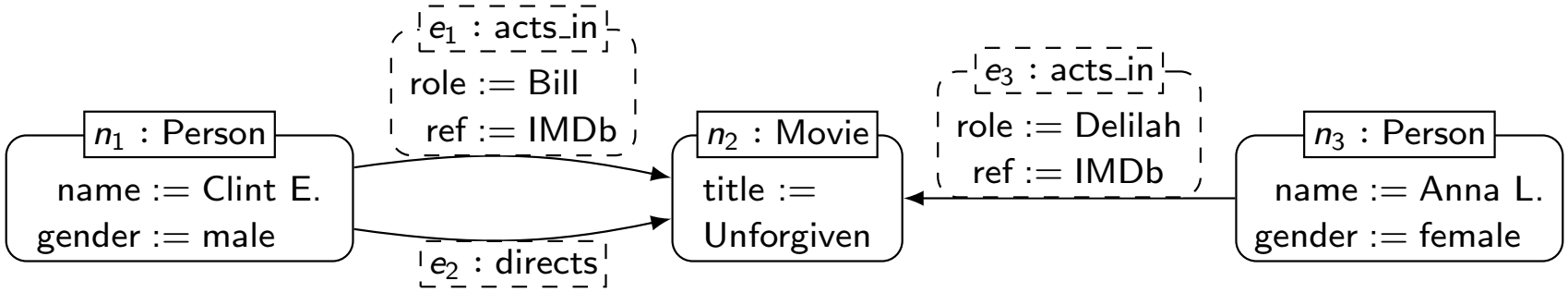
Definición

Una **base de datos de grafos** $(N, E, L, K, V, \rho, \lambda, \sigma)$ esta compuesto por:

1. Un conjunto de nodos N.
2. Un conjunto de aristas E.
3. Un conjunto de labels L.
4. Un conjunto de keys K.
5. Un conjunto de valores V.
6. Una función $\rho : E \to N \times N$.
7. Una función parcial $\lambda : (N \cup E) \to 2^L$.
8. Una función parcial $\sigma : (N \cup E) \times K \to V$.

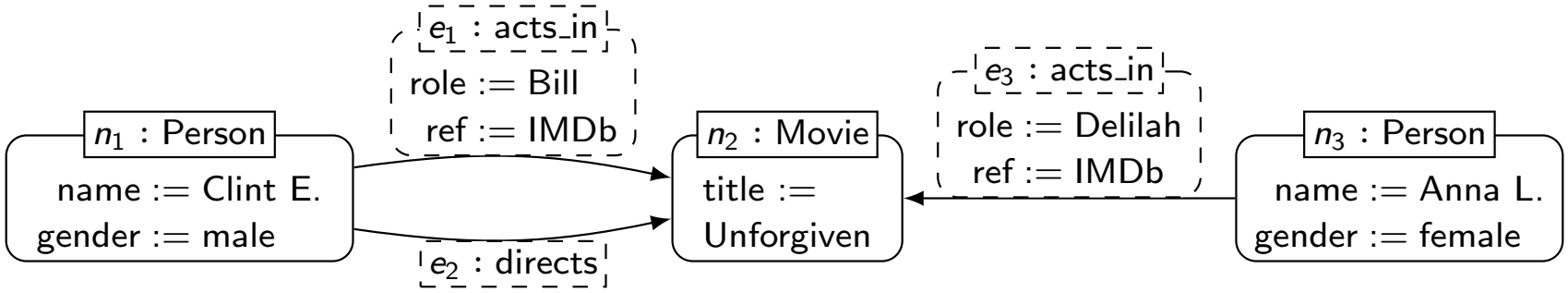
¿cuál es el significado de cada componente en la BD de grafos?

Ejemplo de property graph



$V = \{n_1, n_2, n_3\}$ $\quad$ $E = \{e_1, e_2, e_3\}$

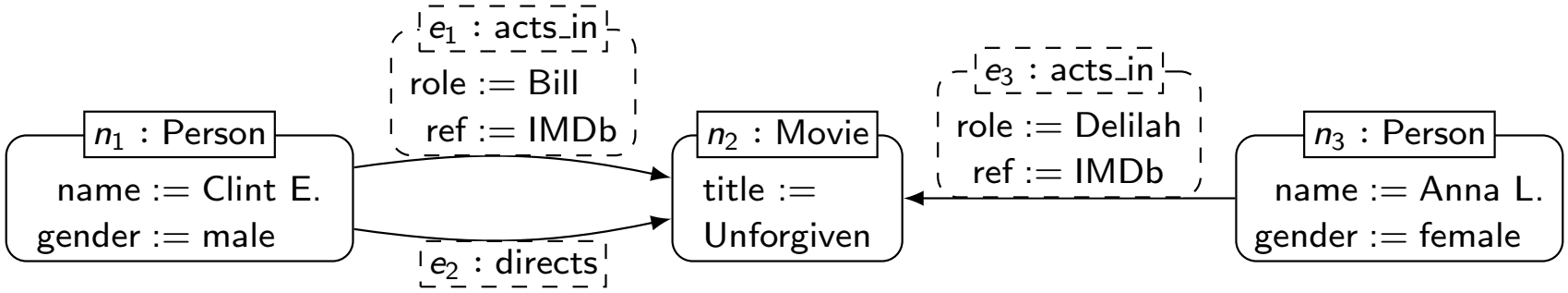
Ejemplo de property graph



$$V = \{n_1, n_2, n_3\} \qquad E = \{e_1, e_2, e_3\}$$

$$\rho(e_1) = (n_1, n_2) \qquad \rho(e_2) = (n_1, n_2)$$
$$\rho(e_3) = (n_3, n_2)$$

Ejemplo de property graph



$V = \{n_1, n_2, n_3\}$ $\quad$ $E = \{e_1, e_2, e_3\}$

$\rho(e_1) = (n_1, n_2)$ $\quad$ $\rho(e_2) = (n_1, n_2)$

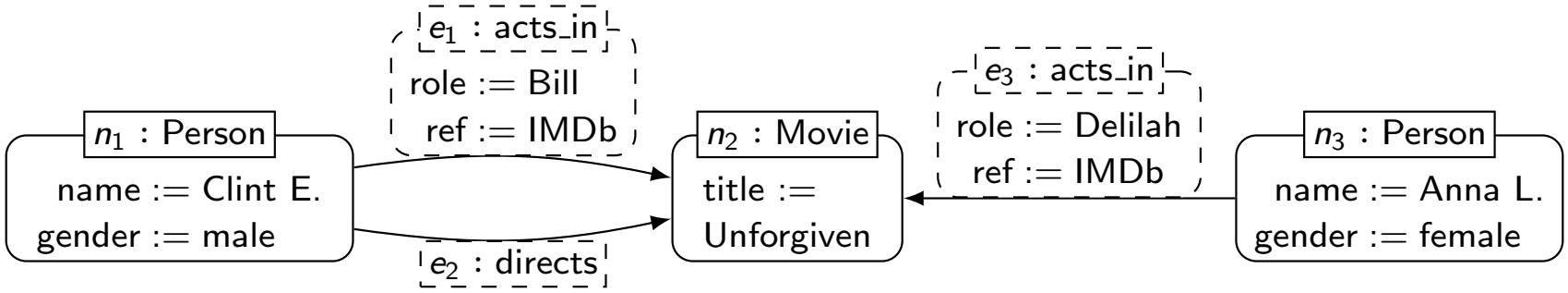
$\rho(e_3) = (n_3, n_2)$

$\lambda(n_1) = \text{Person}$ $\quad$ $\lambda(n_2) = \text{Movie}$

$\lambda(n_3) = \text{Person}$ $\quad$ $\lambda(e_1) = \text{acts\_in}$

$\lambda(e_2) = \text{directs}$ $\quad$ $\lambda(e_3) = \text{acts\_in}$

Ejemplo de property graph



$V = \{n_1, n_2, n_3\}$ $\quad E = \{e_1, e_2, e_3\}$

$\rho(e_1) = (n_1, n_2)$ $\quad \rho(e_2) = (n_1, n_2)$

$\rho(e_3) = (n_3, n_2)$

$\lambda(n_1) = \text{Person}$ $\quad \lambda(n_2) = \text{Movie}$

$\lambda(n_3) = \text{Person}$ $\quad \lambda(e_1) = \text{acts\_in}$

$\lambda(e_2) = \text{directs}$ $\quad \lambda(e_3) = \text{acts\_in}$

$\sigma(n_1, \mathit{name}) = \text{Clint E.}$

$\sigma(n_1, \mathit{gender}) = \text{male}$

$\sigma(n_2, \mathit{title}) = \text{Unforgiven}$

$\sigma(n_3, \mathit{name}) = \text{Anna L.}$

$\sigma(n_3, \mathit{gender}) = \text{female}$

$\sigma(e_1, \mathit{role}) = \text{Bill}$

$\sigma(e_1, \mathit{ref}) = \text{IMDb}$

$\sigma(e_3, \mathit{role}) = \text{Delilah}$

$\sigma(e_3, \mathit{ref}) = \text{IMDb}$

Lenguaje de consultas para property graphs

Forma basica:

`SELECT`	< atributos >
`MATCH`	< patrones >
`WHERE`	< condiciones >

< atributos >: lista de atributos de variables

- ?n . atributo

< patrones >: lista de patrones separados por coma

< condiciones >: lista de condiciones booleanas

- φ_1 `AND` φ_2, φ_1 `OR` φ_2, `NOT` φ
- atributo$_1$ $\sim$ atributo$_2$ donde $\sim \in \{=, \leq, \geq, \ldots\}$.
- atributo$_1$ $\sim$ *constante* donde $\sim \in \{=, \leq, \geq, \ldots\}$, etc...

Lenguaje de consultas para property graphs

Forma basica:

```
SELECT    < atributos >
MATCH     < patrones >
WHERE     < condiciones >
```

Patrón de nodo:

- (?n)
- (?n :label1 ... :labelN)
- (?n {key1:val1, ..., keyM:valM})
- (?n :label1 ... :labelN {key1:val1, ..., keyM:valM})

Lenguaje de consultas para property graphs

Forma basica:

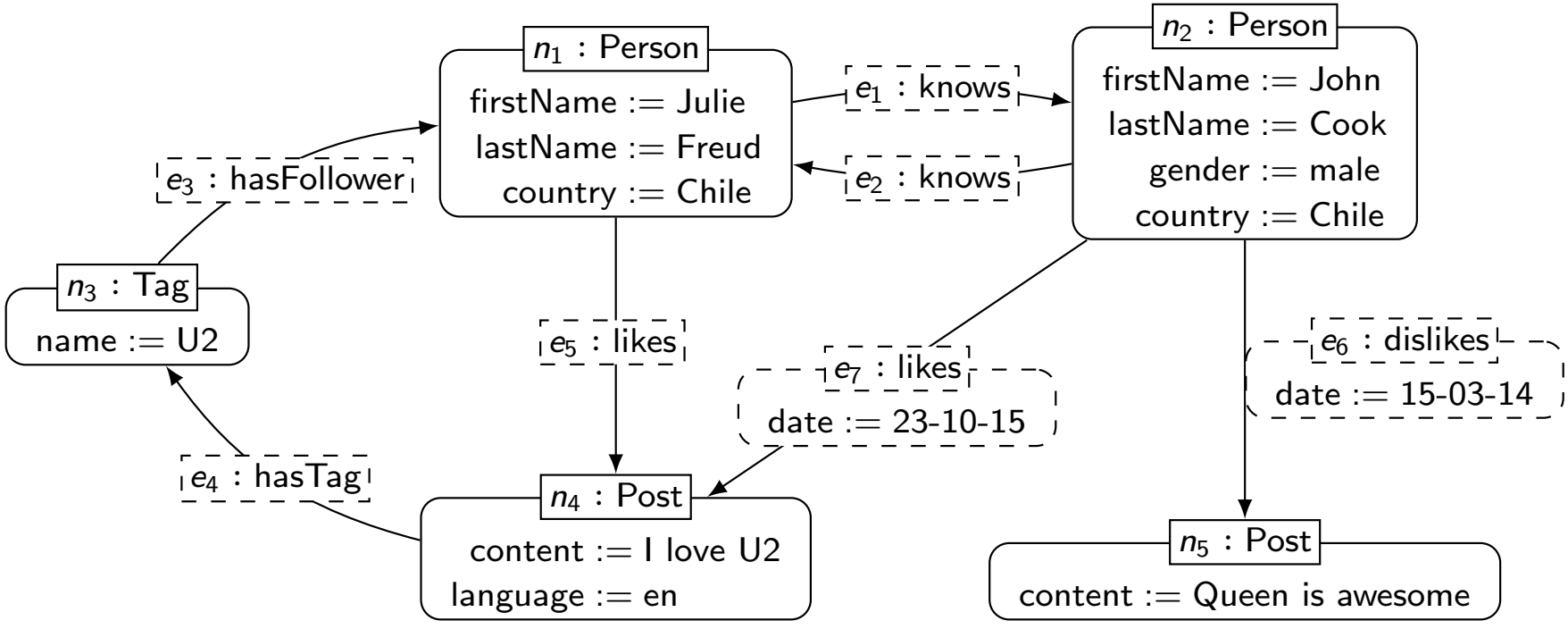
```
SELECT    < atributos >
MATCH     < patrones >
WHERE     < condiciones >
```

Patrón de aristas:

- NP -> NP
- NP -[?e]-> NP
- NP -[:label1 ... :labelN]-> NP
- NP -[?e :label1 ... :labelN]-> NP
- NP -[{key1:val1,...,keyM:valM}]-> NP
- NP -[?e {key1:val1,...,keyM:valM}]-> NP
- NP -[?e :label1 ... :labelN {key1:val1,...,keyM:valM}]-> NP

donde NP es un patrón de nodo.

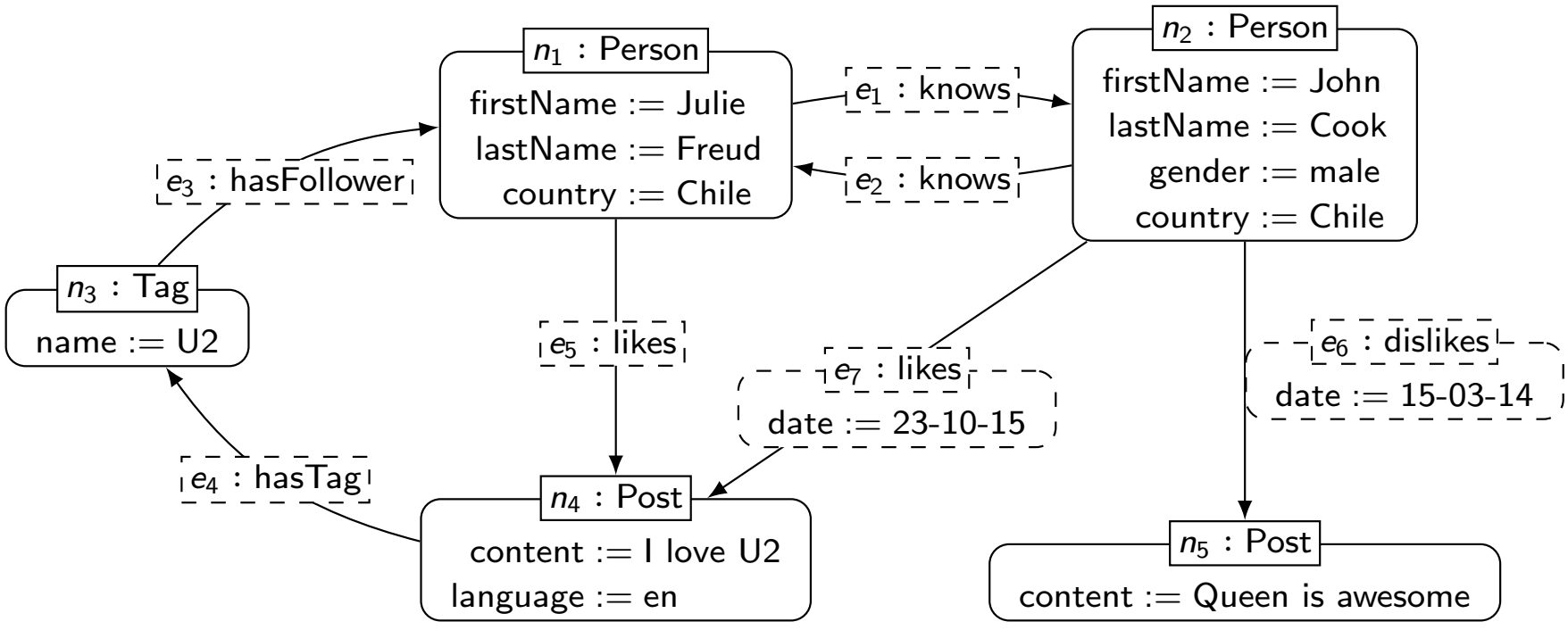
Ejemplos de consultas para property graphs



Nombres de todas las personas en la BD

```
SELECT   ?n.firstName, ?n.lastName
MATCH    (?n :Person)
```

Ejemplos de consultas para property graphs

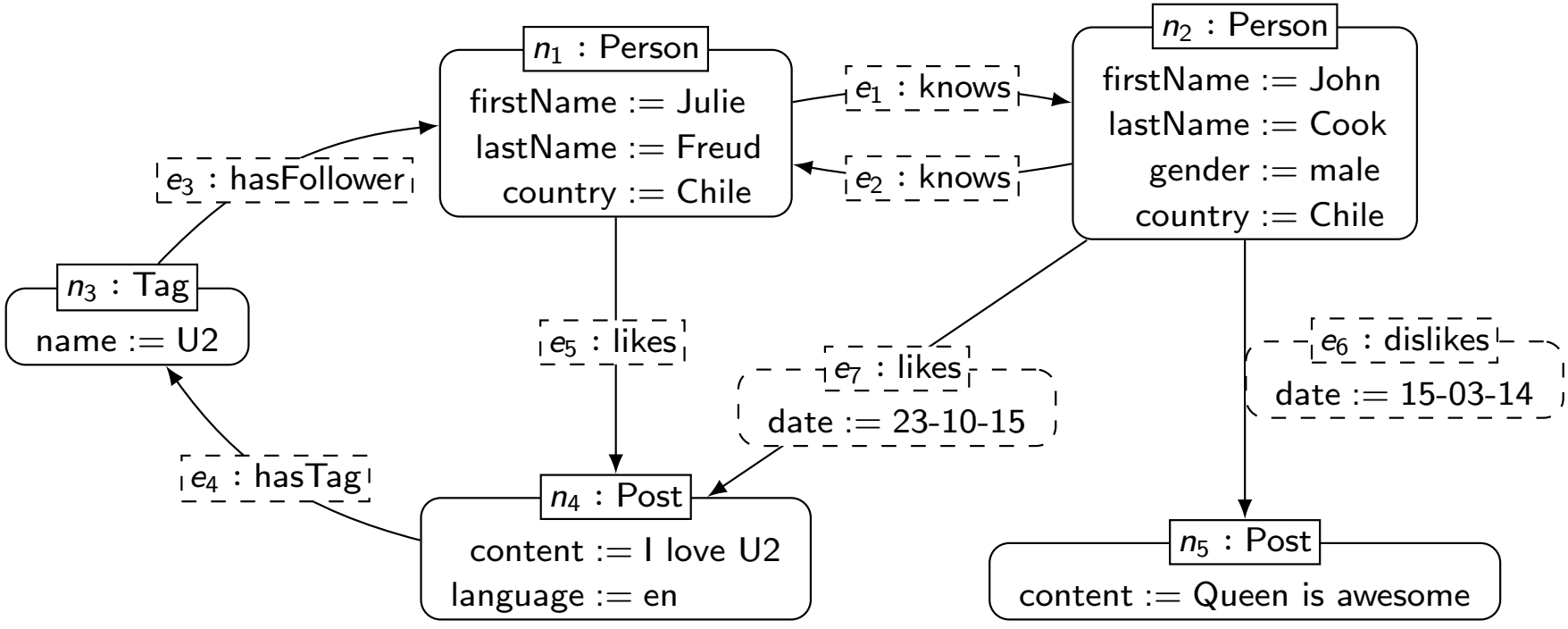


Nombres de todas las personas de Chile en la BD

```
SELECT  ?n.firstName, ?n.lastName
MATCH   (?n :Person)
WHERE   ?n.country == "Chile"
```

```
SELECT  ?n.firstName, ?n.lastName
MATCH   (?n :Person {country: "Chile"})
```

Ejemplos de consultas para property graphs

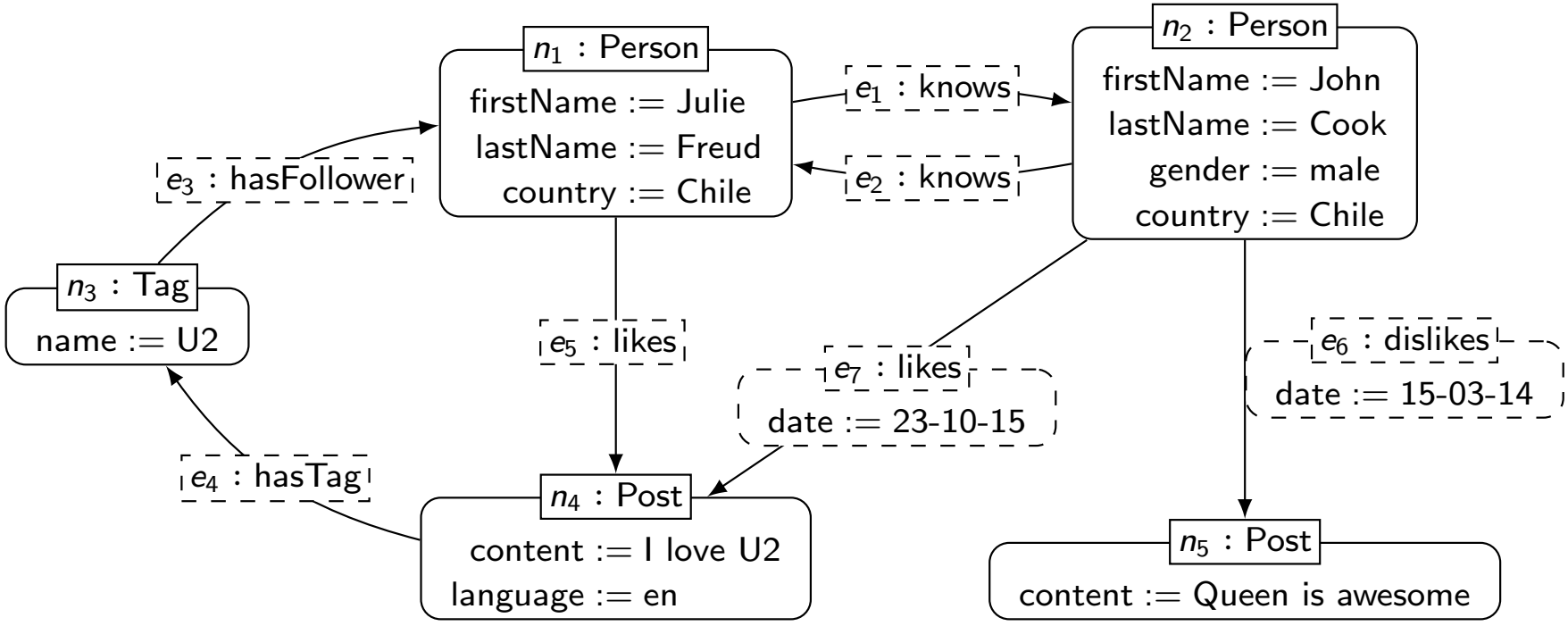


Todas las personas que conocen a Julie

```
SELECT  ?n.firstName, ?n.lastName
MATCH   (?n) -[:knows]->
        ({firstName:"Julie"})
```

```
SELECT  ?n.firstName, ?n.lastName
MATCH   (?n) -[:knows]-> (?m)
WHERE   ?m.firstName == "Julie"
```

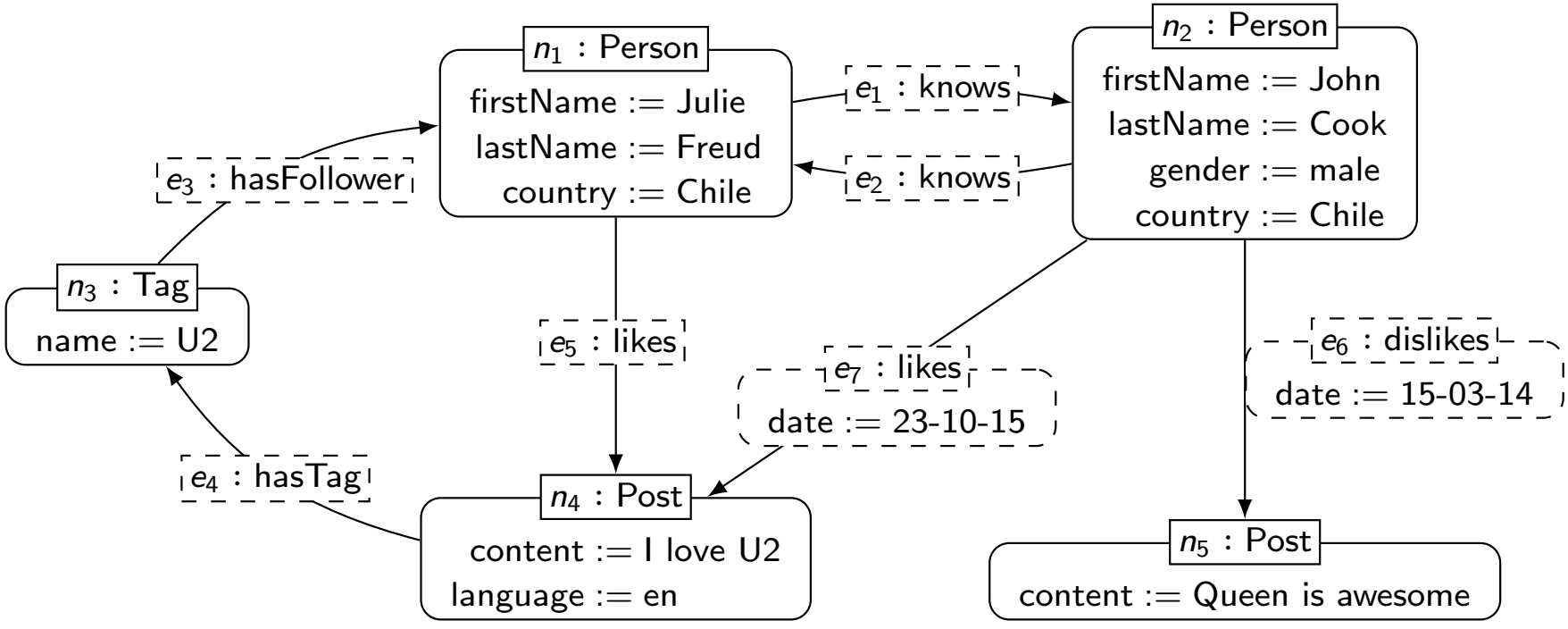
Ejemplos de consultas para property graphs



Personas que conocen a Julie y le gusta uno de sus post

```
SELECT   ?n.firstName, ?n.lastName
MATCH    (?n) -[:knows]-> (?m) -[:likes]-> (?k :Post), (?n) -[:likes]-> (?k)
WHERE    ?m.firstName == "Julie"
```

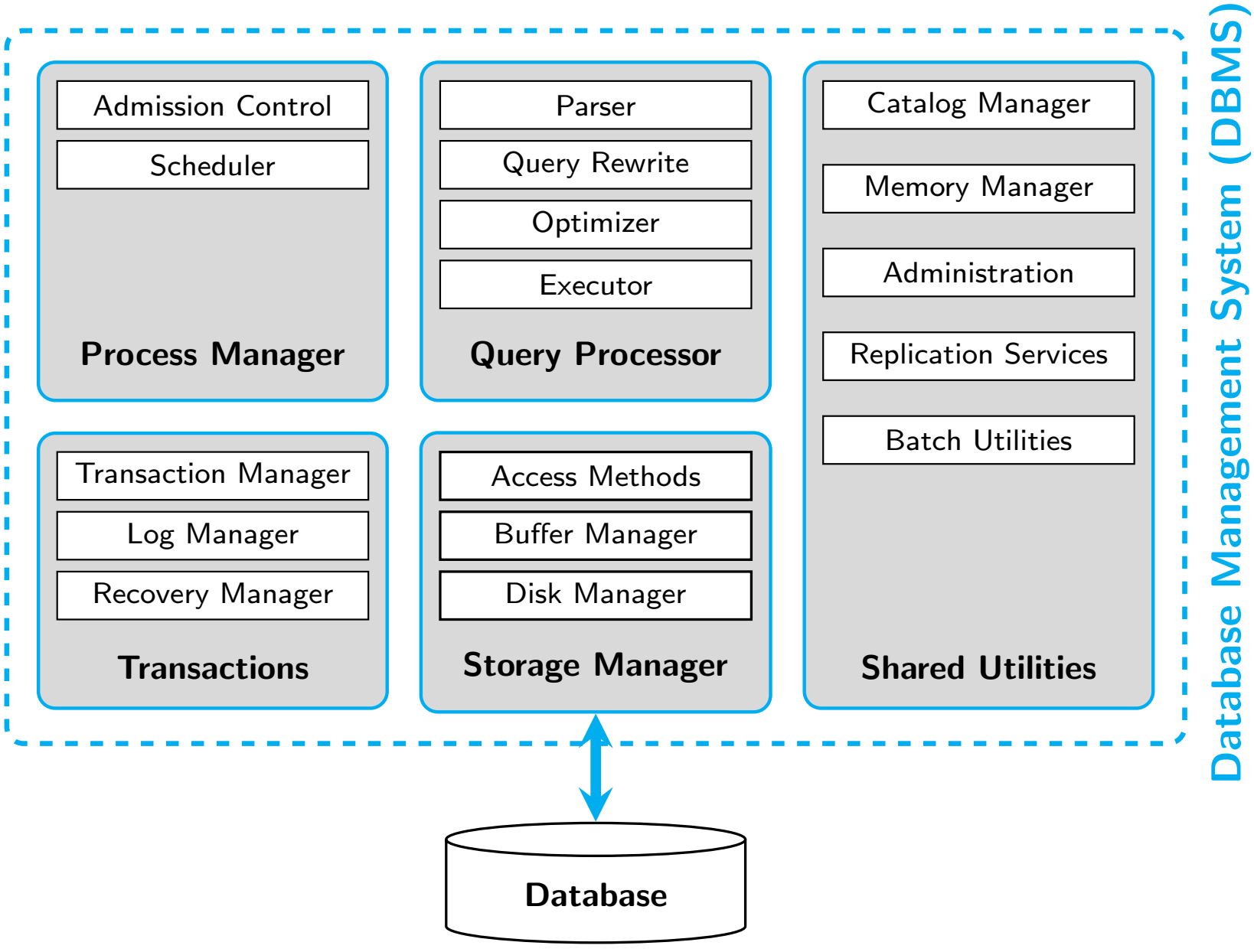
Ejemplos de consultas para property graphs



Para efectos prácticos de los laboratorios,
en el curso nos concentraremos en este lenguaje de consultas (fragmento)

Outline

Arquitectura de un Sistema de Bases de Datos (DBMS)

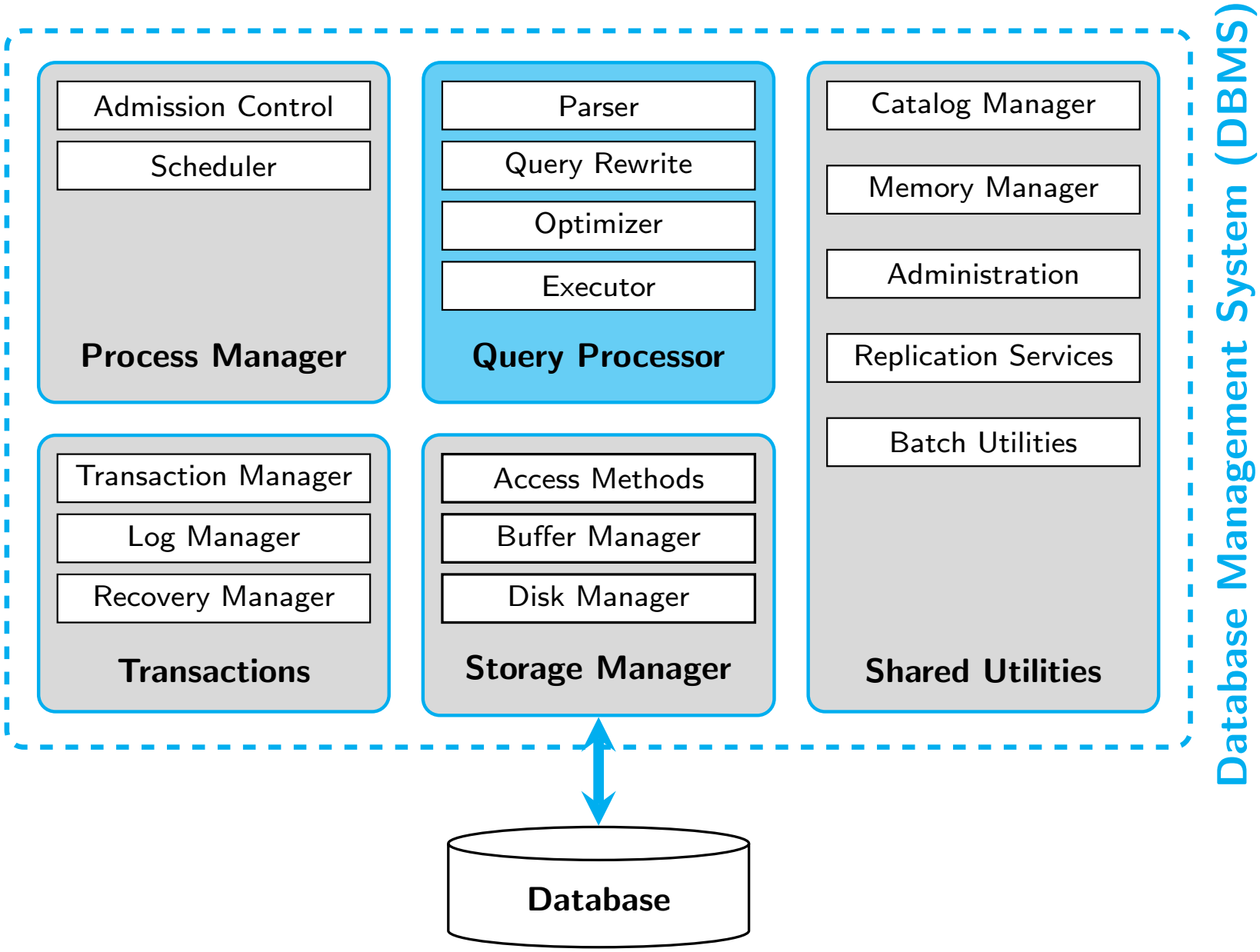


La vida de una consulta SQL

Considere la siguiente consulta:

```
Q  =  SELECT   pName, mStadium, goals
      FROM     Players AS P, Matches AS M, Players_Matches AS PM
      WHERE    P.pId = PM.pId AND PM.mId = M.mId AND
               P.pYear ≥ 1985
```

Procesador de consultas



Procesador de consultas

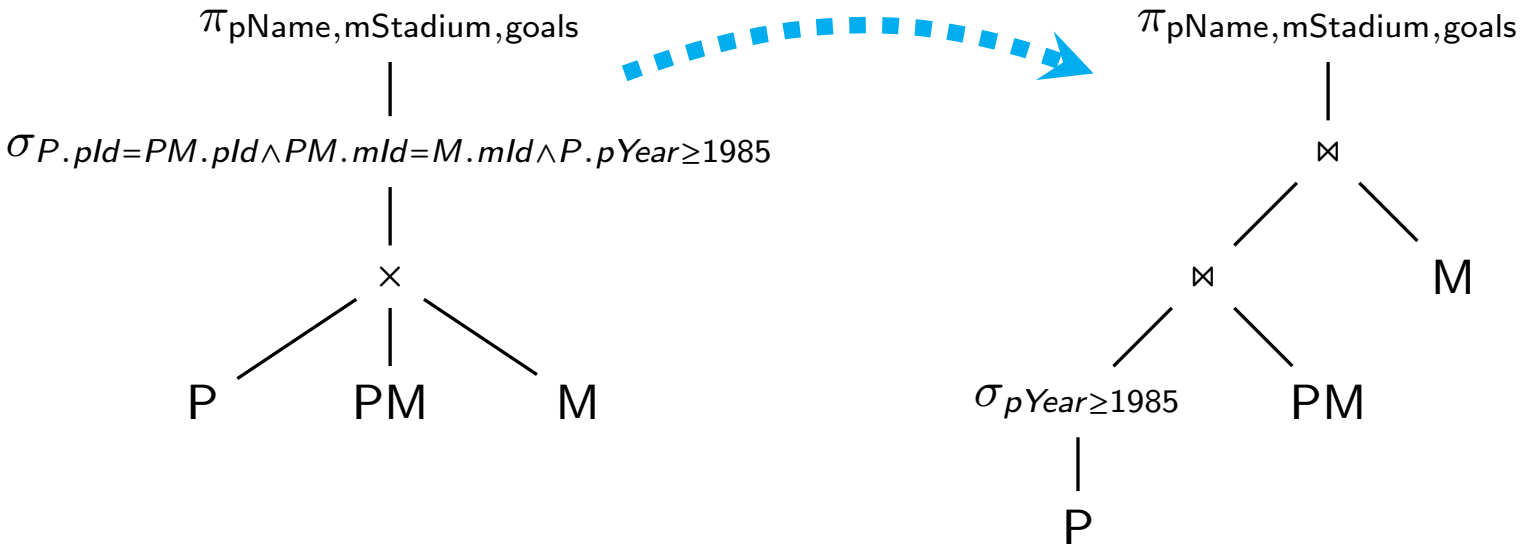
Paso 1: Parser

- Valida la consulta (correctitud, autorización, etc).
- Convierte consulta en formato interno (algebra relacional).
- Optimizaciones menores (numéricas, etc).

Paso 2: Reescritura de consulta

- "Desanidación" de la consulta (flattening).
- Reescribe consulta aplicando reglas de algebra relacional.
- Crea un set de planes lógicos para ser optimizados.

Procesador de consultas



Procesador de consultas

Paso 3: Optimizador

- Encuentra el **plan físico** mas eficiente para ejecutar la consulta.
- El optimizador debe considerar:
  - Tamaño de cada relación y distribución de sus datos.
  - Distintos accesos a las relaciones (índices, etc).
  - Distintos planes lógicos para la misma consulta.
  - Distintos algoritmos para un mismo operador relacional.

Paso 4: Ejecución

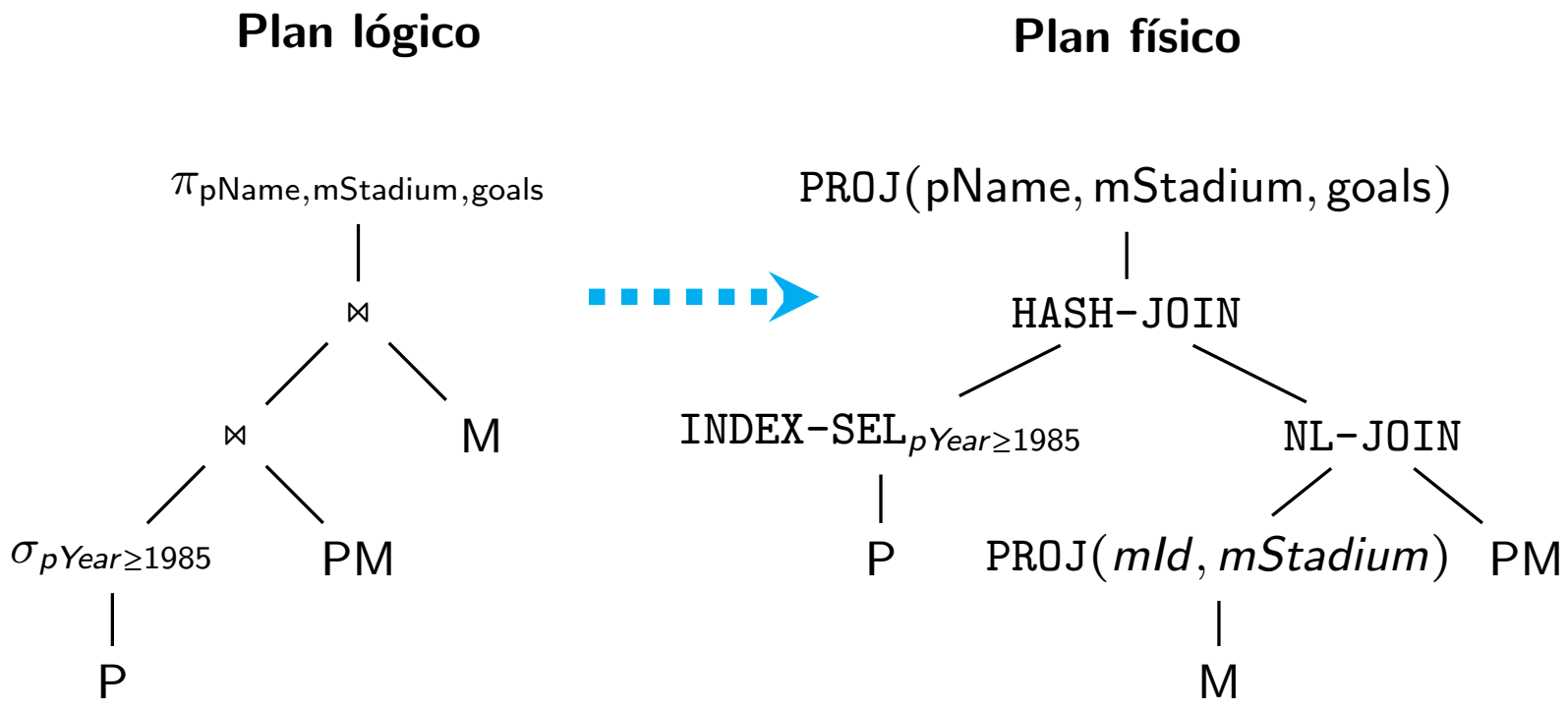
- Ejecuta el plan óptimo encontrado por Optimizador (**pipelining**).

Plan físico

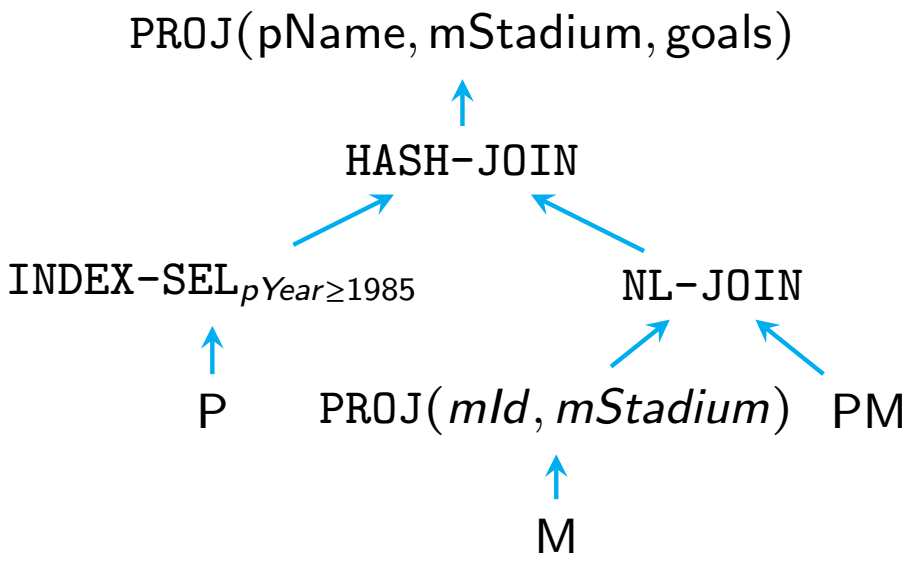
Plan físico $\approx$ Plan lógico con anotaciones adicionales.

- Selecciona el **access path** para cada relación
  - Uso de índices o (simplemente) file scan.
- Decide el algoritmo ha utilizar por cada operador.
- Planifica secuencia y orden de los operadores.

Plan lógico vs. Plan físico

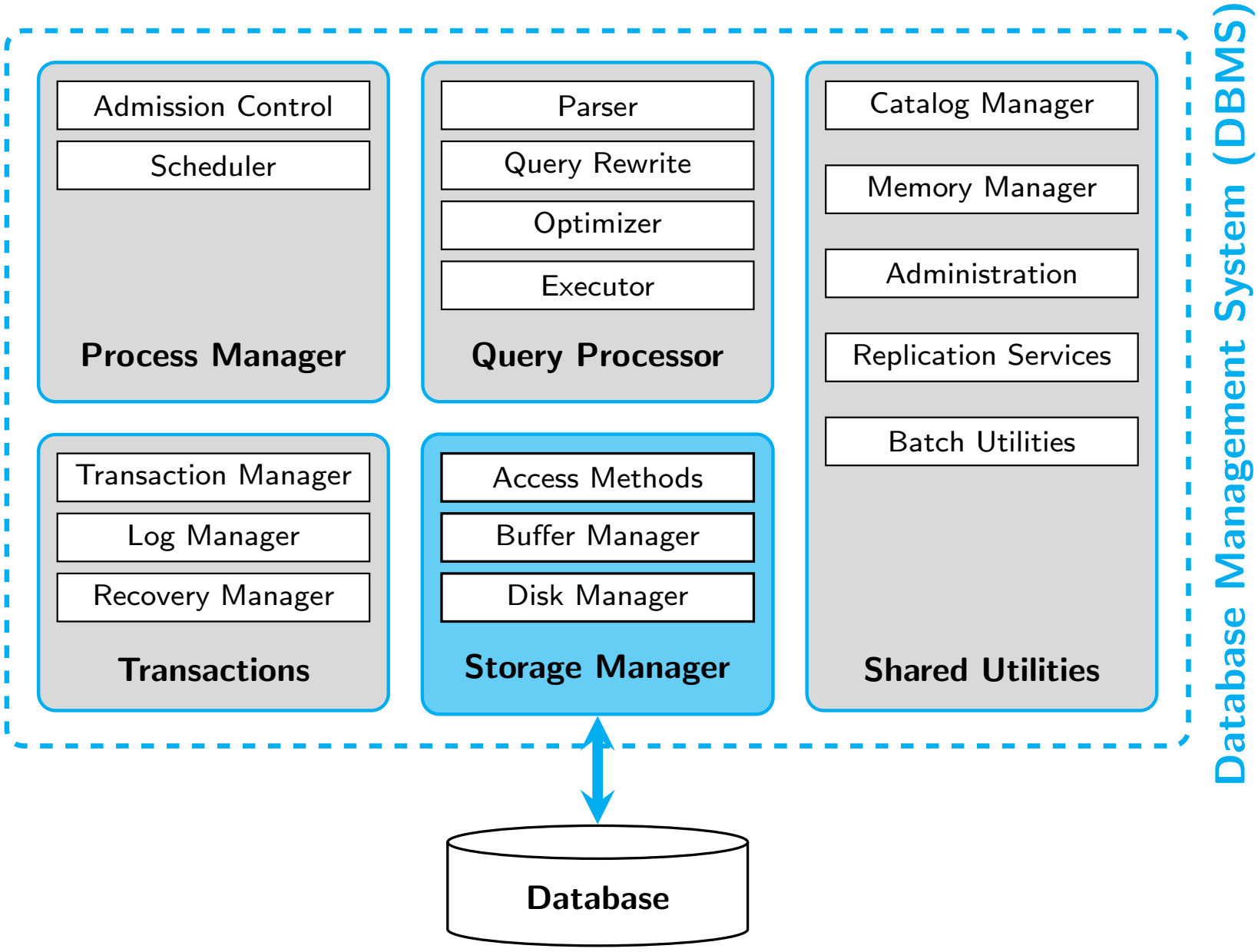


Pipelining (Ejecución)



- Ejecución en serie del plan físico.
- Cada operador retorna tuplas a su operador **padre**.
- Cada tupla es retornada "as soon as possible".

Manejador de almacenamiento



Manejador de almacenamiento

Disk Manager

- Acceso para almacenamiento secundario (disco duro, etc).
- Organizador de tuplas (**Heapfiles**).

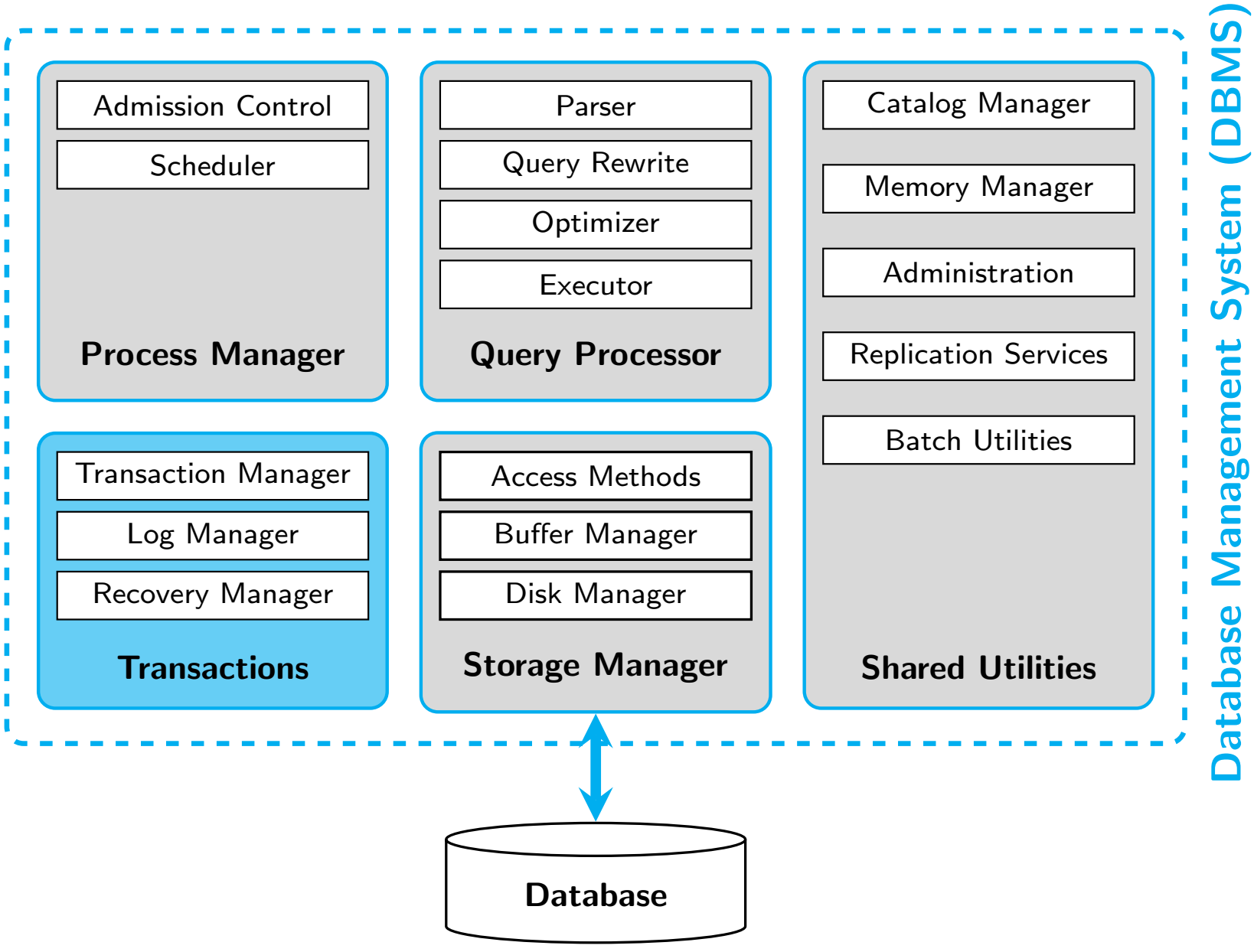
Buffer Manager

- Manejo de datos en memoria ($|\text{memoria}| << |\text{disco duro}|$).
- Optimiza la cantidad de accesos al disco duro (I/O).
- Importante para transacciones (ACID).

Access Methods

- Todo tipo de índices.
- Organización eficiente de los datos.

Transacciones



Transacciones

ACID = **A**tomicity
Consistency
Isolation
Durability

Transaction Manager

- Encargado de asegurar **Isolation** y **Consistency**.

Log y Recovery Manager

- Encargado de asegurar **Atomicity** y **Durability**.