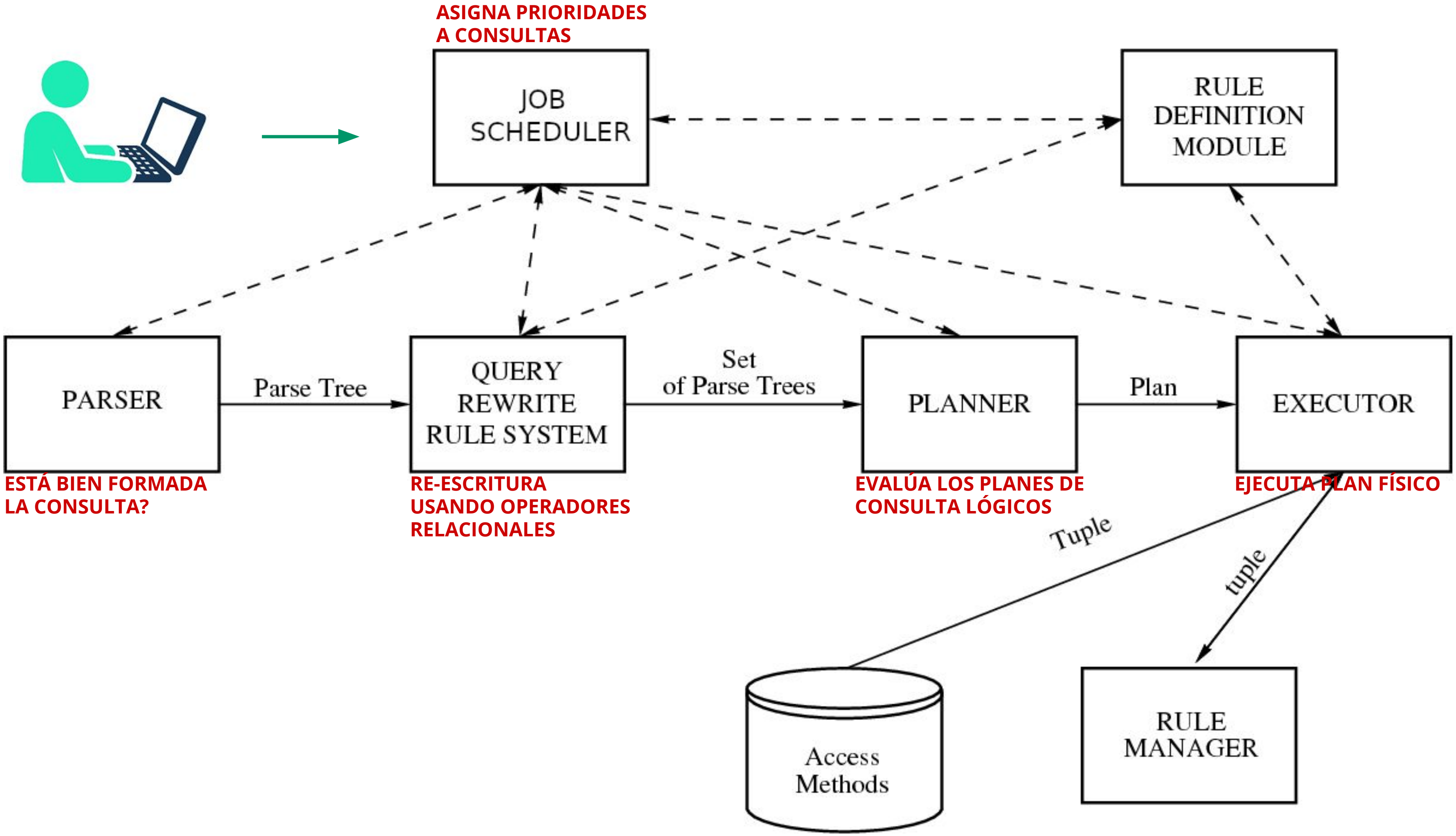
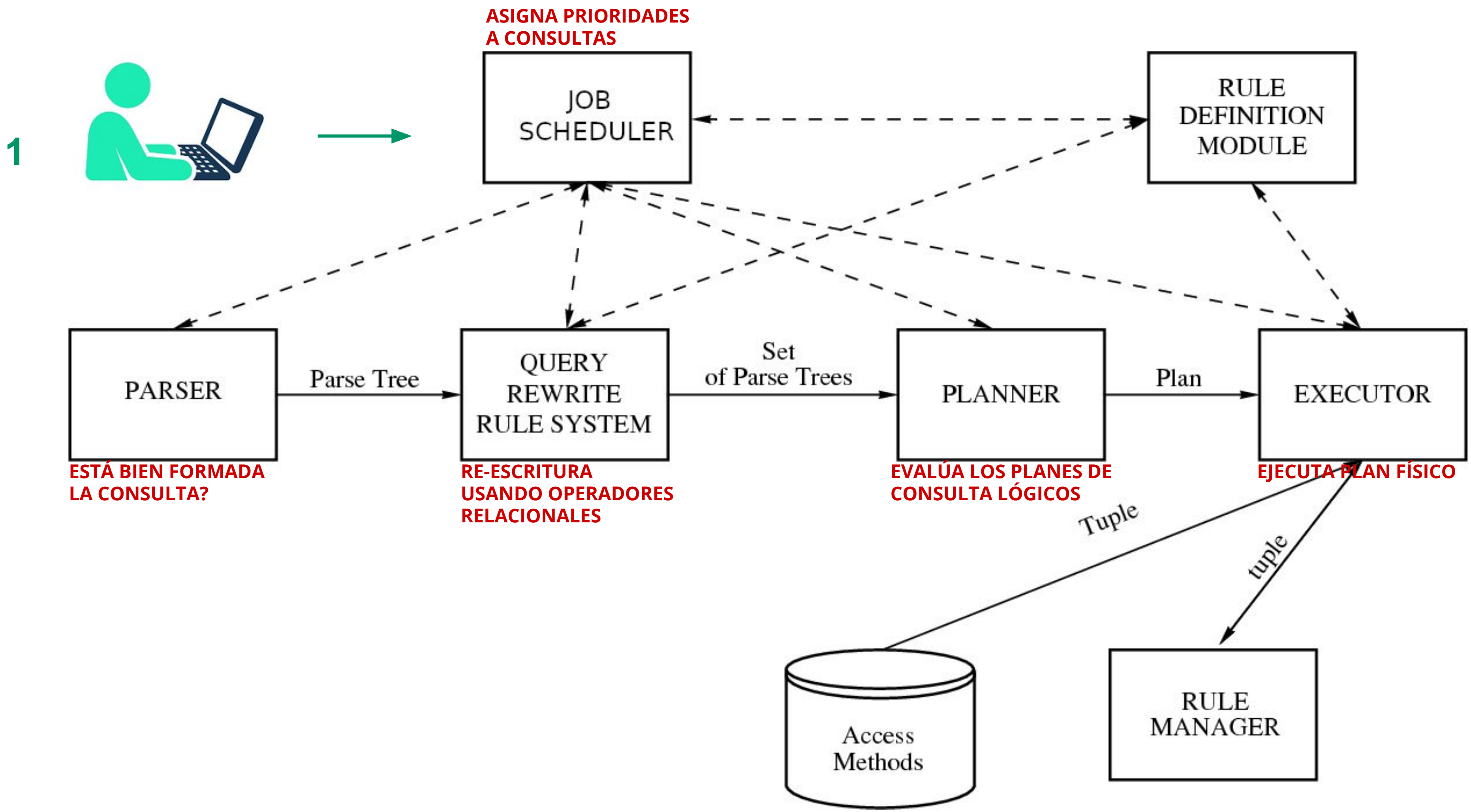
Bases de Datos

Ejecución de consultas

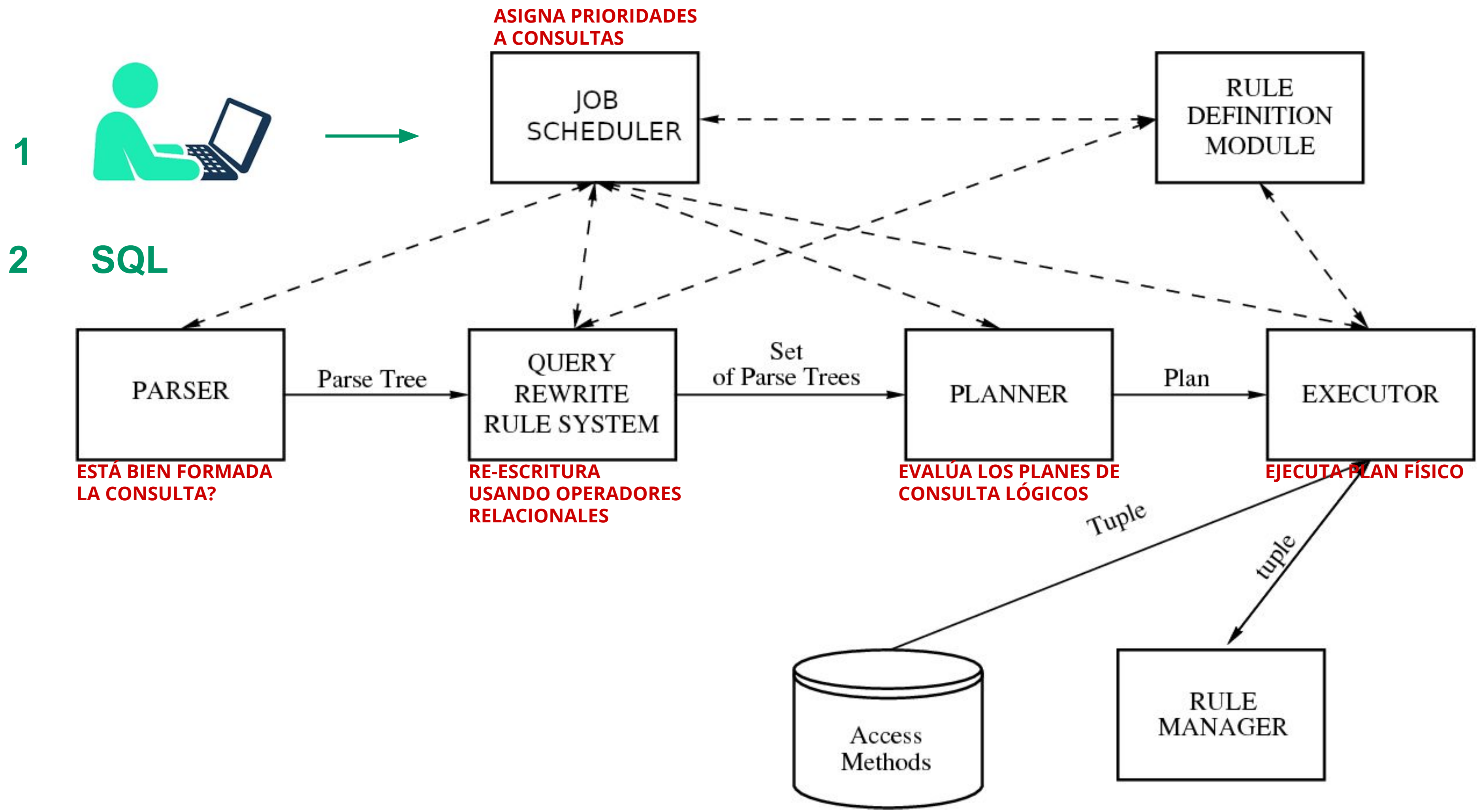
Metodología



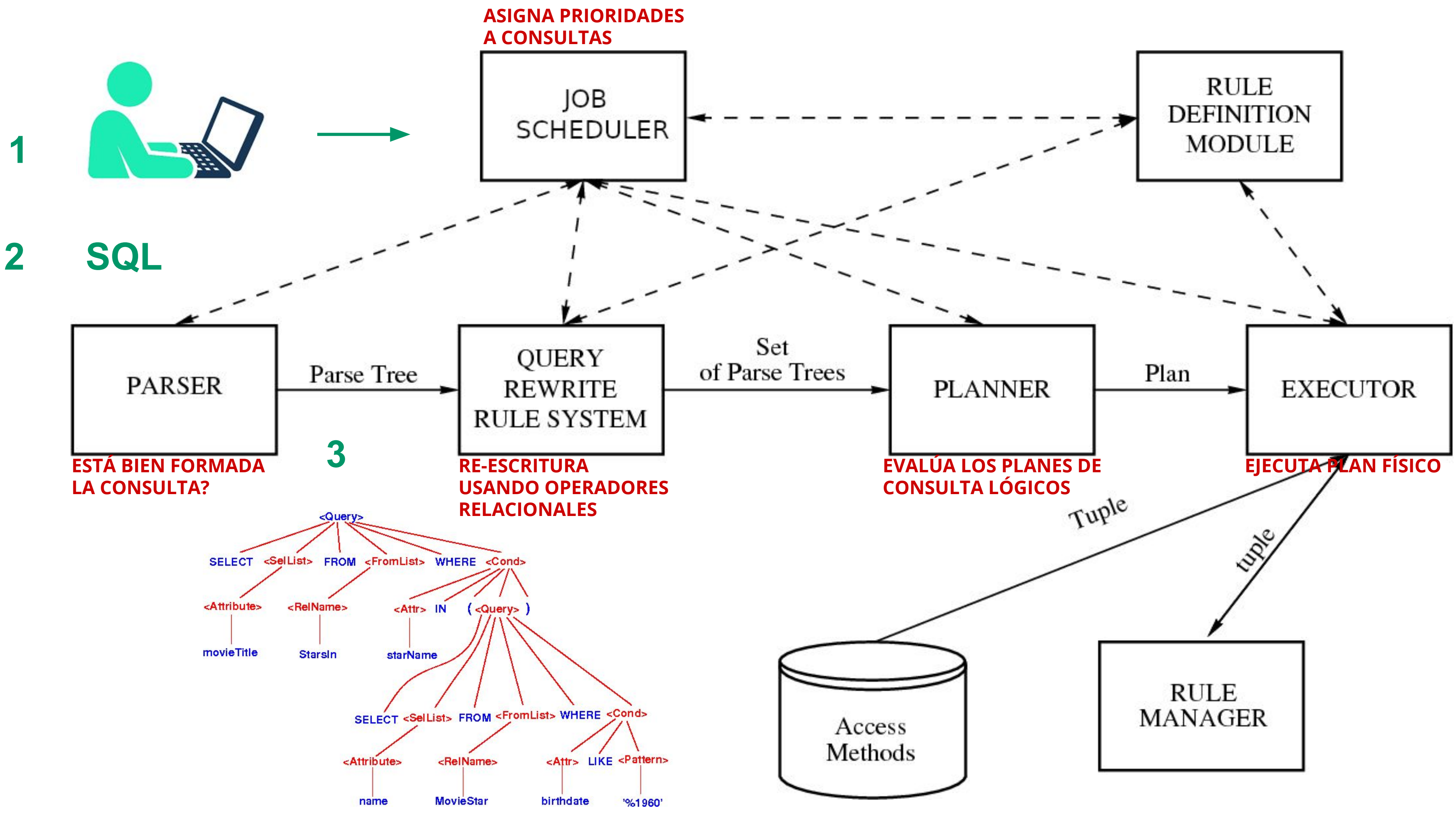
Metodología



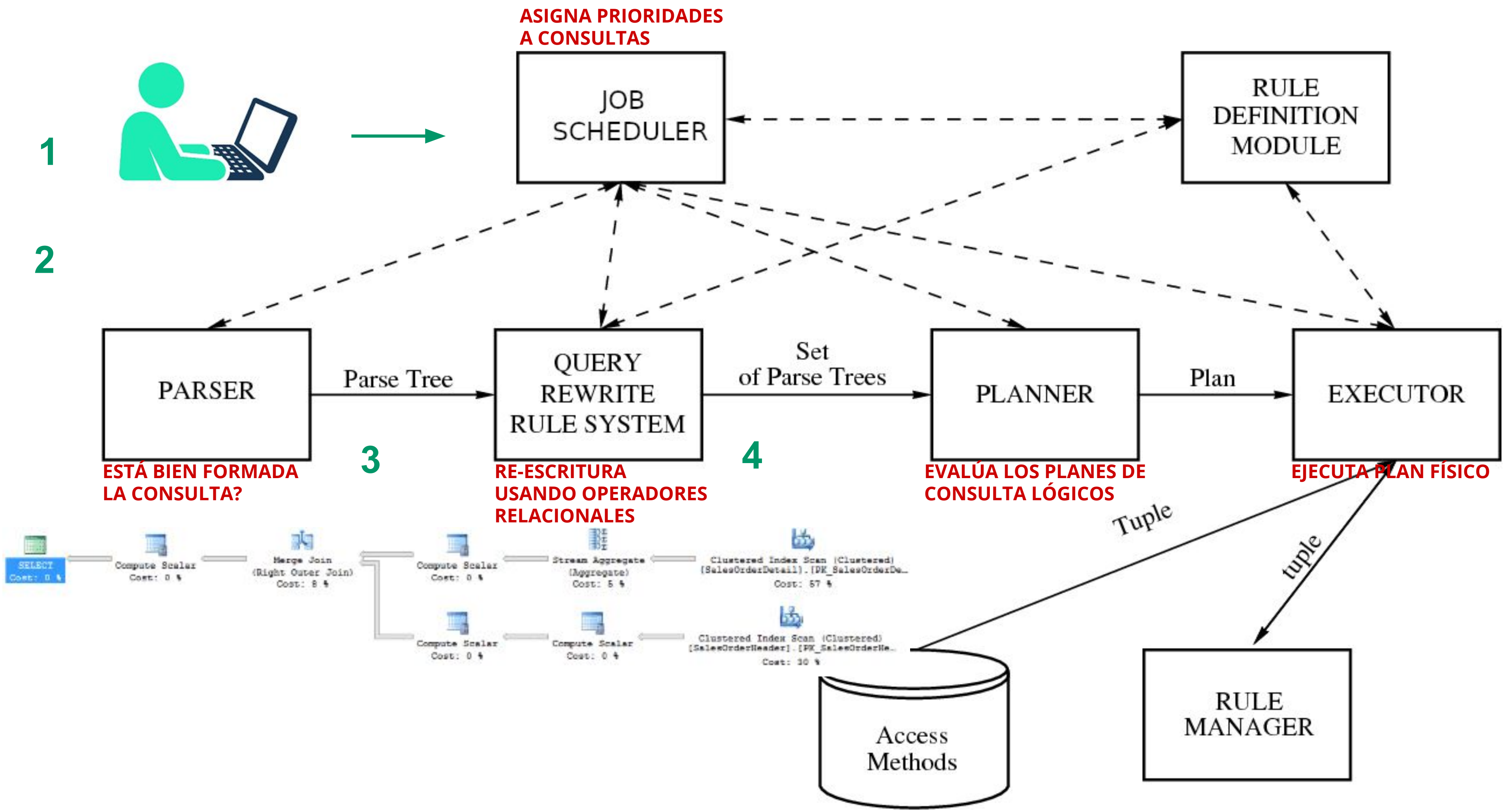
Metodología



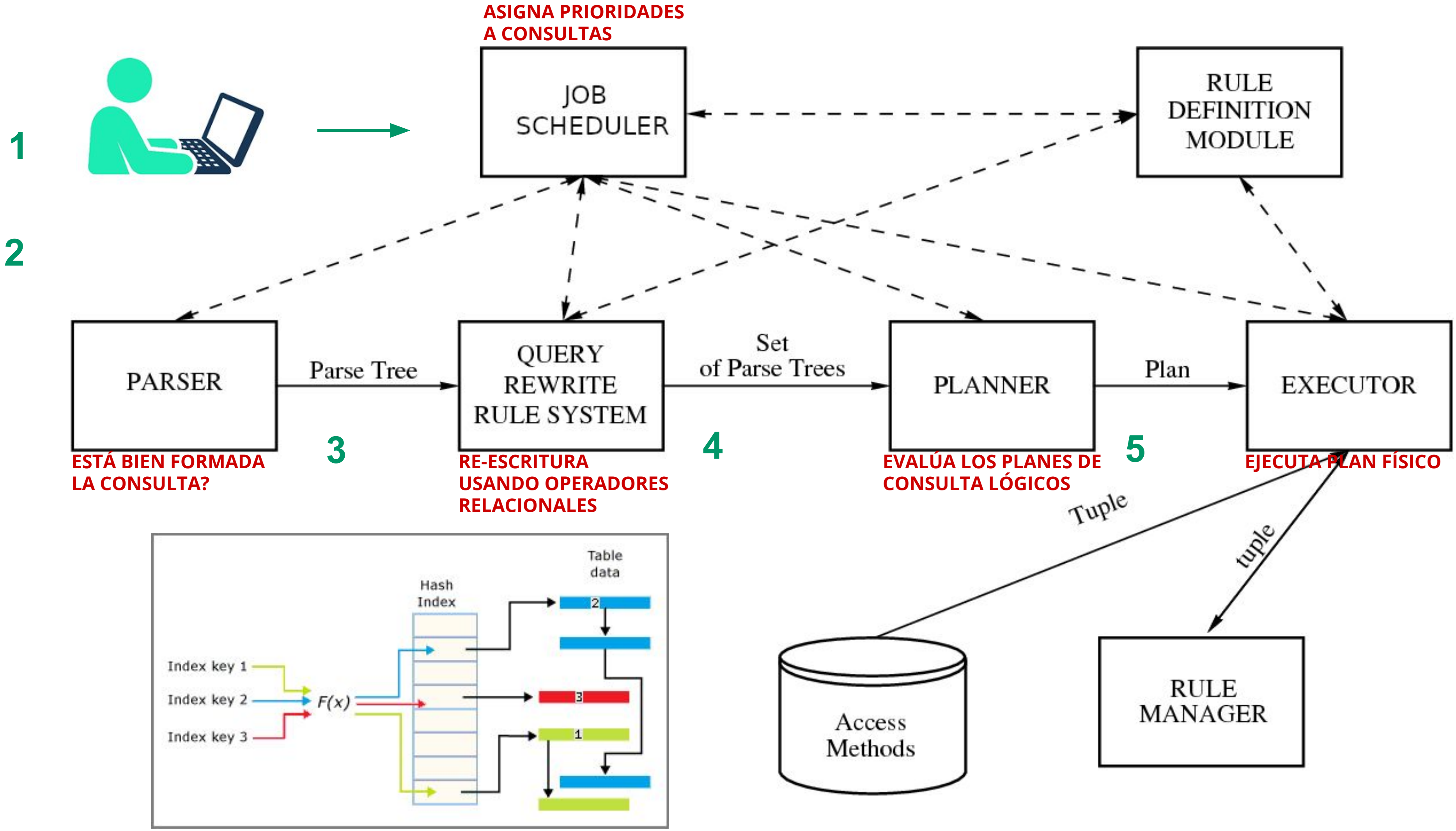
Metodología



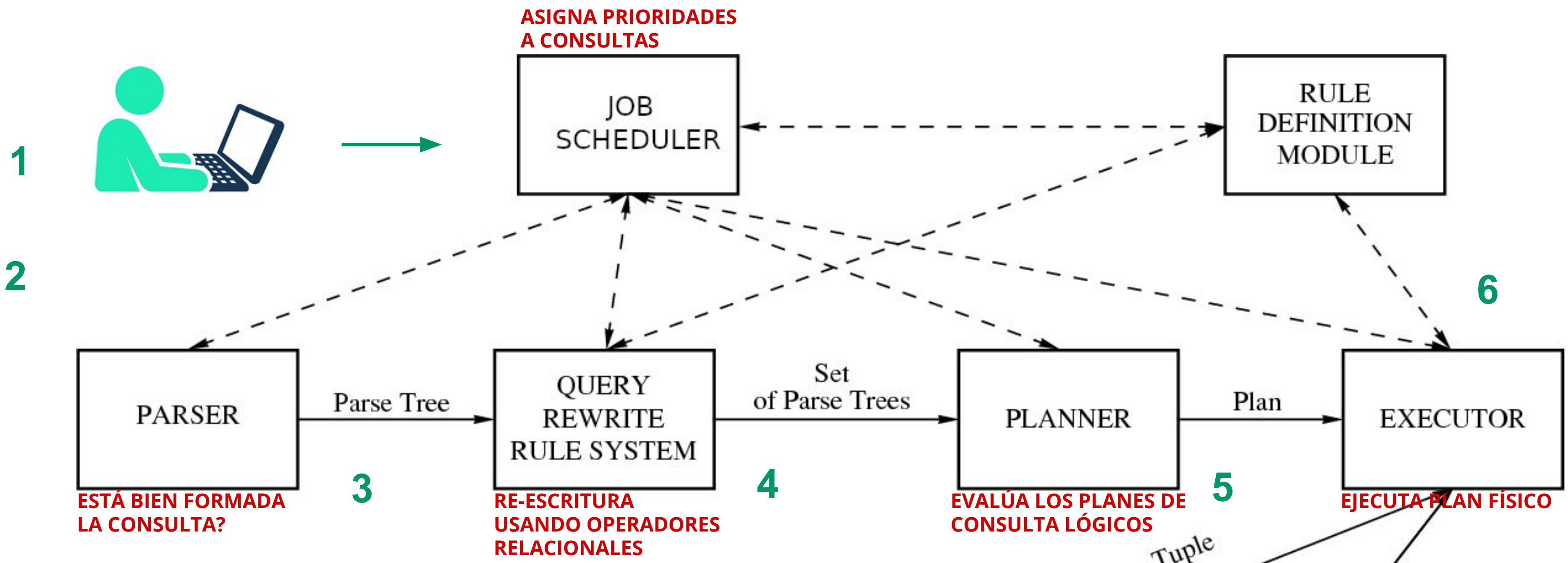
Metodología

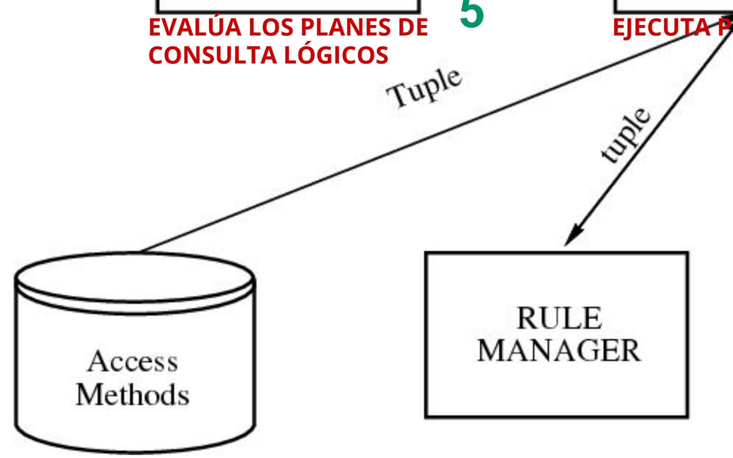


Metodología



Metodología





RESULTADO

ID	First Name	Last Name	Phone	E-mail
1	Joe	Smith	555-1234	jsmith@gmail.com
2	Jack	Williams	555-5668	jwilliams@gmail.com
3	Jill	Davis	555-5432	NULL
4	James	Miller	NULL	jmiller@yahoo.com
5	Jamie	Wilson	555-6527	NULL
6	Justin	Taylor	555-8247	jtaylor@aol.com

SQL Parser

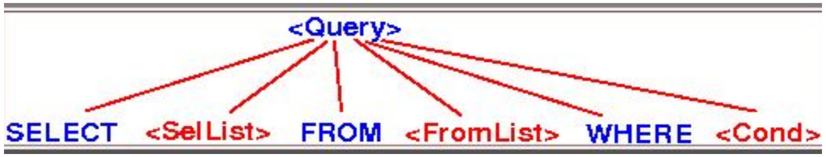
Gramática SQL

```
<Query>      ::=  SELECT <SelList>
                  FROM   <FromList>
                  WHERE  <Condition>

<SelList>    ::=  <Attribute> |
                  <Attribute> , <SelList>

<FromList>   ::=  <Relation>  |
                  <Relation>  , <FromList>

<Condition> ::=   <Condition> AND <Condition>   |
                  <Attribute> IN ( <Query> )    |
                  <Attribute> = <Attribute>     |
                  <Attribute> LIKE <Pattern>
```

SQL Parser

```
SELECT  movieTitle
FROM    StarsIn, MovieStar
WHERE   starName = name
  AND   birthdate LIKE '%1960'
```

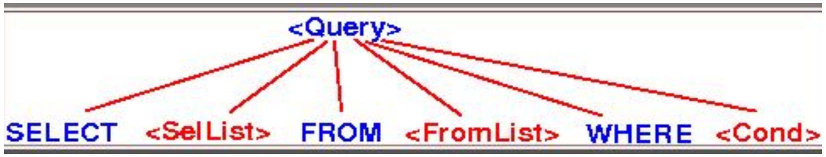
```
<Query>     ::=  SELECT <SelList>
                 FROM   <FromList>
                 WHERE  <Condition>
```



SQL Parser

```
SELECT  movieTitle
FROM    StarsIn, MovieStar
WHERE   starName = name
  AND   birthdate LIKE '%1960'
```

```
<Query>     ::=  SELECT <SelList>
                 FROM   <FromList>
                 WHERE  <Condition>
```

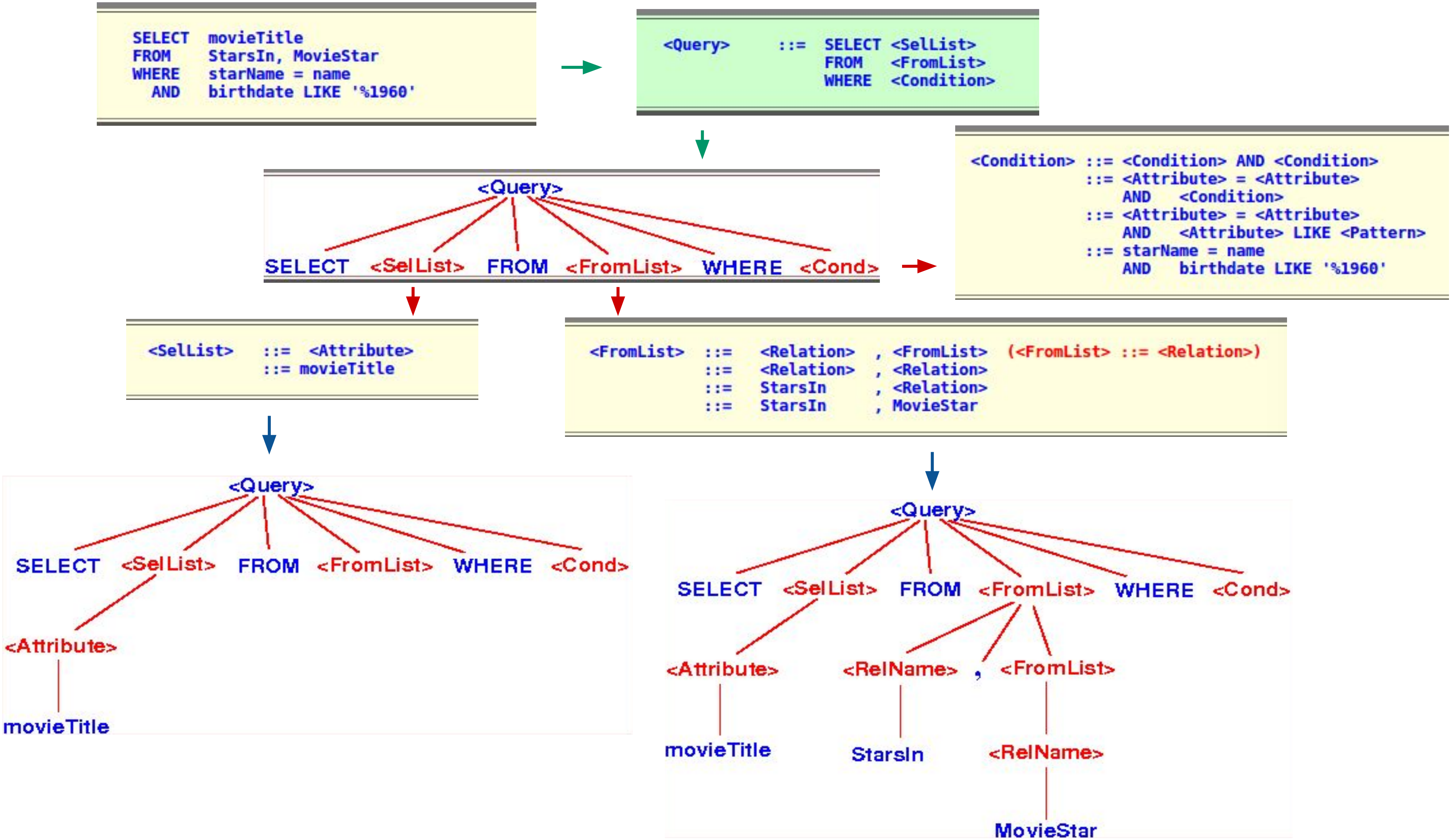



```
<Condition> ::= <Condition> AND <Condition>
            ::= <Attribute> = <Attribute>
                AND   <Condition>
            ::= <Attribute> = <Attribute>
                AND   <Attribute> LIKE <Pattern>
            ::= starName = name
                AND   birthdate LIKE '%1960'
```

```
<SelList>    ::=  <Attribute>
             ::= movieTitle
```

```
<FromList>  ::=   <Relation>   , <FromList>   (<FromList> ::= <Relation>)
            ::=   <Relation>   , <Relation>
            ::=   StarsIn      , <Relation>
            ::=   StarsIn      , MovieStar
```

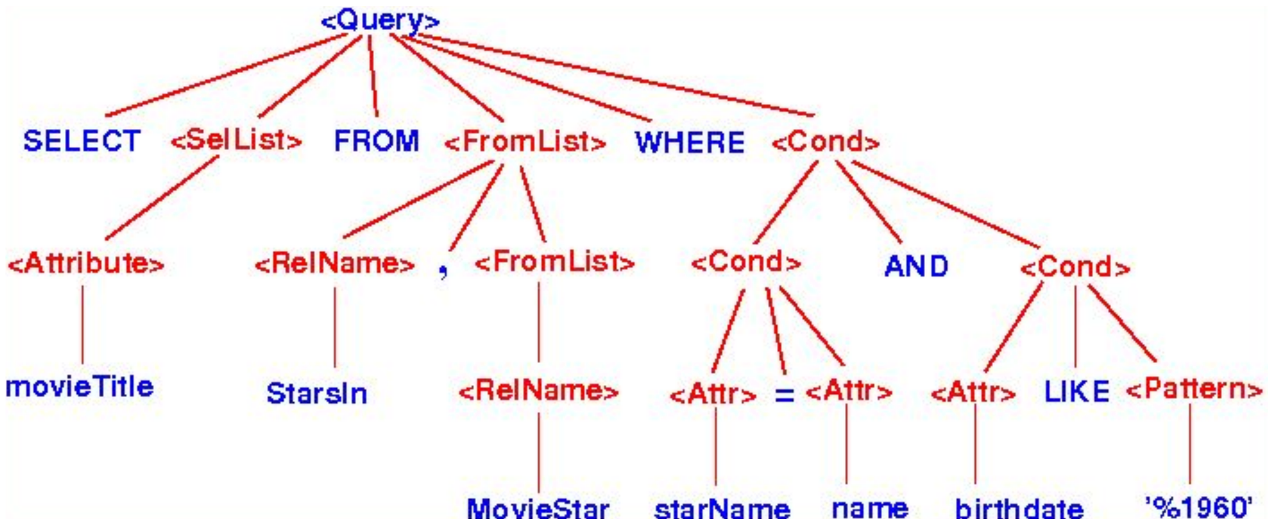
SQL Parser



SELECT movieTitle
FROM StarsIn, MovieStar
WHERE starName = name
AND birthdate LIKE '%1960'
<Query> ::= SELECT <SelList>
FROM <FromList>
WHERE <Condition>
<Query>
SELECT <SelList> FROM <FromList> WHERE <Cond>
<Condition> ::= <Condition> AND <Condition>
::= <Attribute> = <Attribute>
AND <Condition>
::= <Attribute> = <Attribute>
AND <Attribute> LIKE <Pattern>
::= starName = name
AND birthdate LIKE '%1960'
<SelList> ::= <Attribute>
::= movieTitle
<FromList> ::= <Relation> , <FromList> (<FromList> ::= <Relation>)
::= <Relation> , <Relation>
::= StarsIn , <Relation>
::= StarsIn , MovieStar
<Query>
SELECT <SelList> FROM <FromList> WHERE <Cond>
<Attribute>
movieTitle
<Query>
SELECT <SelList> FROM <FromList> WHERE <Cond>
<Attribute>
<RelName> , <FromList>
movieTitle
StarsIn
<RelName>
MovieStar

SQL Parser

```
SELECT  movieTitle
FROM    StarsIn, MovieStar
WHERE   starName = name
  AND   birthdate LIKE '%1960'
```

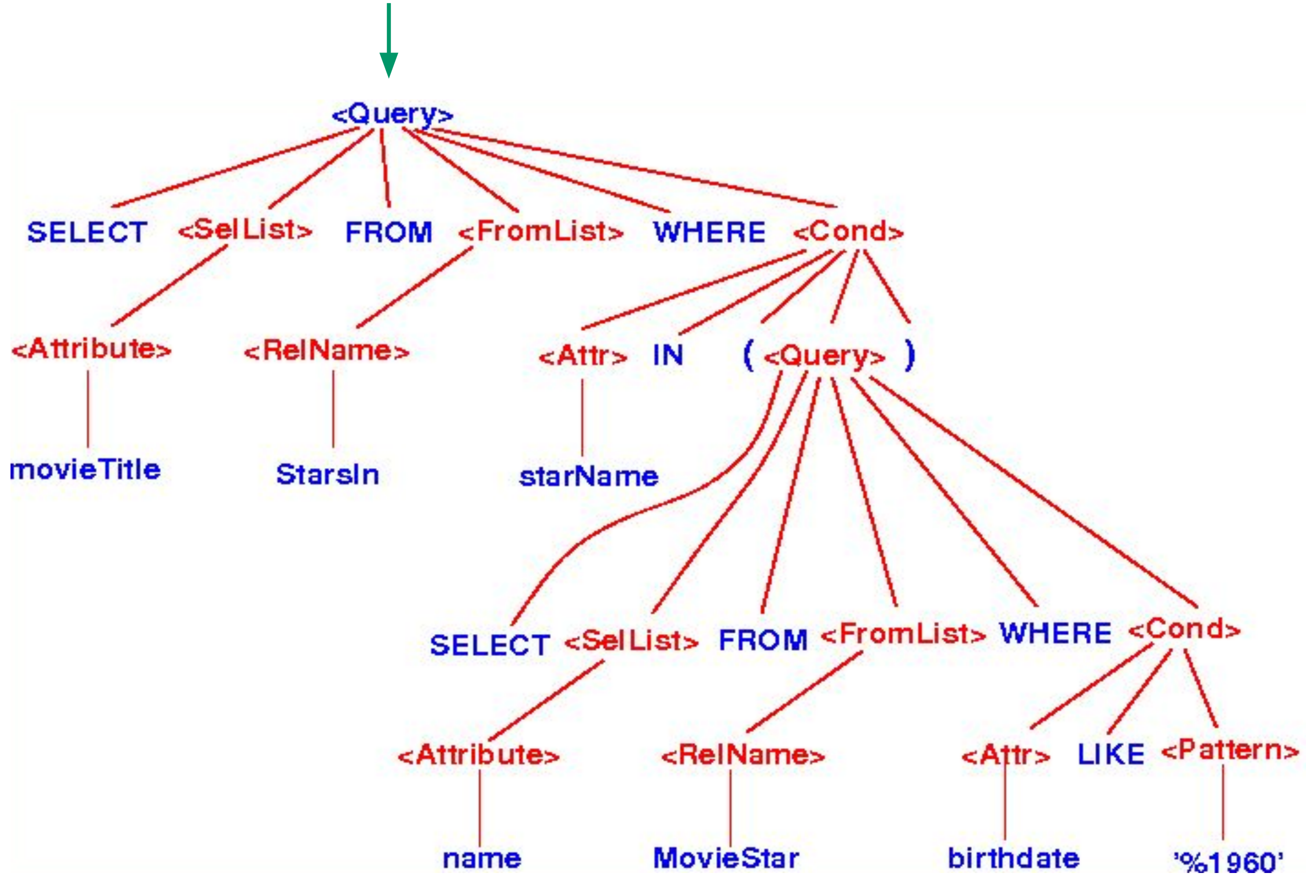


SQL Parser

```
SELECT  movieTitle
FROM    StarsIn
WHERE   starName IN ( SELECT name
                      FROM   MovieStar
                      WHERE  birthdate LIKE '%1960' )
```

SQL Parser

```
SELECT  movieTitle
FROM    StarsIn
WHERE   starName IN ( SELECT name
                      FROM   MovieStar
                      WHERE  birthdate LIKE '%1960' )
```



Query Rewrite

Los planes de ejecución lógicos se expresan usando operadores relacionales

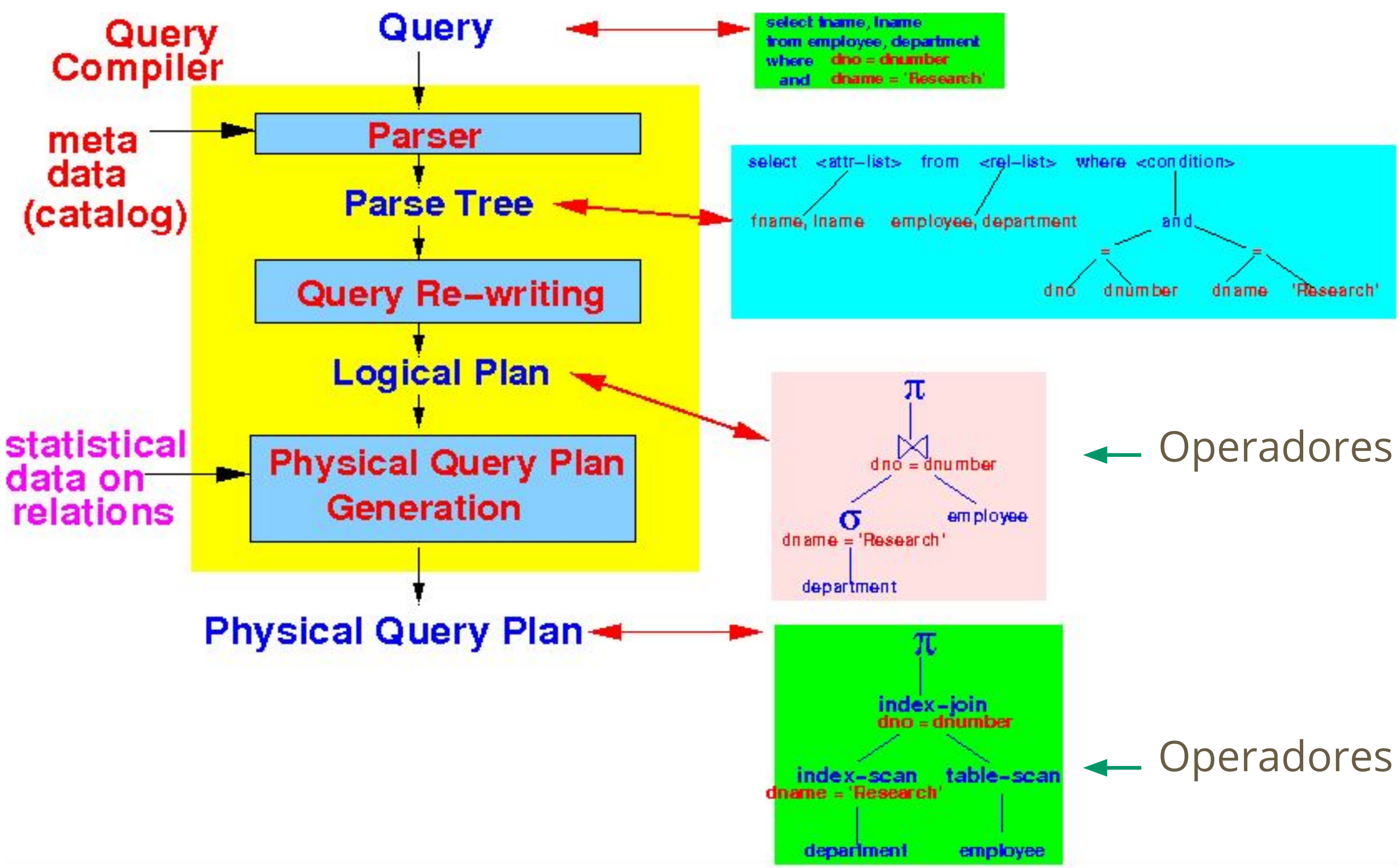
Symbol	Name of operator	Operation
table-scan(R)	Table scan	Reads blocks of the input relation R in - one block at a time.
index-scan(R)	Index scan	Reads blocks of the input relation R in using some index on the relation R
σ_{cond}	Selection	Selects the tuples that satisfies the condition cond
π_{attrs}	Projection	Extracts the attributes in attrs from a tuple Note: result of π is a bag (possible duplicates !!!)
$\bowtie_{cond}$	Theta Join	Joins 2 relations on the condition cond
$\bowtie$	Natural Join	Equi-joins 2 relations on the attributes than have the *same* name
$\times$	Product	Computes the (cartesian) product of 2 relations
δ	Duplicate Elmination	Removes the duplicate tuples from a bag of tuples (the result is a set)
γ_L	Grouping	Form gorups of tuples on common attribute values and apply the function L on each group
$\cup_S$	Set Union	Computes the *set* union of 2 sets (*no* duplicates)
$\cup_B$	Bag Union	Computes the *bag* union of 2 sets (*allows* duplicates)
$\cap_S$	Set Intersection	Computes the *set* intersection of 2 sets (*no* duplicates)
$\cap_B$	Bag Intersection	Computes the *bag* intersection of 2 sets (*allows* duplicates)
$-_S$	Set Difference	Computes the *set* difference of 2 sets (*no* duplicates)
$-_B$	Bag Difference	Computes the *bag* difference of 2 sets (*allows* duplicates)

Query Rewrite

Los planes de ejecución lógicos se expresan usando operadores relacionales

	Symbol	Name of operator	Operation
	table-scan(R)	**Table scan**	**Reads blocks** of the **input relation** R in - one block at a time.
	index-scan(R)	**Index scan**	**Reads blocks** of the **input relation** R in using some **index** on the relation **R**
	σ_{cond}	**Selection**	**Selects** the **tuples** that satisfies the condition **cond**
	π_{attrs}	**Projection**	**Extracts** the **attributes** in **attrs** from a tuple **Note:** result of **π** is a **bag** (possible **duplicates** !!!)
	$\bowtie_{cond}$	**Theta Join**	**Joins** 2 relations on the **condition cond**
	$\bowtie$	**Natural Join**	**Equi-joins** 2 relations on the **attributes** than have the ***same* name**
CROSS JOIN	$\times$	**Product**	Computes the **(cartesian) product** of 2 relations
	δ	**Duplicate Elmination**	**Removes** the **duplicate tuples** from a **bag** of **tuples** (the result is a **set**)
GROUP BY	γ_L	**Grouping**	**Form gorups** of tuples on common **attribute values** and **apply** the **function L** on **each group**
	$\cup_S$	**Set Union**	Computes the ***set* union** of 2 sets (***no* duplicates**)
	$\cup_B$	**Bag Union**	Computes the ***bag* union** of 2 sets (***allows* duplicates**)
	$\cap_S$	**Set Intersection**	Computes the ***set* intersection** of 2 sets (***no* duplicates**)
	$\cap_B$	**Bag Intersection**	Computes the ***bag* intersection** of 2 sets (***allows* duplicates**)
	$-_S$	**Set Difference**	Computes the ***set* difference** of 2 sets (***no* duplicates**)
	$-_B$	**Bag Difference**	Computes the ***bag* difference** of 2 sets (***allows* duplicates**)

Query Rewrite

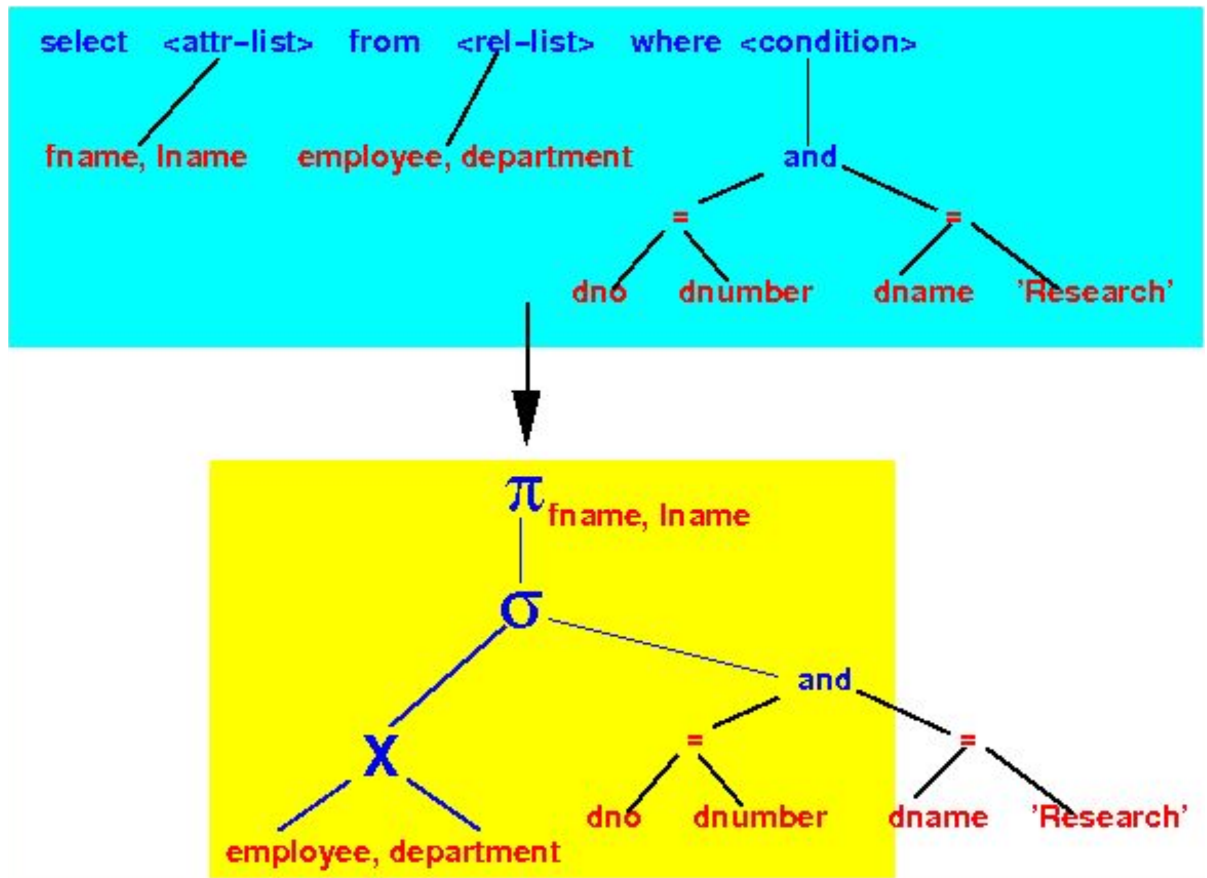


Operadores relacionales

Operadores físicos

Query Rewrite

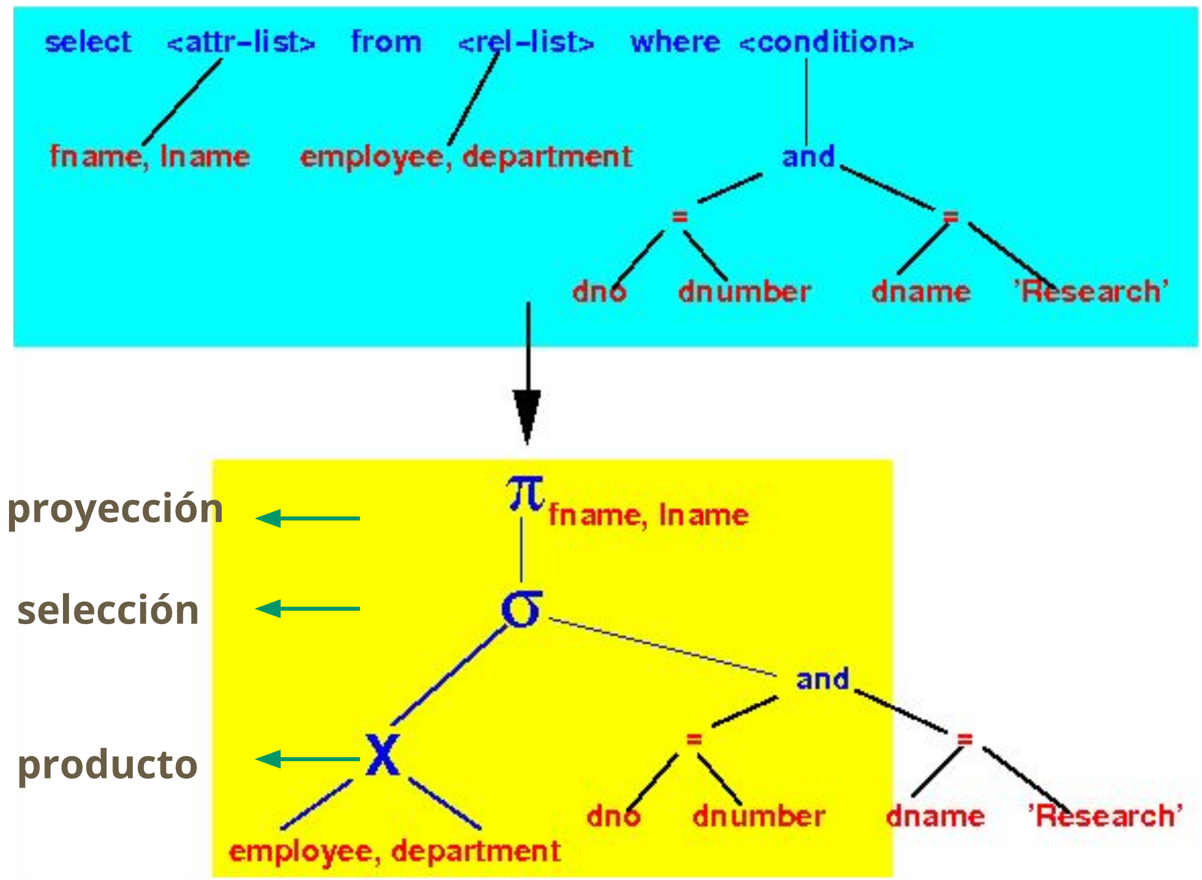
Parse-tree → plan lógico de la consulta



Partir desde el nivel más bajo hacia los niveles altos

Query Rewrite

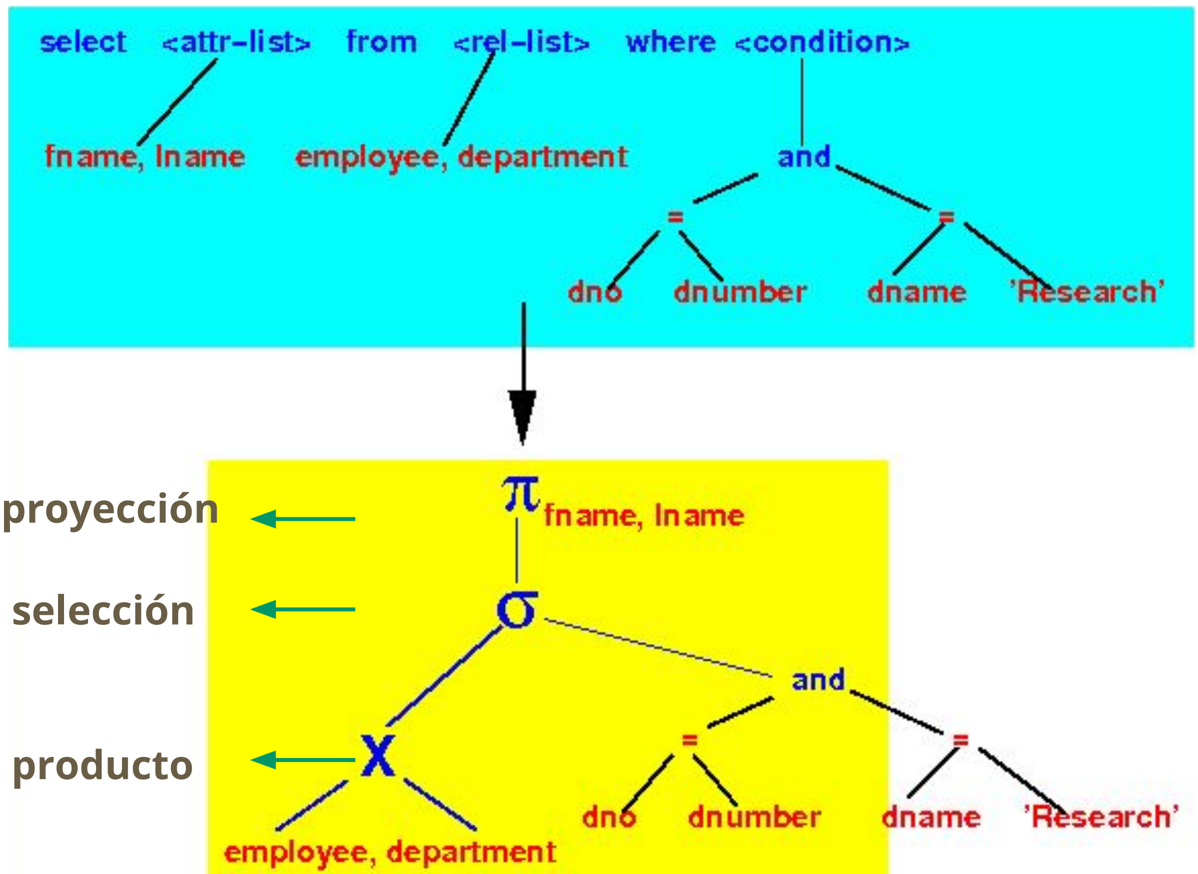
Parse-tree → plan lógico de la consulta



Partir desde el nivel más bajo hacia los niveles altos

Query Rewrite

Parse-tree → plan lógico de la consulta



SELECT ← proyección ←

WHERE ← selección ←

FROM ← producto ←

Partir desde el nivel más bajo hacia los niveles altos

Query Planner

Plan lógico de la consulta → plan optimizado

Criterios de optimización:

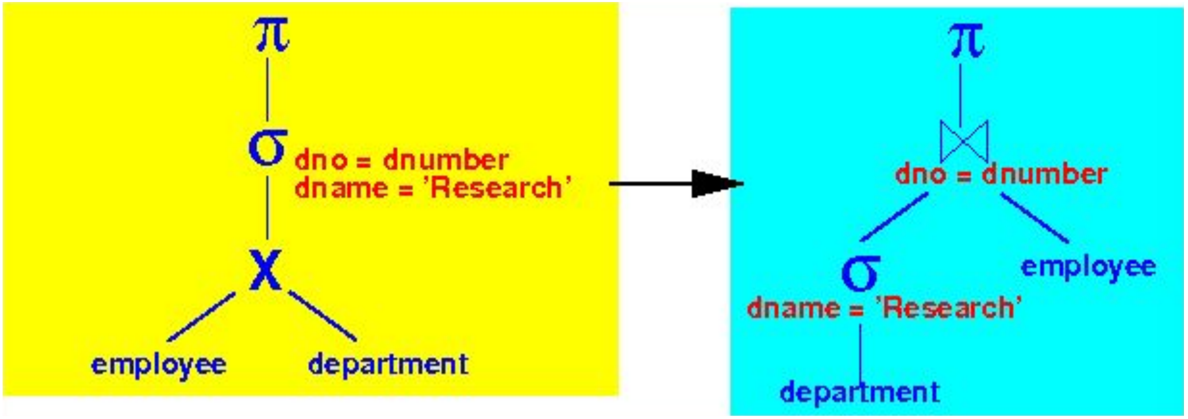
1. ***Selection pushdown*****:** empujar la selección hacia las hojas
2. ***Join reordering*****:** permutar las relaciones en un join

Query Planner

Plan lógico de la consulta → plan optimizado

Criterios de optimización:

1. ***Selection pushdown***: empujar la selección hacia las hojas
2. ***Join reordering***: permutar las relaciones en un join

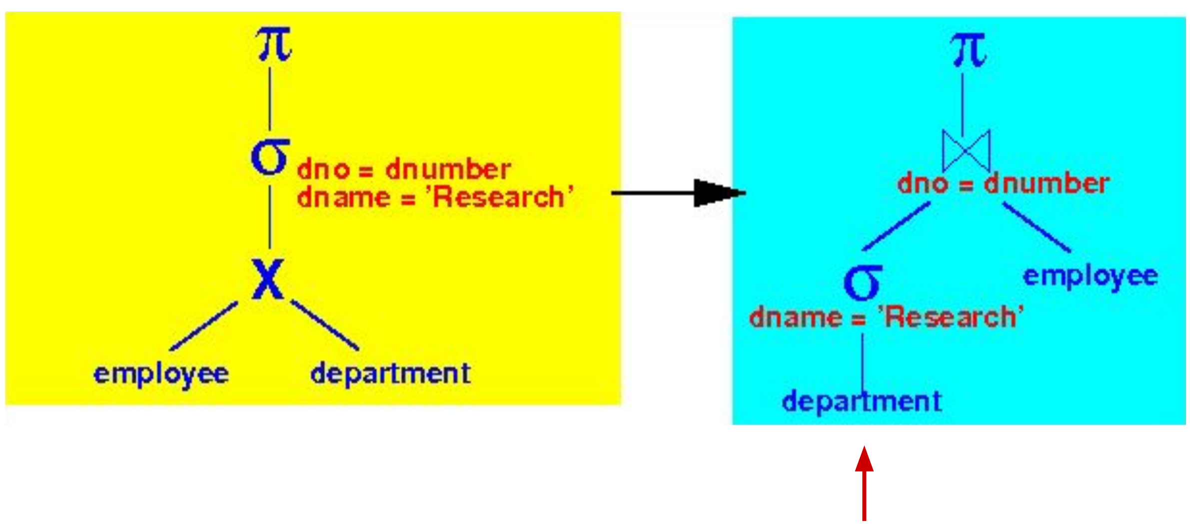


Query Planner

Plan lógico de la consulta → plan optimizado

Criterios de optimización:

1. ***Selection pushdown***: empujar la selección hacia las hojas
2. ***Join reordering***: permutar las relaciones en un join



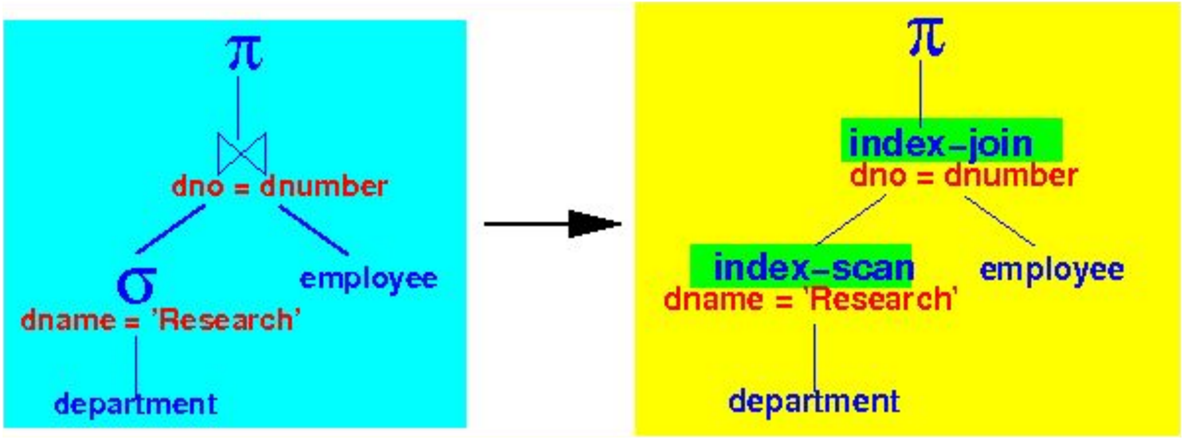
Aquí se hizo selection pushdown

Executor

Selecciona el mejor algoritmo para ejecutar el plan lógico optimizado

El mejor algoritmo depende de:

- Índices disponibles
- Memoria disponible para procesar la consulta

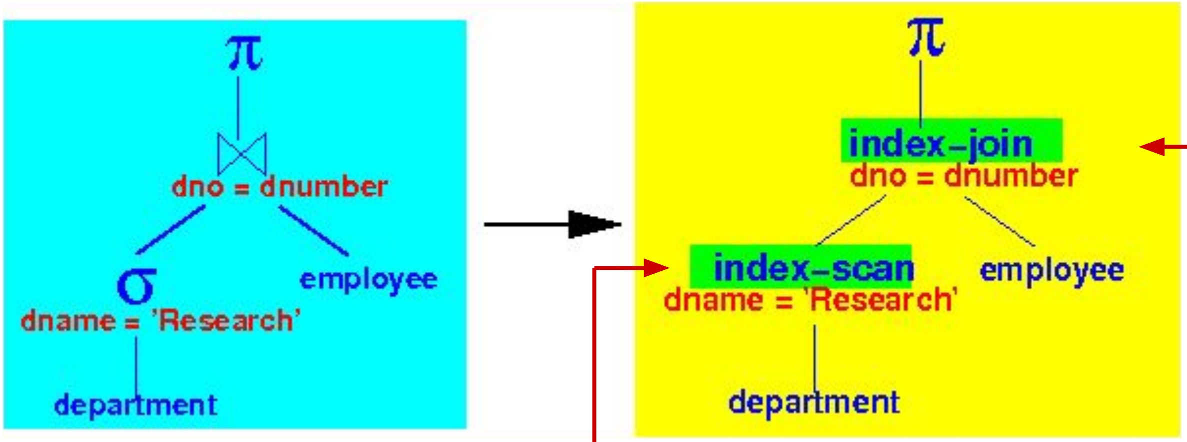


Executor

Selecciona el mejor algoritmo para ejecutar el plan lógico optimizado

El mejor algoritmo depende de:

- Índices disponibles
- Memoria disponible para procesar la consulta



Escogió usar un índice para el join

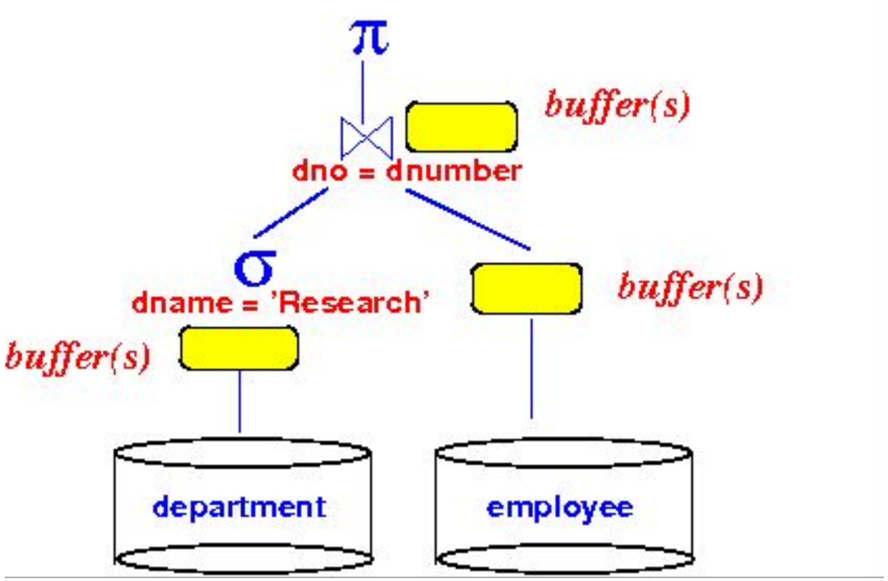
Escogió usar un índice para la selección

Costo de una consulta

¿Qué variables afectan al costo de ejecución?

Costo de una consulta

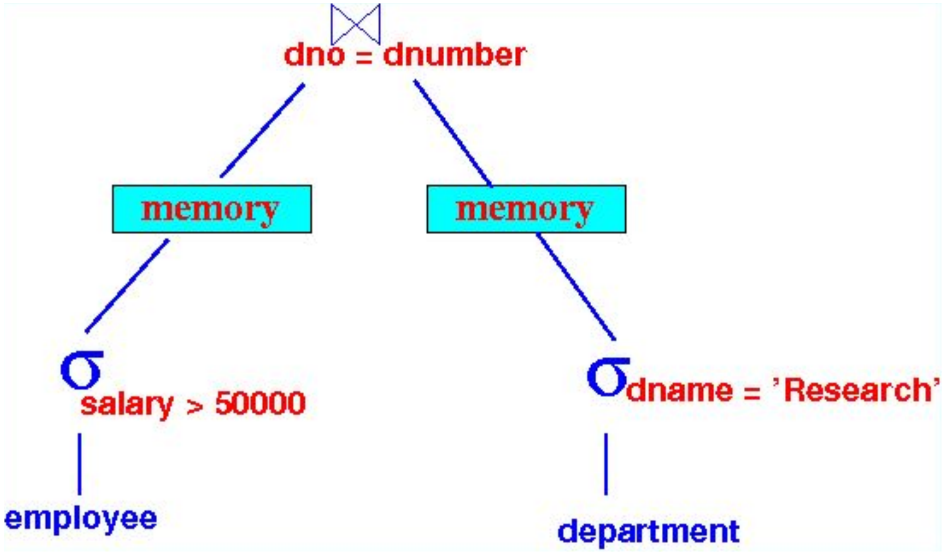
Una consulta requiere de *buffers* de memoria para poder computarse.



- La ejecución del plan físico requiere de varios *buffers* para alojar bloques de discos
- El costo de la consulta está determinado por el número de bloques de disco accesados

Costo de una consulta

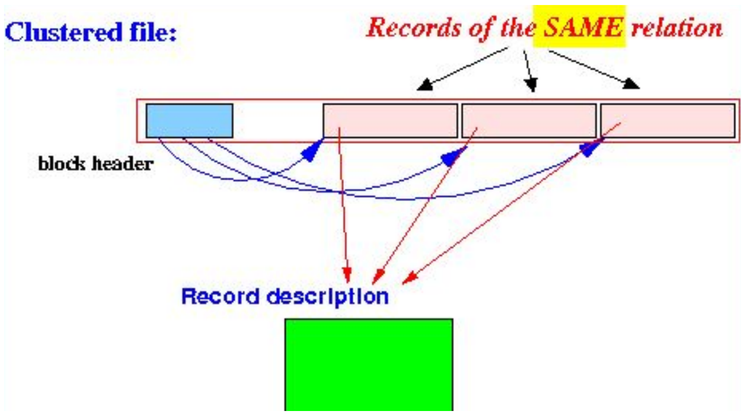
Asumimos que los resultados intermedios pueden serializarse en un pipeline, es decir, el operador que recibe los datos los lee desde un buffer:

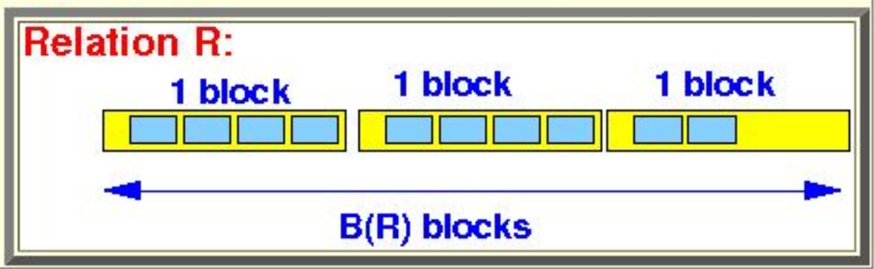


- Los costos pueden contabilizarse como el número de buffers necesarios para transferir los resultados en el plan de la consulta
- Contabilizamos los costos en *buffers* de entrada y en memoria

Costo de operaciones básicas

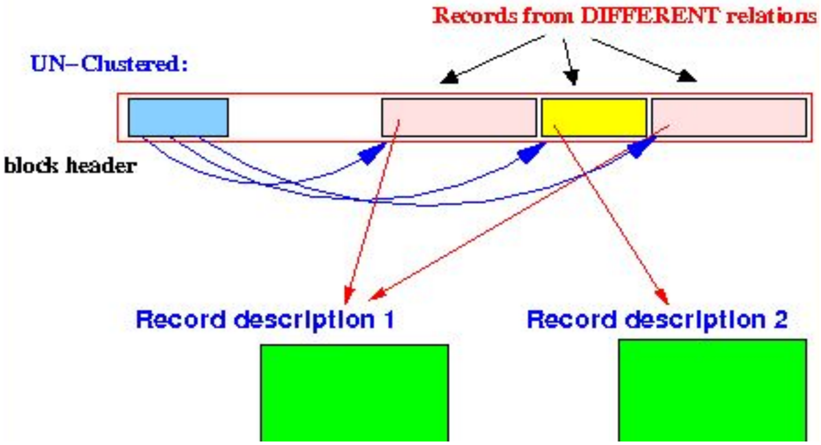
Archivos clusterizados



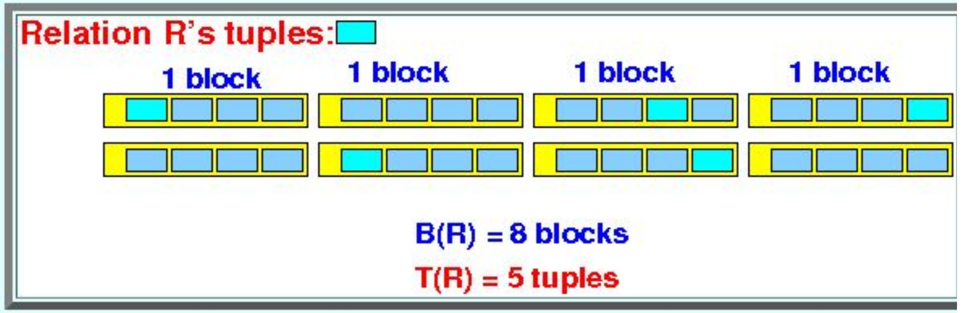
Operator	I/O cost	Explanation
Table-Scan	B(R)	Read all blocks from the relation: B(R) blocks. 
Index-Scan	≤ B(R)	Depends on number of values in the index is scanned Single value: I/O cost = O(1) (a few blocks) << B(R)

Costo de operaciones básicas

Archivos no clusterizados



Record header contains information of the relation to which the record belong

Operator	I/O cost	Explanation
Table-Scan	~ T(R)	Assuming that the **tuples** are **sparsely** stored in the **file**:  We will read **1 disk block** per **tuple**. **Total disk blocks** read = **T(R)**
Index-Scan	≤ T(R)	Depends on **number** of **values** in the **index** is **scanned** **Single value**: **I/O cost = O(1) (a few blocks) << B(R) << T(R)**

Costo de operaciones básicas

Operador de selección

σ_{cond} (R)

Ejemplo:

```
R = { ('John', 30000), ('Jane', 50000) }
```

$\sigma_{salary > 40000}$ (employee) = { ('Jane', 50000) }

Algoritmo de una pasada (*one-pass*)

```
while ( R has more data blocks )
{
   read data blocks to fill available buffers;
   check condition for each tuple in the buffers;
   Move qualifying tuples to output
}
```

Costo de operaciones básicas

Operador de selección

σ_{cond} (R)

Ejemplo:

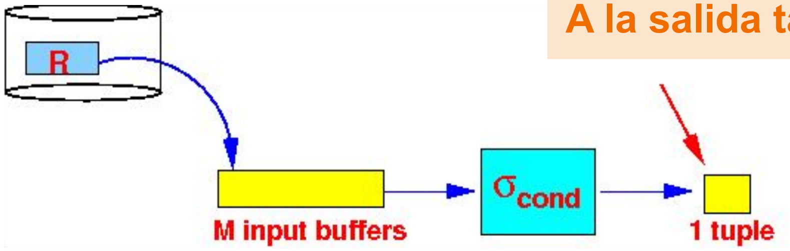
```
R = { ('John', 30000), ('Jane', 50000) }

σsalary > 40000 ( employee ) = { ('Jane', 50000) }
```

Algoritmo de una pasada (*one-pass*)

```
while ( R has more data blocks )
{
   read data blocks to fill available buffers;

   check condition for each tuple in the buffers;

   Move qualifying tuples to output
}
```

Uso del *buffer*

A la salida también se usa un *buffer*



Costos

- **B(R)** --- ***if*** the **relation R** is ***clustered***
- **T(R)** --- ***if*** the **relation R** is ***UNclustered***

Costo de operaciones básicas

Operador de proyección

π_{attrs} (R)

Ejemplo:

R = { ('John', 30000), ('Jane', 50000) }

π_{fname} (employee) = { ('John'), ('Jane') }

Algoritmo de una pasada (*one-pass*)

```
while ( R has more data blocks )
{
   read data blocks to fill available buffers;

   Extract the projection attrs for each tuple in buffers;

   Move projected tuples to output
}
```

Costo de operaciones básicas

Operador de proyección

π_{attrs} (R)

Ejemplo:

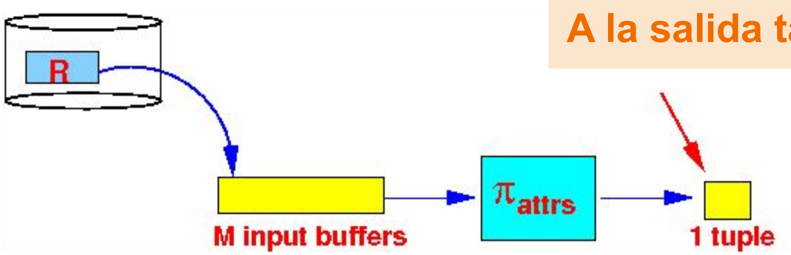
```
R = { ('John', 30000), ('Jane', 50000) }

πfname ( employee ) = { ('John'), ('Jane') }
```

Algoritmo de una pasada (*one-pass*)

```
while ( R has more data blocks )
{
   read data blocks to fill available buffers;

   Extract the projection attrs for each tuple in buffers;

   Move projected tuples to output
}
```

Uso del *buffer*

A la salida también se usa un *buffer*



Costos

- **B(R)** --- ***if*** the **relation R** is ***clustered***
- **T(R)** --- ***if*** the **relation R** is ***UNclustered***

Costo de operaciones básicas

Operador de agrupamiento:

$$\gamma_L \ (\ R \ )$$

Ejemplo:

```
R = (x,y)
R = { (a,2), (a,3), (b,1), (b,3) }
```

$$\gamma_{avg(y)} \ (\ R \ ) = \{ \ (a,\ 2.5),\ (b,\ 2.0) \ \}$$

Costo de operaciones básicas

Algoritmo de una pasada (*one-pass*):

```
initialize a search structure H on grouping attributes of γ;

/* =========================================================
   Process the statistics for each group
   ========================================================= */
while ( R has more data blocks )
{
   read 1 data block in buffer b;

   for ( each tuple t ∈ b )
   {
      /* =====================================================
         We need a search structure H to implement the test
         t ∈ H  efficiently !!!

         We can use hash table or some bin. search tree
         ====================================================== */
      if ( t ∈ H )
      {
         Update the statistics for group(t);
      }
      else
      {
         insert t in H;

         Initialize the statistics for group(t);
      }
   }
}

/* =========================================================
   Now we can output the aggregate function for each group
   ========================================================= */
for ( each group ∈ H )
{
   Output group search key + statistics;
}
```

- 1 *buffer* para leer las tuplas
- M - 1 *buffers* para almacenar el índice

Costo de operaciones básicas

Algoritmo de una pasada (*one-pass*):

```
initialize a search structure H on grouping attributes of γ;

/* =========================================================
   Process the statistics for each group
   ========================================================= */
while ( R has more data blocks )
{
   read 1 data block in buffer b;

   for ( each tuple t ∈ b )
   {
      /* ====================================================
         We need a search structure H to implement the test
         t ∈ H  efficiently !!!

         We can use hash table or some bin. search tree
         ==================================================== */
      if ( t ∈ H )
      {
         Update the statistics for group(t);
      }
      else
      {
         insert t in H;

         Initialize the statistics for group(t);
      }
   }
}

/* =========================================================
   Now we can output the aggregate function for each group
   ========================================================= */
for ( each group ∈ H )
{
   Output group search key + statistics;
}
```

- 1 *buffer* para leer las tuplas
- M - 1 *buffers* para almacenar el índice

Costo:

- **B(R)** --- ***if*** the **relation R** is ***clustered***

- **T(R)** --- ***if*** the **relation R** is ***UNclustered***

Memoria:

$$M = O(\,V(R, [A_1, A_2, .., A_k])\,)$$

donde:

- $V(R, [A_1, A_2, .., A_k])$ = #valores que $[A_1,\ldots,A_k]$ puede tomar

Consultas?

Recuerden!

- **Marcelo Mendoza: mmendoza@inf.utfsm.cl**
- **Margarita Bugueño: margarita.bugueno@usm.cl**

Bases de Datos

Ejecución de consultas