

Algoritmos y Estructuras de Datos II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 2

“Algebra Relacional”

Integrante	LU	Correo electrónico
Luis Ricardo Bustamante	43/18	luisbustamante097@gmail.com
Leandro Emanuel Rodriguez	521/17	leandro21890000@gmail.com
Ramiro Augusto Ciruzzi	228/17	ramiro.ciruzzi@gmail.com
David Alejandro Venegas	783/18	venegasr.david@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Modulo BaseDeDatos

El modulo BaseDeDatos provee operaciones básicas a realizarse sobre una base de datos. Permite crear una base de datos, agregar o eliminar una tabla, agregar o eliminar registros a una tabla, y también permite ejecutar consultas para obtener registros de una tabla.

Interfaz

se explica con: BASEDEDATOS

usa: REGISTRO, TABLA, CONSULTA, BOOL, NAT, CONJUNTOLINEAL

género: base_de_datos

NUEVABD() $\rightarrow res : \text{BaseDeDatos}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res = \text{nuevaBD}\}$

Complejidad: $O(1)$

Descripción: Crea una nueva base de datos.

AGREGARTABLA(in/out b: BaseDeDatos, in t: Tabla, in nt: NombreTabla)

Pre $\equiv \{b = b0\}$

Post $\equiv \{b = \text{AgregarTabla}(b0, t, nt)\}$

Complejidad: $O(nt + \text{copy}(t))$

Descripción: Inserta una tabla a una base de datos. En caso de que la tabla ya existe en la base, la sobrescribe.

BORRARTABLA(in/out b: BaseDeDatos, in nt: NombreTabla)

Pre $\equiv \{b = b0\}$

Post $\equiv \{(nt \notin \text{nombreTablas}(b0) \wedge b=b0) \vee (nt \in \text{nombreTablas}(b0) \wedge b=\text{sacarTabla}(b0, nt))\}$

Complejidad: $O(|nt|)$

Descripción: Elimina una tabla de una base de datos. En caso de que la tabla no exista en la base, la operacion queda sin efecto.

AGREGARREGISTRO(in/out b: BaseDeDatos, in nt: NombreTabla, in r: Registro)

Pre $\equiv \{b = b0 \wedge (\text{campos}(\text{tabla}(b, nt)) = \text{campos}(r))\}$

Post $\equiv \{b = \text{AgregarRegistro}(b0, \text{tabla}(b0, nt), r)\}$

Complejidad: $O(|nt| + \text{copy}(r))$

Descripción: Inserta una registro en una tabla en una base de datos dándole valor a todos (y solamente a) los campos de la tabla. En caso de que el registro ya existe en la base, lo sobrescribe.

BORRARREGISTRO(in/out b: BaseDeDatos, in nt: NombreTabla, in v: Valor)

Pre $\equiv \{b = b0 \wedge nt \in \text{nombreTablas}(b0)\}$

Post $\equiv \{b = \text{sacarRegistro}(b0, \text{tabla}(b0, nt), v)\}$

Complejidad: $O(|nt| + k), k = \#registros$

Descripción: Elimina un registro de una tabla en una base de datos. En caso de que el registro no pertenezca a la tabla, esta queda sin efecto

NOMBRETABLAS(in b: BaseDeDatos) $\rightarrow res : \text{Conj}(\text{nombreTabla})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res = \text{nombreTablas}(b)\}$

Complejidad: $O(|n| * k)$

Descripción: Devuelve el conjunto de nombres de todas las tablas de la base de datos.

TABLA(in b: BaseDeDatos, in nt: NombreTabla) $\rightarrow res : \text{tabla}$

Pre $\equiv \{nt \in \text{nombreTablas}(b)\}$

Post $\equiv \{res = \text{tabla}(b, nt)\}$

Complejidad: $O(|nt|)$

Descripción: Dados un nombre de tabla y una base de datos, devuelve la tabla correspondiente al nombre en dicha base.

EJECUTARCONSULTA(in b: BaseDeDatos, in q: Consulta) $\rightarrow res : \text{conj}(\text{Registro})$

Pre $\equiv \{$

$(\text{TipoConsulta}(q) \in \{FROM\} \wedge \text{ExisteTablaEnBd}(b, \text{NombreTabla}(q))) \wedge_L$

$(\text{TipoConsulta}(q) \in \{SELECT\} \wedge \text{RegistrosTienenCampo}(\text{ejecutarConsulta}(b, \text{subconsulta1}(q)), \text{campo1}(q))) \wedge_L$

$(\text{TipoConsulta}(q) \in \{MATCH\} \wedge \text{RegistrosTienenCampo}(\text{ejecutarConsulta}(b, \text{subconsulta1}(q)), \text{campo1}(q)) \wedge \text{RegistrosTienenCampo}(\text{ejecutarConsulta}(b, \text{subconsulta1}(q)), \text{campo2}(q))) \wedge_L$

$(\text{TipoConsulta}(q) \in \{RENAME\} \wedge \text{RegistrosTienenCampo}(\text{ejecutarConsulta}(b, \text{subconsulta1}(q)), \text{campo1}(q)) \wedge$

$\neg \text{RegistrosTienenCampo}(\text{ejecutarConsulta}(b, \text{subconsulta1}(q)), \text{campo2}(q))) \wedge_L$
 $(\text{TipoConsulta}(q) \in \{PRODUCT\} \wedge \neg \text{HayCamposEnComun}(\text{ejecutarConsulta}(b, \text{subconsulta1}(q)), \text{ejecutarConsulta}(b, \text{subconsulta2}(q)))) \wedge_L$
 $\text{TipoConsulta}(q) \in \{PROJ, INTER, UNION\}$
 $\}$

Post $\equiv \{ \text{res} = \text{ejecutarConsulta}(b, q) \}$

Descripción: Dados una consulta c y una base de datos b , devuelve los registros resultantes luego de haber ejecutado la consulta c en la base b , añadiendo las siguientes particularidades dependiendo del $\text{TipoConsulta}(c)$:

Si $\text{TipoConsulta}(c) \in \{FROM\}$, devuelve todos los registros contenidos en la tabla correspondiente de b cuyo nombre es $\text{nombre_tabla}(c)$.

Si $\text{TipoConsulta}(c) \in \{SELECT\}$, devuelve todos los registros resultantes al ejecutar la consulta $\text{subconsulta1}(c)$ en b , tales que el campo $\text{campo1}(c)$ tiene valor $\text{valor}(c)$.

Si $\text{TipoConsulta}(c) \in \{MATCH\}$, devuelve todos los registros resultantes al ejecutar la consulta $\text{subconsulta1}(c)$ en b , tales que los campos $\text{campo1}(c)$ y $\text{campo2}(c)$ tienen el mismo valor.

Si $\text{TipoConsulta}(c) \in \{PROJ\}$, devuelve todos los registros resultantes al ejecutar la consulta $\text{subconsulta1}(c)$ en b , pero que incluyen solamente los campos del conjunto $\text{conj_campos}(c)$.

Si $\text{TipoConsulta}(c) \in \{RENAME\}$, devuelve todos los registros resultantes al ejecutar la consulta $\text{subconsulta1}(c)$ en b , tales que el campo $\text{campo1}(c)$ será reemplazado por el campo $\text{campo2}(c)$.

Si $\text{TipoConsulta}(c) \in \{INTER\}$, devuelve la intersección entre los registros resultantes de la ejecución de $\text{subconsulta1}(c)$ y $\text{subconsulta2}(c)$

Si $\text{TipoConsulta}(c) \in \{UNION\}$, devuelve la unión entre los registros resultantes de la ejecución de $\text{subconsulta1}(c)$ y $\text{subconsulta2}(c)$

Si $\text{TipoConsulta}(c) \in \{PRODUCT\}$, devuelve el producto cartesiano entre los registros resultantes de la ejecución de $\text{subconsulta1}(c)$ y $\text{subconsulta2}(c)$.

$\text{RegistrosTienenCampo} : \text{conj}(\text{Registro}) \times \text{NombreCampo} \rightarrow \text{bool}$

$\text{RegistrosTienenCampo}(c, nc) \equiv (\forall r: \text{Registro})(r \in c \Rightarrow_L nc \in \text{campos}(r))$

$\text{HayCamposEnComun} : \text{conj}(\text{Registro}) \times \text{conj}(\text{Registro}) \rightarrow \text{bool}$

$\text{HayCamposEnComun}(cr1, cr2) \equiv (\exists r1: \text{Registro})(\exists r2: \text{Registro})(r1 \in cr1 \wedge r2 \in cr2 \wedge (\exists nc: \text{nombreCampo})(nc \in \text{campos}(r1) \wedge (nc \in \text{campos}(r2))))$

Representación

La base de datos se representa con un `diccString`, en donde dado un nombre de tabla nt , podemos obtener la tabla correspondiente de la bd asociada con nt .

BaseDeDatos se representa con estr

donde estr es `diccString(nt:string, t:tabla)`

Rep True, porque diccString mantiene consistente toda la informacion interna.

Abs Para todo e : estr , se cumple que: $\text{abs}(e) = \text{base}/\text{dicString}::\text{claves}(e) = \text{nombreTablas}(\text{base})$ y para todo nt que pertenece a $\text{claves}(\text{base})$ implica que $\text{diccString}::\text{obtener}(e, nt) = \text{tabla}(\text{base}, nt)$

Algoritmos

iNuevaBd() $\rightarrow \text{res} : \text{base_de_datos}$

$\text{res} \leftarrow \text{diccString}::\text{vacio}()$

$//O(1)$

agregarTabla(in/out bd: estr, in t: tabla, in nt: string) $\rightarrow \text{res} : \text{base_de_datos}$

$\text{diccString}::\text{definir}(\text{bd}, nt, t)$

$//O(|nt| + \text{copy}(t))$

```

borrarTabla(in/out bd: estr, in nt: string)
    if (diccString::definido?(bd, nt) then                                     //  $O(|nt|)$ 
        diccString::borrar(bd, nt)                                           //  $O(|nt|)$ 
    end if

```

```

agregarRegistro(in/out bd: estr, in nt: string, in r: registro)
    tablaAModificar  $\leftarrow$  diccString::obtener(bd, nt)                               //  $O(|nt|)$ 
    tabla::insertar(tablaAModificar, r)                                           //  $O(\text{copy}(r))$ 

```

```

borrarRegistro(in/out bd: estr, in nt: string, in v: string)
    tablaAModificar  $\leftarrow$  diccString::obtener(bd, nt)                               //  $O(|nt|)$ 
    tabla::borrar(tablaAModificar, v)                                           //  $O(n), n = \#registros$ 

```

```

numeroTablas(in/out bd: estr)  $\rightarrow$  res: conj(string)
    diccString::claves(bd)                                                         //  $O(n * |k|), n = \#registros$ 

```

```

tabla(in bd: estr, in nt: string)  $\rightarrow$  res: conj(string)
    res  $\leftarrow$  diccString::obtener(bd, nt)                                     //  $O(|nt|)$ 

```

```

iSelectFrom(in bd: estr, in q: consulta)  $\rightarrow$  res: conj(registro)
    conjRes  $\leftarrow$  vacio()                                                         //  $O(1)$ 
    if c == tabla::clave(tabla) then                                             //  $O(|c|)$ 
        reg  $\leftarrow$  *camposXClave(tabla, v)                                       //  $O(|t| + |v|)$ 
        if !conjLineal::esVacio?(registro::campos(reg)) then                 //  $O(1)$ 
            agregarRapido(conjRes, reg)                                           //  $O(|v| + |c|)$ 
        end if
    else
        nroCampo  $\leftarrow$  tabla::obtenerNroCampo(tabla, c)                       //  $O(|c|)$ 
        listaDeItReg  $\leftarrow$  vacia()                                             //  $O(1)$ 
        it  $\leftarrow$  crearItConj(tabla::registros(tabla))                         //  $O(1)$ 
        while haySiguiente(it) do                                               //  $O(n)$ 
            if registro::valorEnRegistro(siguiente(it), nroCampo) == v then //  $O(|v|)$ 
                agregar(listaDeItReg, it)                                       //  $O(1)$ 
            end if
            avanzar(itReg)                                                         //  $O(1)$ 
        end while
        itLista  $\leftarrow$  crearItLista(listaDeItReg)                             //  $O(1)$ 
        while haySiguiente(itLista) do                                           //  $O(k) * O(|v| + |c|)$ 
            agregarRapido(conjRes, siguiente(siguiente(itLista)))             //  $O(|v| + |c|)$ 
            avanzar(itLista)                                                       //  $O(1)$ 
        end while
    end if
    return conjRes                                                                //  $O(1)$ 

```

```

imatchFrom(in tabla1 : tabla, in tabla2 : tabla, in c1 : string, in c2 : string )  $\rightarrow$  res : conj(registro)
  campoClave1  $\leftarrow$  tabla::clave(t1) //O(|c|)
  campoClave2  $\leftarrow$  tabla::clave(t2) //O(|c|)
  nroCampo1  $\leftarrow$  tabla::obtenerNroCampo(tabla1,camposClave1) //O(|c|)
  nroCampo2  $\leftarrow$  tabla::obtenerNrocampo(tabla2,campoClave2) //O(|c|)
  listaDePunteroReg1  $\leftarrow$  vacía() //O(1)
  listaDePunteroReg2  $\leftarrow$  vacía() //O(1)
  it1  $\leftarrow$  crearItConj(tabla::registros(tabla1)) //O(1)
  it2  $\leftarrow$  crearItConj(tabla::registros(tabla2)) //O(1)
  conjRes1  $\leftarrow$  vacío() //O(1)
  conjRes2  $\leftarrow$  vacío() //O(1)
  while haySiguiente(it1) && haySiguiente(it2) do //O(min(n1, n2))
    puntRegMatch1  $\leftarrow$  tabla::camposXClave(tabla2, tabla::valorEnRegistro(siguiente(it1),nroCampo1)) O(|v|)
    puntRegMatch2  $\leftarrow$  tabla::camposXClave(tabla1, tabla::valorEnRegistro(siguiente(it2),nroCampo2)) O(|v|)
    if !conjLineal::esVacio?(registro::campos(*puntRegMatch1)) then
      agregar(listaDePunteroReg1,  $\langle$  it1,puntRegMatch1  $\angle$  )
    end if
    if !conjLineal::esVacio?(registro::campos(*puntRegMatch2)) then
      agregar(listaDePunteroReg1,  $\langle$  it2,puntRegMatch2  $\angle$  )
    end if
    avanzar(it1) //O(1)
    avanzar(it2) //O(1)
  end while
  itLista1  $\leftarrow$  crearItLista(listaDeItReg1) //O(1)
  while haySiguiente(itLista1) do //O(k) * O(|v| + |c|)
    agregarRapido(conjRes1,tabla::mergeReg(siguiente(siguiente(itLista1).it1),*(siguiente(itLista1).puntRegMatch1))) O(|v| + |c|) + O(1)
    avanzar(itLista1) O(1)
  end while
  itLista2  $\leftarrow$  crearItLista(listaDeItReg2) //O(1)
  while haySiguiente(itLista2) do //O(k) * O(|v| + |c|)
    agregarRapido(conjRes2,tabla::mergeReg(siguiente(siguiente(itLista2).it2),*(siguiente(itLista2).puntRegMatch2))) O(|v| + |c|) + O(1)
    avanzar(itLista2) O(1)
  end while
  if cardinal(conjRes1) > cardinal(conjRes2) then
    return conjRes1
  end if
  return conjRes2

```

```

iSelectOptimizado(in bd: estr, in q: consulta) → res : conj(registro)
  nombreTabla ← consulta::nombreTabla(consulta::subconsultaUno(consulta::subconsultaUno(q))) //O(|c|)
  tablaEnBd ← base_de_datos::tabla(bd,nombreTabla) //O(|t|)
  c1 ← consulta::primerCampo(q) //O(1)
  c2 ← consulta::primerCampo(consulta::subconsulta1(q)) //O(1)
  v1 ← consulta::valor(q) //O(1)
  v2 ← consulta::valor(consulta::subconsulta1(q)) //O(1)
  nroC2 ← tabla::obtenerNroCampo(tablaEnBd,c2) //O(|c1|)
  conjSelect1 ← base_de_datos::select(tablaEnBd,c1,v1) //O(|c1| + |v1|)
  itConjSelect1 ← itConjLineal::crearIt(conjSelect1) //O(1)
  conjRes = conjLineal::vacio() //O(1)
  if itConjLineal::haySiguiente(itConjSelect1) then //O(1)
    reg ← siguiente(itConjSelect1) //O(1)
  end if
  itCamposReg::crearIt(registro::campos(reg)) //O(1)
  while itConjLineal::haySiguiente(itCamposReg) do //O(1)
    if nroC2 == tabla::obtenerNroCampo(tablaEnBd, siguiente(it)) then //O(1)
      nroCampoReg ← tabla::obtenerNroCampo(tablaEnBd,siguiente(it)) //O(|c|)
    end if
  end while
  if nroC2 == nro CampoReg && registro::valorEnRegistro(reg, nroCampoReg) == v2 then //O(|v2|)
    conjRes←conjLineal::agregarRapido(reg) //O(|v| + |c|)
  end if
  return conjRes //O(1)

```

```

iSelect(in bd: estr, in q: consulta) → res : conj(registro)
  conjRes ← base_de_datos::ejecutarConsulta(consulta::subconsulta1(q))
  itConjRes ← conjLineal::crearIt(conjRes) //O(1)
  while itConjLineal::haySiguiente(itConjRes) do //O(1)
    huboMatch ← false //O(1)
    itCamposReg = conjLineal::crearIt(registro::campos(itConjLineal::siguiente(itConjRes))) //O(1)
    reg = itConjLineal::siguiente(itConjRes)
    while itConjLineal::haySiguiente(itCamposReg) do //O(1)
      nombreCampo ← itConjLineal::siguiente(itCamposReg)
      if nombreCampo == consulta::nombreCampo1(q) reg[nombreCampo] == consulta::valor then //O(|c| + |v|)
        huboMatch←true //O(1)
      end if
      itConjLineal::avanzar(itCamposReg) //O(1)
    end while
    if !huboMatch then //O(1)
      conjLineal::eliminar(conjRes, reg) //O(k * (|v| + |c|))
    end if
    itConjLineal::avanzar(itConjRes) //O(1)
  end while
  return conjRes //O(1)

```

```

iInter(in bd: estr, in q: consulta) → res : conj(registro)
  conj1 ← base_de_datos::ejecutarConsulta(consulta::subconsulta1(q))
  conj2 ← base_de_datos::ejecutarConsulta(consulta::subconsulta2(q))
  conjRes ← base_de_datos::interseccion(conj1,conj2)
  return conjRes //O(1)

```

```

iUnion(in bd: estr, in q: consulta) → res : conj(registro)
  conj1 ← base_de_datos::ejecutarConsulta(consulta::subconsulta1(q))
  conj2 ← base_de_datos::ejecutarConsulta(consulta::subconsulta2(q))
  conjRes ← base_de_datos::union(conj1,conj2)
  return conjRes //O(1)

```

```

iProduct(in bd: estr, in q: consulta)  $\rightarrow$  res : conj(registro)
  conj1  $\leftarrow$  base_de_datos::ejecutarConsulta(consulta::subconsulta1(q))
  conj2  $\leftarrow$  base_de_datos::ejecutarConsulta(consulta::subconsulta2(q))
  conjRes  $\leftarrow$  base_de_datos::product(conj1,conj2)
  return conjRes

```

//O(1)

```

iDefinirRegistroRenameCampo(in/out reg: registro, in campo1: string, in campo2: string)
  regRes  $\leftarrow$  registro::nuevo()
  itCamposReg  $\leftarrow$  itConjLineal::crearIt(registro::campos(reg))
  while itConjLineal::haySiguiente(itCamposReg) do
    if itConjLineal::siguiente(itCamposReg) == campo1 then
      registro::definir(regRes, campo2, reg[itConjLineal::siguiente(itCamposReg)])
    end if
    itConjLineal::avanzar(itCamposReg)
  end while
  return regRes

```

```

iDefinirRegistroFiltroCampos(in/out reg: registro, in camposFiltro: conj(string))
  regRes  $\leftarrow$  registro::nuevo()
  itCamposReg  $\leftarrow$  itConjLineal::crearIt(registro::campos(reg))
  while itConjLineal::haySiguiente(itCamposReg) do
    nombreCampoReg  $\leftarrow$  itConjLineal::siguiente(itCamposReg)
    if !conjLineal::esVacía?(base_de_datos::filtrarCampo(nombreCampoReg, camposFiltro)) then
      registro::definir(regRes, nombreCampoReg, reg[nombreCampoReg])
    end if
    itConjLineal::avanzar(itCamposReg)
  end while
  return regRes

```

```

iInterseccion(in conj1: conj(registro), in conj2: conj(registro))  $\rightarrow$  res : conj(registro)
  conjRes  $\leftarrow$  conjLineal::vacío()
  itConj1  $\leftarrow$  itConjLineal::crearIt(conj1)
  while itConjLineal::haySiguiente(itConj1) do
    reg  $\leftarrow$  itConjLineal::siguiente(itConj1)
    if base_de_datos::pertenece(reg, conj2) then
      conjLineal::agregarRapido(conjRes, reg)
    end if
    itConjLineal::avanzar(itConj1)
  end while
  return conjRes

```

```

iUnion(in conj1: conj(registro), in conj2: conj(registro))  $\rightarrow$  res : conj(registro)
  conjRes  $\leftarrow$  conj2
  itConj1  $\leftarrow$  itConjLineal::crearIt(conj1)
  while itConjLineal::haySiguiente(itConj1) do
    reg  $\leftarrow$  itConjLineal::siguiente(itConj1)
    conjLineal::agregar(conjRes, reg)
    itConjLineal::avanzar(itConj1)
  end while
  return conjRes

```

iProduct(in *conj1*: conj(registro), in *conj2*: conj(registro)) → *res*: conj(registro)

```
conjRes ← conjLineal::vacio()
itConj1 ← itConjLineal::crearIt(conj1)
while itConjLineal::haySiguiente(itConj1) do
  regsMergeados ← itConjLineal::crearIt(regsMergeados)
  while itConjLineal::haySiguiente(itRegsMergeados) do
    conjLineal::agregar(conjRes, itConjLineal::siguiente(itRegsMergeados))
    itConjLineal::avanzar(itRegsMergeados)
  end while
  itConjLineal::avanzar(itConj1)
end while
return conjRes
```

iMergeRegs(in *regToMerge*: registro, in *conjRegs*: conj(registro)) → *res*: conj(registro)

```
conjRes ← conjLineal::vacio()
itConj ← itConjLineal::crearIt(conjRegs)
while itConjLineal::haySiguiente(itConj) do
  reg ← base_de_datos::mergeDosRegs(regToMerge, itConjLineal::siguiente(itConj))
  conjLineal::agregar(conjRes, reg)
  itConjLineal::avanzar(itConj)
end while
return conjRes
```

iMergeDosRegs(in *reg1*: registro, in *reg2*: registro) → *res*: registro

```
regRes ← reg1
itCampos ← itConjLineal::crearIt(registro::campos(reg2))
while itConjLineal::haySiguiente(itCampos) do
  if conjLineal::esVacia?base_de_datos::interseccion(registro::campos(reg1), registro::campos(reg2)) then
    registro::definir(regRes, itConjLineal::siguiente(itCampos), registro::reg2[itConjLineal::siguiente(itCampos)])
  end if
  itConjLineal::avanzar(itCampos)
end while
return regRes
```

iInterseccion(in *conjCampos1*: conj(string), in *conjCampos2*: conj(string)) → *res*: conj(string)

```
conjRes ← conjLineal::vacio()
itCampos1 ← itConjLineal::crearIt(conjCampos1)
while itConjLineal::haySiguiente(itCampos1) do
  campo ← itConjLineal::siguiente(itCampos1)
  if base_de_datos::pertenece(campo, conjCampos2) then
    conjLineal::agregarRapido(conjRes, campo)
  end if
  itConjLineal::avanzar(itCampos1)
end while
return conjRes
```

iFiltrarCampo(in *campo1*: string, in *conjCampos2*: conj(string)) → *res*: string

```
conjRes ← conjLineal::vacio()
if base_de_datos::pertenece(campo1, conjCampos2) then
  conjLineal::agregarRapido(conjRes, campo1)
end if
return conjRes
```

```

iPertenece(in reg: registro, in conj: conj(registro))  $\rightarrow res : bool$ 
  itConj  $\leftarrow$  itConjLineal::crearIt(conj)
  while itConjLineal::haySiguiente(itConj) && reg  $\neq$  itConjLineal::siguiente(itConj) do
    itConjLineal::avanzar(itConj)
  end while
  return itConjLineal::haySiguiente(itConj)

```

```

iPertenece(in nc: string, in conj: conj(string))  $\rightarrow res : bool$ 
  itConj  $\leftarrow$  itConjLineal::crearIt(conj)
  while itConjLineal::haySiguiente(itConj) && nc  $\neq$  itConjLineal::siguiente(itConj) do
    itConjLineal::avanzar(itConj)
  end while
  return itConjLineal::haySiguiente(itConj)

```

```

iSelectProductOptimizado(in bd: estr, in q: consulta)  $\rightarrow res : conj(registro)$ 
  tabla1  $\leftarrow$  base_de_datos::tabla(bd, consulta::nombreTabla(consulta::subconsultaUno(consulta::subconsultaUno(q))))
  //O(|t|)
  tabla2  $\leftarrow$  base_de_datos::tabla(bd, consulta::nombreTabla(consulta::subconsultaDos(consulta::subconsultaUno(q))))
  //O(|t|)
  conjRes  $\leftarrow$  conjLineal::vacio() //O(1)
  regTabla1 = *tabla::camposXClave(tabla1, consulta::valor(q)) //O(|v|)
  itConj = conjLineal::crearIt(tabla::registros(tabla2)) //O(n2 * (|v| + |c|))
  while (itConjLineal::haySiguiente(itConj)) do //O(|n2|)
    regRes  $\leftarrow$  mergeReg(regTabla1, itConjLineal::siguiente(itConj)) //O(|v| + |c|)
    conjLineal::agregarRapido(conjRes, regRes) //O(|v| + |c|)
  end while
  return conjRes //O(1)

```

```

iMatch(in bd: estr, in q: consulta)  $\rightarrow res : conj(registro)$ 
  conjRes  $\leftarrow$  base_de_datos::ejecutarConsulta(consulta::subconsulta1(q)) //O(|ejecutarConsulta(subconsulta1(q))|)
  itConjRes  $\leftarrow$  conjLineal::crearIt(conjRes) //O(|l|)
  while (itConjLineal::haySiguiente(itConjRes)) do //O(|l|)
    reg  $\leftarrow$  itConjLineal::siguiente(itConjRes) //O(|l|)
    if (reg[consulta::primerCampo(q)] != reg[consulta::segundoCampo(q)] then
      conjLineal::eliminar(conjRes, reg) //O(k * (|c| + |v|))
    end if
    itConjLineal::avanzar(itConjRes) //O(|l|)
  end while
  return conjRes //O(|l|)

```

```

iProj(in bd: estr, in q: consulta)  $\rightarrow res : conj(registro)$ 
  conjRes  $\leftarrow$  base_de_datos::ejecutarConsulta(consulta::subconsulta1(q)) //O(|ejecutarConsulta(subconsulta1(q))|)
  itConjRes  $\leftarrow$  conjLineal::crearIt(conjRes) //O(|l|)
  camposDeConsulta  $\leftarrow$  consulta::conjCampos(q) //O(|l|)
  while (itConjLineal::haySiguiente(itConjRes)) do //O(|l|)
    regRes  $\leftarrow$  base_de_datos::definirRegistroFiltroCampos(itConjLineal::siguiente(itConjRes), camposDeConsulta)
    conjLineal::agregarRapido(conjRes, regRes) //O(|c| + |v|)
    itConjLineal::avanzar(itConjRes) //O(|l|)
  end while
  return conjRes //O(|l|)

```

```

iRename(in bd: estr, in q: consulta)  $\rightarrow$  res : conj(registro)
  conjRes  $\leftarrow$  base_de_datos::ejecutarConsulta(consulta::subconsulta1(q))    //  $O(|ejecutarConsulta(subconsulta1(q))|)$ 
  itConjRes  $\leftarrow$  conjLineal::crearIt(conjRes)                                //  $O(|1|)$ 
  camposDeConsulta  $\leftarrow$  consulta::conjCampos(q)                          //  $O(|1|)$ 
  while (itConjLineal::haySiguiente(itConjRes)) do                            //  $O(|1|)$ 
    regRes  $\leftarrow$  base_de_datos::definirRegistroRenameCampo(itConjLineal::siguiente(itConjRes),    consul-
    ta::nombreCampo1(q), consulta::nombreCampo2(q))                                //
     $O(|1|)$ 
    conjLineal::agregarRapido(conjRes, regRes)                                    //  $O(|c| + |v|)$ 
    itConjLineal::avanzar(itConjRes)                                              //  $O(|1|)$ 
  end while
  return conjRes                                                                //  $O(|1|)$ 

```

```

iEjecutarConsulta(in bd: estr, in q: consulta) → res : conj(registro)
conjRes ← conjLineal::vacio() //O(|1|)
switch(consulta::tipoConsulta(q)) //O(|1|)
{
CASE 'FROM': //O(|1|)
tablaEnBd ← base_de_datos::tabla(bd, consulta::nombreTabla(q)) //O(|t|)
conjRes ← tabla::registros(tablaEnBd)
break; //O(|1|)
CASE 'SELECT': //O(|1|)
if (consulta::tipoConsulta(consulta::subconsultaUno(q)) == 'FROM') then //O(|1|)
    tablaEnBd ← base_de_datos::tabla(bd, consulta::nombreTabla(consulta::subconsultaUno(q))) //O(|t|)
    conjRes ← selectFrom(tablaEnBd, consulta::primerCampo(q), consulta::valor(q)) //O(|?|)
else if (consulta::tipoConsulta(consulta::subconsultaUno(q)) == 'SELECT' && consulta::tipoConsulta(consulta::subconsultaUno(consulta::subconsultaUno(q))) == 'FROM') then //O(|1|)
    tablaEnBd ← base_de_datos::tabla(bd, consulta::nombreTabla(consulta::subconsultaUno(consulta::subconsultaUno(q)))) //O(|t|)
    if (base_de_datos::esClaveTabla(tablaEnBd, consulta::primerCampo(consulta::subconsultaUno(q))) && !base_de_datos::esClaveTabla(tablaEnBd, consulta::primerCampo(consulta::subconsultaUno(consulta::subconsultaUno(q)))) then //O(|c|)
        conjRes ← base_de_datos::selectOptimizado(bd, q)
    end if
else if (consulta::tipoConsulta(consulta::subconsultaUno(q)) == 'PRODUCT' && consulta::tipoConsulta(consulta::subconsultaUno(consulta::subconsultaUno(q))) == 'FROM' && consulta::tipoConsulta(consulta::subconsultaDos(consulta::subconsultaUno(q))) == 'FROM') then //O(|1|)
    conjRes ← selectProductOptimizado(bd, q)
else
    conjRes ← select(bd, q)
end if
break; //O(|1|)
CASE 'MATCH': //O(|1|)
if (consulta::tipoConsulta(consulta::subconsulta1(q)) == 'PRODUCT' && consulta::tipoConsulta(consulta::subconsultaUno(consulta::subconsultaUno(q))) == 'FROM' && consulta::tipoConsulta(consulta::subconsultaDos(consulta::subconsultaUno(q))) == 'FROM') then
    tablaEnBd1 ← base_de_datos::tabla(bd, consulta::nombreTabla(consulta::subconsultaUno(consulta::subconsultaUno(q))))
    tablaEnBd2 ← base_de_datos::tabla(bd, consulta::nombreTabla(consulta::subconsultaDos(consulta::subconsultaUno(q))))
    conjRes ← matchFrom(tablaEnBd1, tablaEnBd2, consulta::nombreCampo1(consulta::subconsulta1(q)), consulta::nombreCampo2(consulta::subconsulta1(q)))
else
    conjRes ← base_de_datos::match(bd, q)
end if
break; //O(|1|)
CASE 'PROJ': //O(|1|)
conjRes ← base_de_datos::proj(bd, q)
break; //O(|1|)
CASE 'RENAME': //O(|1|)
conjRes ← base_de_datos::rename(bd, q)
break; //O(|1|)
CASE 'INTER': //O(|1|)
conjRes ← base_de_datos::inter(bd, q)
break; //O(|1|)
CASE 'UNION': //O(|1|)
conjRes ← base_de_datos::union(bd, q)
break; //O(|1|)
CASE 'PRODUCT': //O(|1|)
conjRes ← base_de_datos::product(bd, q)
CASE 'DEFAULT': //O(|1|)
conjRes ← conjLineal::vacio()
break; //O(|1|)
}
return conjRes; //O(|1|)

```

2. Modulo Consulta

El modulo Consulta provee operaciones básicas para generar una consulta. Permite construir una consulta a partir de una tabla.

Para describir la complejidad de las operaciones, el costo de copiar un $s \in string$ e igualar $s_1, s_2 \in string$ es igual a $O(|s|)$. Y el costo de copiar una consulta q sera expresada como $copy(q)$ y sera tomado como la copia recursiva de todos los campos correspondiente en cada una de las instancias de la recursion.

Interfaz

se explica con: CONSULTA

usa: REGISTRO, TABLA, BOOL, NAT, STRING

género: consulta

FROM(**in** nt : nombreTabla) $\rightarrow res$: Consulta

Pre $\equiv \{true\}$

Post $\equiv \{res = FROM(nt)\}$

Complejidad: $O(|nt|)$

Descripción: Crea una consulta c a partir de un nombreTabla nt , que tiene las siguientes particularidades: TipoConsulta(c): FROM, NombreTabla(c): nt

SELECT(**in** c : Consulta, **in** nc : nombreCampo, **in** v : Valor) $\rightarrow res$: Consulta

Pre $\equiv \{true\}$

Post $\equiv \{res = SELECT(c, nc, v)\}$

Complejidad: $O(|nc| + |v| + copy(c))$

Descripción: Dada una consulta c , un nombreCampo nc y un valor v , SELECT es una nueva consulta que contiene las siguientes particularidades: TipoConsulta(res) : SELECT, PrimerCampo(res): nombre, Valor(res): v , SubconsultaUno(res): c

MATCH(**in** c : Consulta, **in** $nc1$: nombreCampo, **in** $nc2$: nombreCampo) $\rightarrow res$: Consulta

Pre $\equiv \{true\}$

Post $\equiv \{res = MATCH(c0, nc1, nc2)\}$

Complejidad: $O(|nc1| + |nc2| + copy(c))$

Descripción: Dada una consulta c y dos nombreCampo $nc1$ y $nc2$, MATCH es una nueva consulta que contiene las siguientes particularidades: TipoConsulta(res) : MATCH, PrimerCampo(res): $nc1$, SegundoCampo(res): $nc2$, SubconsultaUno(res): c

PROJ(**in** c : Consulta, **in** cnc : conj(nombreCampo)) $\rightarrow res$: Consulta

Pre $\equiv \{true\}$

Post $\equiv \{c = PROJ(c, cnc)\}$

Complejidad: $O(|cnc| + copy(c))$

Descripción: Dada una consulta c y un conj(nombreCampo) campos, PROJ es una nueva consulta que contiene las siguientes particularidades: TipoConsulta(nc) : PROJ, Conj_Campos(nc): campos, SubconsultaUno(nc): c

RENAME(**in** c : Consulta, **in** $nc1$: nombreCampo, **in** $nc2$: nombreCampo) $\rightarrow res$: Consulta

Pre $\equiv \{true\}$

Post $\equiv \{c = RENAME(c, nc1, nc2)\}$

Complejidad: $O(|nc1| + |nc2| + copy(c))$

Descripción: Dada una consulta c y dos nombresCampo $nc1$ y $nc2$, RENAME es una nueva consulta que contiene las siguientes particularidades: TipoConsulta(res) : RENAME, PrimerCampo(res) : $nc1$, SegundoCampo(res) : $nc2$, SubconsultaUno(res): c

INTER(**in** $q1$: consulta, **in** $q2$: Consulta) $\rightarrow res$: Consulta

Pre $\equiv \{true\}$

Post $\equiv \{c = INTER(q1, q2)\}$

Complejidad: $O(copy(q1) + (copy(q1)))$

Descripción: Dada una consulta c y dos nombresCampo $nc1$ y $nc2$, INTER es una nueva consulta que contiene las siguientes particularidades: TipoConsulta(res) : INTER, SubconsultaUno(res): $q1$ SubconsultaDos(res): $q2$

UNION(**in** $q1$: consulta, **in** $q2$: Consulta) $\rightarrow res$: Consulta

Pre $\equiv \{true\}$

Post $\equiv \{c = UNION(q1, q2)\}$

Complejidad: $O(copy(q1) + (copy(q1)))$

Descripción: Dada una consulta c y dos nombresCampo $nc1$ y $nc2$, UNION es una nueva consulta que contiene las siguientes particularidades: TipoConsulta(res) : UNION, SubconsultaUno(res): $q1$ SubconsultaDos(res): $q2$

PRODUCT(**in** $q1: consulta$, **in** $q2: Consulta$) $\rightarrow res : Consulta$

Pre $\equiv \{true\}$

Post $\equiv \{c = PRODUCT(q1, q2)\}$

Complejidad: $O(copy(q1) + (copy(q1)))$

Descripción: Dada una consulta c y dos nombresCampo $nc1$ y $nc2$, UNION es una nueva consulta que contiene las siguientes particularidades: TipoConsulta(res) : PRODUCT, SubconsultaUno(res): $q1$ SubconsultaDos(res): $q2$

TIPOCONSULTA(**in** $c: Consulta$) $\rightarrow res : tipoConsulta$

Pre $\equiv \{true\}$

Post $\equiv \{res = tipo_consulta(c)\}$

Complejidad: $O(1)$

Descripción: Dada una consulta, devuelve su tipo

NOMBRETABLA(**in** $c: Consulta$) $\rightarrow res : nombreTabla$

Pre $\equiv \{tipo_consulta(c) \in \{FROM\}\}$

Post $\equiv \{res = nombre_tabla(c)\}$

Complejidad: $O(copy(nombreTabla))$

Descripción: Dada una consulta de tipo FROM, devuelve la tabla asociada a dicha consulta

PRIMERCAMPO(**in** $c: Consulta$) $\rightarrow res : nombre_campo$

Pre $\equiv \{tipo_consulta(c) \in \{SELECT, MATCH, RENAME\}\}$

Post $\equiv \{res = campo1(c)\}$

Complejidad: $O(copy(nombreCampo))$

Descripción: Dada una consulta de tipo FROM, MATCH O RENAME, devuelve el primer parámetro de tipo nombreCampo asociado a dicha consulta

SEGUNDOCAMPO(**in** $c: Consulta$) $\rightarrow res : nombre_campo$

Pre $\equiv \{tipo_consulta(c) \in \{MATCH, RENAME\}\}$

Post $\equiv \{res = campo2(c)\}$

Complejidad: $O(copy(nombreCampo))$

Descripción: Dada una consulta de tipo MATCH O RENAME, devuelve el segundo parámetro de tipo nombreCampo asociado a dicha consulta

VALOR(**in** $c: Consulta$) $\rightarrow res : valor$

Pre $\equiv \{tipo_consulta(c) \in \{SELECT\}\}$

Post $\equiv \{res = valor(c)\}$

Complejidad: $O(copy(valor))$

Descripción: Dada una consulta de tipo SELECT($consulta$, $nombreCampo$, $valor$), devuelve el valor asociado a dicha consulta

CONJCAMPOS(**in** $c: Consulta$) $\rightarrow res : Conj(nombreCampo)$

Pre $\equiv \{tipo_consulta(c) \in \{PROJ\}\}$

Post $\equiv \{res = conj_campos(c)\}$

Complejidad: $O(copy(conj(nombreCampo)))$

Descripción: Dada una consulta de tipo PROJ($consulta$, $conjCampos$), devuelve el conjunto de campos asociado a dicha consulta

SUBCONSULTAUNO(**in/out** $c: Consulta$)

Pre $\equiv \{c = c0 \wedge tipo_consulta(c0) \in \{SELECT, MATCH, PROJ, RENAME, INTER, UNION, PRODUCT\}\}$

Post $\equiv \{c = subconsulta1(c0)\}$

Complejidad: $O(copy(consulta))$

Descripción: Dada una consulta de tipo SELECT, MATCH, PROJ, RENAME, INTER, UNION O PRODUCT, devuelve la primer subconsulta asociada a dicha consulta

SUBCONSULTADOS(**in/out** $c: Consulta$)

Pre $\equiv \{c = c0 \wedge tipo_consulta(c0) \in \{INTER, UNION, PRODUCT\}\}$

Post $\equiv \{c = subconsulta2(c0)\}$

Complejidad: $O(copy(consulta))$

Descripción: Dada una consulta de tipo INTER, UNION, PRODUCT, nos da la segunda subconsulta asociada a dicha consulta.

Representación

La consulta se representan de forma recursiva, con los campos correspondientes a la consulta adecuada.

consulta se representa con *estr*

donde *estr* es `tupla(tipoConsulta: string,
nombreTabla: string,
campo1: string,
campo2: string,
valor: string,
conjCampos: conj(string),
subConsulta1: consulta,
subConsulta2: consulta)`

Rep

- Para todo *c* de tipo consulta, el campo *c.tipoConsulta* debe pertenecer al conjunto {FROM, SELECT, MATCH, RENAME, PROJ, INTER, UNION, PRODUCT}
- Si *c.tipoConsulta* es FROM: tendra que tener un *c.nombreTabla* valido, y todos los demas campos seran nulos (segun su tipo)
- Si *c.tipoConsulta* es SELECT: tendra que tener un *c.nombreCampo1* y *c.valor* validos, y todos los demas campos seran nulos (segun su tipo)
- Si *c.tipoConsulta* es MATCH: tendra que tener un *c.nombreCampo1* y *c.nombreCampo2* validos, y todos los demas campos seran nulos (segun su tipo)
- Si *c.tipoConsulta* es PROJ: tendra que tener un *c.conjCampos* valido, y todos los demas campos seran nulos (segun su tipo)
- Si *c.tipoConsulta* es RENAME: tendra que tener un *c.nombreCampo1* y *c.nombreCampo2* validos, y todos los demas campos seran nulos (segun su tipo)
- Si *c.tipoConsulta* es INTER: tendra que tener un *c.subconsulta1* y *c.subconsulta2* validos, y todos los demas campos seran nulos (segun su tipo)
- Si *c.tipoConsulta* es UNION: tendra que tener un *c.subconsulta1* y *c.subconsulta2* validos, y todos los demas campos seran nulos (segun su tipo)
- Si *c.tipoConsulta* es PRODUCT: tendra que tener un *c.subconsulta1* y *c.subconsulta2* validos, y todos los demas campos seran nulos (segun su tipo)

Abs

Abs : *estr e* \rightarrow consulta

{Rep(*e*)}

Abs(*e*) \equiv *c*: consulta | *tipo_consulta*(*c*) =_{obs} *e.tipoConsulta*
nombre_tabla(*c*) =_{obs} *e.nombreTabla*
campo1(*c*) =_{obs} *e.campo1*
campo2(*c*) =_{obs} *e.campo2*
valor(*c*) =_{obs} *e.valor*
conj_campos(*c*) =_{obs} *e.conjCampos*
subconsulta1(*c*) =_{obs} *e.subconsulta1*
subconsulta2(*c*) =_{obs} *e.subconsulta2*

Algoritmos

iFrom(in *nt*: nombreTabla) \rightarrow *res*: *estr*

1: *res* \leftarrow (FROM, *nt*, string::Vacio(), string::Vacio(), string::Vacio(), conj::Vacio(), NULL, NULL)

Complejidad: $O(|nt|)$

iSelect(in *c*: *estr*, in *nc*: nombreCampo, in *v*: valor,) \rightarrow *res*: *estr*

1: *res* \leftarrow (SELECT, string::Vacio(), *nc*, string::Vacio(), *v*, conj::Vacio(), *c*, NULL)

Complejidad: $O(|nc| + |v| + copy(c))$

iMatch(in c : **estr**, in $nc1$: **nombreCampo**, in $nc2$: **nombreCampo**) $\rightarrow res$: **estr**

1: $res \leftarrow \langle \text{MATCH}, \text{string}::\text{Vacio}(), nc1, nc2, \text{string}::\text{Vacio}(), \text{conj}::\text{Vacio}(), c, \text{NULL} \rangle$

Complejidad: $O(|nc1| + |nc2| + \text{copy}(c))$

iProj(in c : **estr**, in cnc : **conj**(**nombreCampo**)) $\rightarrow res$: **estr**

1: $res \leftarrow \langle \text{PROJ}, \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), cnc, c, \text{NULL} \rangle$

Complejidad: $O(|cnc| + \text{copy}(c))$

iRename(in c : **estr**, in $nc1$: **nombreCampo**, in $nc2$: **nombreCampo**) $\rightarrow res$: **estr**

1: $res \leftarrow \langle \text{RENAME}, \text{string}::\text{Vacio}(), nc1, nc2, \text{string}::\text{Vacio}(), \text{conj}::\text{Vacio}(), c, \text{NULL} \rangle$

Complejidad: $O(|nc1| + |nc2| + \text{copy}(c))$

iInter(in $c1$: **estr**, in $c2$: **estr**) $\rightarrow res$: **estr**

1: $res \leftarrow \langle \text{INTER}, \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{conj}::\text{Vacio}(), c1, c2 \rangle$

Complejidad: $O(\text{copy}(c1) + \text{copy}(c2))$

iUnion(in $c1$: **estr**, in $c2$: **estr**) $\rightarrow res$: **estr**

1: $res \leftarrow \langle \text{UNION}, \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{conj}::\text{Vacio}(), c1, c2 \rangle$

Complejidad: $O(\text{copy}(c1) + \text{copy}(c2))$

iProduct(in $c1$: **estr**, in $c2$: **estr**) $\rightarrow res$: **estr**

1: $res \leftarrow \langle \text{PRODUCT}, \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{string}::\text{Vacio}(), \text{conj}::\text{Vacio}(), c1, c2 \rangle$

Complejidad: $O(\text{copy}(c1) + \text{copy}(c2))$

iTipoConsulta(in c : **estr**) $\rightarrow res$: **tipoConsulta**

1: $res \leftarrow c.\text{tipoConsulta}$

Complejidad: $O(\text{tipoConsulta}) = O(1)$

iNombreTabla(in c : **estr**) $\rightarrow res$: **nombreTabla**

1: $res \leftarrow c.\text{nombreTabla}$

Complejidad: $O(\text{copy}(\text{nombreTabla}))$

iPrimerCampo(in c : **estr**) $\rightarrow res$: **nombreCampo**

1: $res \leftarrow c.\text{campo1}$

Complejidad: $O(\text{copy}(\text{nombreCampo}))$

iSegundoCampo(in c : **estr**) $\rightarrow res$: **nombreCampo**

1: $res \leftarrow c.\text{campo2}$

Complejidad: $O(\text{copy}(\text{nombreCampo}))$

iValor(in c : **estr**) $\rightarrow res$: valor

1: $res \leftarrow c.valor$

Complejidad: $O(copy(valor))$

iConjCampos(in c : **estr**) $\rightarrow res$: conj(nombreCampo)

1: $res \leftarrow c.conjCampos$

Complejidad: $O(copy(conj(nombreCampo)))$

iSubconsulta1(in c : **estr**) $\rightarrow res$: consulta

1: $res \leftarrow c.subconsulta1$

Complejidad: $O(copy(consulta))$

iSubconsulta2(in c : **estr**) $\rightarrow res$: consulta

1: $res \leftarrow c.subconsulta2$

Complejidad: $O(copy(consulta))$

copiarConsulta(in c : **estr**) $\rightarrow res$: consulta

1: $res.nombreTabla \leftarrow c.nombreTabla$

2: $res.campo1 \leftarrow c.campo1$

3: $res.campo2 \leftarrow c.campo2$

4: $res.valor \leftarrow c.valor$

5: $res.conjCampos \leftarrow c.conjCampos$

6: $res.subConsulta1 \leftarrow c.subConsulta1$

7: $res.subConsulta2 \leftarrow c.subConsulta2$

Complejidad: $copy(c) = O(|c.nombreTabla|, |c.campo1|, |c.campo2|, |c.valor|, copy(c.conjCampos), copy(c.subConsulta1), copy(c.subConsulta2))$

3. Modulo Registro

El modulo Registro provee operaciones básicas a realizarse sobre un registro. Permite crear un nuevo registro, obtener los campos de un registro, y obtener el valor de uno de los campos de un registro. Para describir la complejidad de las operaciones, el costo de copiar un $s \in string$ e igualar $s_1, s_2 \in string$ es igual a $O(|s|)$.

Interfaz

se explica con: REGISTRO

usa: REGISTRO, TABLA, CONSULTA, BOOL, NAT, STRING

género: registro

NUEVO() $\rightarrow res : Registro$

Pre $\equiv \{true\}$

Post $\equiv \{res = Registro\}$

Complejidad: $O(1)$

Descripción: Crea un nueva registro vacio.

Aliasing: No Aplica

DEFINIR(in/out $r : Registro$, in $nc : nombreCampo$, in $v : Valor$)

Pre $\equiv \{r = r0 \wedge (nc \notin campos(r))\}$

Post $\equiv \{res = definir(r0, nc, v)\}$

Complejidad: $O(|nc| + |v|)$

Descripción: Define un nuevo campo y su valor correspondiente dentro de un registro.

Aliasing: r es modificable

CAMPOS(in $r : Registro$) $\rightarrow res : Conj(nombreCampo)$

Pre $\equiv \{true\}$

Post $\equiv \{res = campos(r)\}$

Complejidad: $O(|c|)$

Descripción: Devuelve las campos de un registro ($conj(nombreCampo)$).

Aliasing: no aplica

• [•](in $r : Registro$, in $nc : NombreCampo$) $\rightarrow res : Valor$

Pre $\equiv \{nc \in campos(r)\}$

Post $\equiv \{res = r[nc]\}$

Complejidad: $O(|c| + |v|)$

Descripción: Devuelve el valor del campo de un registro.

Aliasing: no aplica

VALORXNROCAMPO(in $r : Registro$, in $nroCampo : nat$) $\rightarrow res : Valor$

Pre $\equiv \{nroCampo \leq \#campos(r)\}$

Post $\equiv \{res = r[ordenLexicografico(r, nc)]\}$

Complejidad: $O(|v|)$

Descripción: Devuelve el valor del campo de un registro.

Aliasing: no aplica

ordenLexicografico : registro $r \times nombreCampo nc \rightarrow nat$ $\{nc \in campos(r)\}$

ordenLexicografico(r, nc) $\equiv auxOrdenLex(campos(r), nc, 1)$

auxOrdenLex : conj(nombreCampo) $cnc \times nombreCampo nc \times nat \rightarrow nat$ $\{nc \in cnc\}$

auxOrdenLex(cnc, nc, n) \equiv
 if ($\emptyset?(cnc)$) **then**
 n
 else
 if ($dameUno(cnc) < nc$) **then**
 auxOrdenLex(sinUno(cnc), nc, n + 1)
 else
 auxOrdenLex(sinUno(cnc), nc, n)
 fi
 fi

• < • : string \times string $\rightarrow bool$

• < • ($s1, s2$) \equiv **if ($vacia?(s1) \vee_L vacia?(s2)$) **then** vacia?(s1) **else** prim(s1) < prim(s2) (fin(s2) < fin(s2)) **fi****

Representación

El registro se representa con un conjunto de tuplas que guardan toda la información requerida. Para describir la complejidad de las operaciones, el costo de copiar un $s \in \text{string}$ e igualar $s_1, s_2 \in \text{string}$ es igual a $O(|s|)$.

registro se representa con estr

donde estr es $\text{conj}(\text{Tupla} < \text{idCampo} : \text{nat}, \text{nc} : \text{string}, \text{valor} : \text{string} >)$

Rep

- No existe una tupla que tenga el mismo nc e idCampo que otra dentro del conjunto
- Cada idCampo de cualquier tupla tiene que estar mapeado correctamente con el orden lexicográfico asignado a cada nc, o sea que para toda tupla, tiene que cumplirse que el idCampo sea el correspondiente a nc

Abs

$\text{Abs} : \text{estr } e \rightarrow \text{registro} \quad \{\text{Rep}(e)\}$

$\text{Abs}(e) \equiv r : \text{registro} \mid \text{campos}(r) =_{\text{obs}} \text{filtroDeCampos}(e, \emptyset)$
 $(\forall \text{nc} : \text{nombre_campo})(\text{nc} \in \text{campos}(r) \Rightarrow r[\text{nc}] =_{\text{obs}} \text{buscarNC}(e, \text{nc}))$

$\text{filtroDeCampos} : \text{estr} \times \text{conj}(\text{string}) \rightarrow \text{conj}(\text{string})$

$\text{filtroDeCampos}(e, \text{res}) \equiv \text{if } (\text{vacía?}(e)) \text{ then } \text{res} \text{ else } \text{filtroDeCampos}(\text{sinUno}(e), \text{Ag}(\text{res}, \text{dameUno}(e).\text{nc})) \text{ fi}$

$\text{buscarNC} : \text{estr} \times \text{nombre_campo} \rightarrow \text{valor} \quad \{(\exists \text{tupla} : < \text{nat}, \text{string}, \text{string} >)(\text{tupla} \in \text{estr} \Rightarrow \text{nc} = \text{tupla}.\text{nc})\}$

$\text{buscarNC}(e, \text{nc}) \equiv \text{if } (\text{dameUno}(e).\text{nc} = \text{nc}) \text{ then } \text{dameUno}(e).\text{valor} \text{ else } \text{buscarNC}(\text{sinUno}(e), \text{nc}) \text{ fi}$

Algoritmos

iNuevo() $\rightarrow \text{res} : \text{estr}$

$\text{res} \leftarrow \text{conj}::\text{vacío}() \quad //O(1)$

iDefinir(in/out $r : \text{estr}$, in $\text{nc} : \text{nombreCampo}$, in $v : \text{valor}$)

$\text{nroCampo} \leftarrow 0 \quad //O(1)$

$\text{itCampos} \leftarrow \text{conj}::\text{crearIt}(r) \quad //O(1)$

while $\text{conj}::\text{haySiguiente}(\text{itCampos})$ **do** $//O(|\text{nc}|)$

if $\text{menorLexicografico}(\text{siguiente}(\text{itCampos}), \text{nombreCampo})$ **then** $//O(|\text{nc}|)$

$\text{nroCampo} \leftarrow \text{nroCampo} + 1 \quad //O(1)$

end if

$\text{conj}::\text{avanzar}(\text{itCampos}) \quad //O(1)$

end while

$\text{conj}::\text{agregar}(r, \langle \text{nroCampo}, \text{nc}, v \rangle) \quad //O(|\text{nc}| + |v|)$

iMenorLexicografico(in $s_1 : \text{string}$, in $s_2 : \text{string} \rightarrow \text{res} : \text{bool}$)

$\text{menor} \leftarrow \text{false} \quad //O(1)$

$\text{it1} \leftarrow \text{string}::\text{itCrear}(s_1) \quad //O(1)$

$\text{it2} \leftarrow \text{string}::\text{itCrear}(s_2) \quad //O(1)$

while $\text{string}::\text{haySiguiente}(\text{it1}) \ \&\& \ \text{string}::\text{haySiguiente}(\text{it2})$ **do** $//O(\min(|s_1|, |s_2|))$

if $\text{siguiente}(\text{it1}) < \text{siguiente}(\text{it2})$ **then** $//O(1)$

$\text{menor} \leftarrow \text{true} \quad //O(1)$

end if

$\text{string}::\text{avanzar}(\text{it1}) \quad //O(1)$

$\text{string}::\text{avanzar}(\text{it2}) \quad //O(1)$

end while

$\text{return } \text{menor} \quad //O(1)$

```

iCampos(in  $r : \text{estr}$ )  $\rightarrow res : \text{conj}(\text{numeroCampo})$ 
  itCampos  $\leftarrow \text{conj}::\text{crearIt}(r)$  //O(1)
  while  $\text{conj}::\text{haySiguiente}(\text{itCampos})$  do //O(|c|)
    res  $\leftarrow \text{conj}::\text{agregar}(r, \text{siguiente}(\text{itCampos}))$  //O(|c|)
    avanzar(itCampos) //O(1)
  end while
  return res //O(1)

```

```

i[•](in  $r : \text{estr}$ , in  $nc : \text{numeroCampo} \rightarrow v : \text{valor}$ )
  itCampos  $\leftarrow \text{conj}::\text{crearIt}(r)$  //O(1)
  while  $\text{conj}::\text{haySiguiente}(\text{itCampos})$  do //O(|v| + |c|)
    if  $nc == \text{siguiente}(\text{itCampos}).nc$  then //O(|v|)
      res  $\leftarrow \text{siguiente}(\text{itCampos}).valor$  //O(|v|)
    end if
    avanzar(itCampos) //O(1)
  end while
  return res //O(1)

```

```

iValorXNroCampo(in  $r : \text{estr}$ , in  $nroCampo : \text{nat} \rightarrow res : \text{valor}$ )
  itCampos  $\leftarrow \text{crearItConj}(r)$  //O(1)
  while ( $\text{haySiguiente}(\text{itCampos})$ ) do //O(1)
    if ( $\text{siguiente}(\text{itCampos}).idCampo == nroCampo$ ) then //O(1)
      valor  $\leftarrow \text{siguiente}(\text{itCampos}).valor$  //O(|v|)
    end if
    avanzar(itCampos) //O(1)
  end while
  return valor //O(1)

```

4. Modulo Tabla

El modulo Tabla provee operaciones básicas a realizarse sobre una tabla. Permite crear una tabla, agregar o eliminar un registro de una tabla, y brinda operaciones para obtener la clave de una tabla, obtener las columnas($\text{conj}(\text{nombreCampo})$) y las filas($\text{conj}(\text{Registro})$) de una tabla. Para describir la complejidad de las operaciones, el costo de copiar un $s \in \text{string}$ e igualar $s_1, s_2 \in \text{string}$ es igual a $O(|s|)$.

Interfaz

se explica con: TABLA

usa: REGISTRO, CONSULTA, BOOL, NAT, CONJUNTO LINEAL

género: tabla.

NUEVA TABLA(**in** $c: \text{conj}(\text{nombreTabla})$, **in** $nt: \text{nombreTabla}$) $\rightarrow res: \text{Tabla}$

Pre $\equiv \{nt \in c\}$

Post $\equiv \{res = \text{nueva}(t, nt)\}$

Complejidad: $O(1)$

Descripción: Crea una tabla a una base de datos.

Aliasing: No aplica

INSERTAR(**in/out** $t: \text{Tabla}$, **in** $r: \text{Registro}$)

Pre $\equiv \{t = t0 \wedge \text{campos}(t) = \text{campos}(r)\}$

Post $\equiv \{t = \text{insertar}(t0, r)\}$

Complejidad: $O()$

Descripción: Dados una tabla y un registro, inserta el registro en dicha tabla. En caso de que el registro ya exista en la tabla, lo sobrescribe

BORRAR(**in/out** $t: \text{Tabla}$, **in** $v: \text{Valor}$)

Pre $\equiv \{t = t0\}$

Post $\equiv \{(\neg \text{ValorEnTabla}(t, v) \wedge t=t0) \vee (\text{ValorEnTabla}(t, v) \wedge t=\text{borrar}(t, v))\}$

Complejidad: $O()$

Descripción: Dados una tabla y un valor, se fija si dicho valor coincide con alguno de los valores del campo clave de la tabla. En caso afirmativo borra el registro que contenga dicho valor, de lo contrario devuelve la tabla sin modificaciones.

CAMPOS(**in** $t: \text{Tabla}$) $\rightarrow res: \text{Conj}(\text{nombreCampo})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res = \text{campos}(t)\}$

Complejidad: $O()$

Descripción: Devuelve las columnas($\text{conj}(\text{nombreCampo})$) de la tabla.

REGISTROS(**in** $t: \text{Tabla}$) $\rightarrow res: \text{Conj}(\text{Registro})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res = \text{registros}(t)\}$

Complejidad: $O()$

Descripción: Devuelve las filas($\text{conj}(\text{Registro})$) de la tabla.

CLAVE(**in** $t: \text{Tabla}$) $\rightarrow res: \text{nombreCampo}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res = \text{clave}(t)\}$

Complejidad: $O()$

Descripción: Devuelve el nombre del campo declarado como clave de la tabla.

OBTENER NRO CAMPO(**in** $t: \text{Tabla}$ **in** $nc: \text{string}$) $\rightarrow res: \text{nat}$

Pre $\equiv \{(\text{Existe } r: \text{registro}) (r \text{ pertenece a } t.\text{registros}()) \text{ y Luego } nc \text{ pertenece a } r)\}$

Post $\equiv \{(\text{Para Todo } r: \text{registro}) (r \text{ pertenece a } t.\text{registros}() \text{ implica Luego } res = \text{ordenLexicografico}(r, nc))\}$

Complejidad: $O(|nc|)$

Descripción: Dados una tabla t y un nombreCampo c , devuelve el nat asociado al mapeo existente según un orden lexicografico que poseen los registros.

CAMPOS X CLAVE(**in** $t: \text{Tabla}$ **in** $valor: \text{string}$) $\rightarrow res: \text{puntero}(\text{registro})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{(\text{Existe } r: \text{registro}) (r \text{ pertenece a } t.\text{registros}() \text{ y } (\text{Existe } nc: \text{nombreCampo}) (\text{nombreCampo pertenece a } r.\text{campos} \text{ y Luego } r[nc] = v \text{ implica Luego } \text{puntero}(v) = res))\}$

Complejidad: $O(|valor|)$

Descripción: Dados una tabla t y un valor v , devuelve un puntero al registro perteneciente a la tabla cuyo valor de

su campo clave es v.

ValorEnTabla : Tabla \times Valor \rightarrow bool

ValorEnTabla(t, v) $\equiv (\exists r : Registro)(r \in \text{registros}(t) \wedge (\exists nc : NombreCampo)(nc = \text{clave}(t)) \wedge (r[nc] = v))$

Representación

Descripcion de representacion de tabla

Tabla se representa con estr

donde estr es tupla(*camposXClave*: DiccString(valorClave: string, reg: itConj),
registros: conj(registro),
idPorCampo: conj(Tupla<idCampo: nat, nc: string>),
campoClave: string)

Rep

- Todo idCampo que pertenece a alguna tupla perteneciente a e.idPorcampo tiene que ser corresponderse lexicograficamente al nc de la misma.
- #claves(e.camposXClave) = (e.registros)
- #(e.registros) = #(e.idPorCampo)
- $(\forall r : registro)(r \in e.registros \Rightarrow_L \text{campoClave} \in \text{claves}(r))$
- $(\exists t : \text{tupla} < \text{nat}, \text{string} >)(t \in e.idPorCampo \wedge_L \text{campoClave} = t.nc)$
- $(\forall vc : \text{string})(vc \in \text{claves}(e.camposXClave) \iff (\forall r : registro)(r \in e.registros \Rightarrow_L (r[\text{campoClave}] = vc)))$
- $(\forall t : \text{tupla} < \text{nat}, \text{string} >)(t \in e.idPorCampo \iff (\forall r : registro)(r \in e.registros \Rightarrow_L (t.nc \in \text{claves}(r))))$
- $(\forall vc : \text{string})(vc \in \text{claves}(e.camposXClave) \iff *obtener(e.obtener(e.camposXClave, vc) \in e.registros))$
- $(\forall r1 : registro)(r1 \in e.registros \Rightarrow_L (\forall r2 : registro)((r2 \in e.registros \wedge_L \neg(r1 = r2)) \Rightarrow_L \text{campos}(r1) = \text{campos}(r2)))$

Abs

Abs : estr e \rightarrow tabla

{Rep(e)}

Abs(e) $\equiv t$: tabla |
campos(t) =_{obs} campos(dameUno(e.registros))
clave(t) =_{obs} e.campoClave
registros(t) =_{obs} e.registros

Algoritmos

iNuevaTabla(in c : conj(nombreCampo), in $clave$: nombreCampo) $\rightarrow res$: tabla

1: $res \leftarrow \langle \text{diccString}::\text{Vacio}(), \text{conjunto}::\text{Vacio}(), \text{conjunto}::\text{Vacio}(), \text{clave} \rangle$

Complejidad: $O(||)$

iInsertar(in/out t : Tabla in r : Registro)

if Pertenece?(e.registros,r) then
eliminar(e.registros,r)
borrar(e.camposXClave, r[e.campoXClave])
end if

iBorrar(in t : tabla, in v : valor)

if (definido?(camposPorClave, v)) then
borrar(camposXClave, v) //O(|nombreCampo|)
eliminar(registros, v) //O(|nombreCampo|)
end if

iEsClaveTabla(in t : tabla, in nc : string) $\rightarrow res$: ?

return tabla::clave(t) == nombreCampo //O(|nombreCampo|)

```

iMergeReg(in  $r1$ : registro, in  $r2$ : registro)  $\rightarrow res : ?$ 
  regRes  $\leftarrow$  Nuevo() //O(1)
  itReg  $\leftarrow$  itConj(Campos(r2)) //O(1)
  while haySiguiente(itReg) do //O(1)
    registro::definir(r1, siguiente(itReg), r2[siguiente(itReg)]) //O(copy(campo) + O(1))
  end while
  return r1 //O(1)

```

```

iCampos(in  $t$ : tabla  $\rightarrow res$ : conj(nombreCampo)
   $res \leftarrow$  conjunto::vacio()
   $campos \leftarrow$  idPorCampo
  while (conj::haySiguiente(itCampos)) do
    conj::agregar(res, conj::siguiente(itCampos).nc)
    conj::avanzar(itCampos)
  end while
  return res

```

```

iRegistros(in  $t$ : tabla)  $\rightarrow res$ : conj(Registros)
1:  $res \leftarrow$  registros
Complejidad :O(1)

```

```

iClave(in  $t$ : tabla)  $\rightarrow res$ : conj(nombreCampo)
1:  $res \leftarrow$  campoPorClave
Complejidad :O(1)

```

```

iCamposPorClave(in  $t$ : tabla in  $v$ : string)  $\rightarrow res$ : puntero(Registro)
  res  $\leftarrow$  registro::nuevo()
  itReg  $\leftarrow$  diccString::obtener(tabla.camposXClave, v)
  if ( thenhaySiguiente(itReg))
    res  $\leftarrow$  siguiente(itReg) return res
  end if
  return &res

```

```

iObtenerNroCampo(in  $t$ : tabla, in  $nc$ : string)  $\rightarrow res$ : nat
  res  $\leftarrow$  1
  itTuplaIdPorCampo  $\leftarrow$  crearIt(tabla.idPorCampo)
  while haySiguiente(itTuplaIdPorCampo) do
    if siguiente(itTuplaIdPorCampo).nc = nc then
      res  $\leftarrow$  siguiente(itTuplaIdPorCampo).idCampo
    end if
    avanzar(itTuplaIdPorCampo)
  end while
  return res

```

5. Módulo Diccionario String(*string*, σ)

El módulo Diccionario String provee un diccionario básico en el que se puede definir, borrar, y testear si una clave está definida en tiempo $O(|k|)$ donde $|k|$ es la longitud de la clave.

Para describir la complejidad de las operaciones, el costo de copiar un $k \in \text{string}$ e igualar $k_1, k_2 \in \text{string}$ es igual a $O(|k|)$.

Por ultimo vamos a llamar $\text{copy}(s)$ al costo de copiar el elemento $s \in \sigma$ y $\text{equal}(s_1, s_2)$ al costo de evaluar si dos elementos $s_1, s_2 \in \sigma$ son iguales.

Interfaz

parámetros formales

géneros string, σ

<p>función $\bullet = \bullet(\text{in } k_1 : \text{string}, \text{in } k_2 : \text{string}) \rightarrow \text{res} : \text{bool}$</p> <p>bool</p> <p>Pre $\equiv \{\text{true}\}$</p> <p>Post $\equiv \{\text{res} =_{\text{obs}} (k_1 = k_2)\}$</p> <p>Complejidad: $O(k)$</p> <p>Descripción: función de igualdad de <i>string</i></p>	<p>función $\text{COPIAR}(\text{in } k : \text{string}) \rightarrow \text{res} : \text{string}$</p> <p>Pre $\equiv \{\text{true}\}$</p> <p>Post $\equiv \{\text{res} =_{\text{obs}} k\}$</p> <p>Complejidad: $O(k)$</p> <p>Descripción: función de copia de <i>string</i></p>
<p>función $\text{COPIAR}(\text{in } s : \sigma) \rightarrow \text{res} : \sigma$</p> <p>Pre $\equiv \{\text{true}\}$</p> <p>Post $\equiv \{\text{res} =_{\text{obs}} s\}$</p> <p>Complejidad: $\Theta(\text{copy}(s))$</p> <p>Descripción: función de copia de σ's</p>	

se explica con: $\text{DICCIONARIO}(\text{string}, \sigma)$

géneros: $\text{dicc}(\text{string}, \sigma)$.

$\text{VACÍO}() \rightarrow \text{res} : \text{dicc}(\text{string}, \sigma)$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{\text{res} =_{\text{obs}} \text{vacío}\}$

Complejidad: $O(1)$

Descripción: genera un diccionario vacío.

Aliasing: no aplica

$\text{DEFINIR}(\text{in/out } d : \text{dicc}(\text{string}, \sigma), \text{in } k : \text{string}, \text{in } s : \sigma)$

Pre $\equiv \{d =_{\text{obs}} d_0\}$

Post $\equiv \{d =_{\text{obs}} \text{definir}(d, k, s)\}$

Complejidad: $O(|k| + \text{copy}(s))$

Descripción: Define la clave k con el significado s en el diccionario. Si es que la clave ya existia se sobrescribe el significado

Aliasing: los elementos k y s se definen por copia.

$\text{DEFINIDO?}(\text{in } d : \text{dicc}(\text{string}, \sigma), \text{in } k : \text{string}) \rightarrow \text{res} : \text{bool}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{\text{res} =_{\text{obs}} \text{def?}(d, k)\}$

Complejidad: $O(|k|)$

Descripción: devuelve **true** si y sólo k está definido en el diccionario.

Aliasing: no aplica

$\text{OBTENER}(\text{in } d : \text{dicc}(\text{string}, \sigma), \text{in } k : \text{string}) \rightarrow \text{res} : \sigma$

Pre $\equiv \{\text{def?}(d, k)\}$

Post $\equiv \{\text{alias}(\text{res} =_{\text{obs}} \text{obtener}(d, k))\}$

Complejidad: $O(|k|)$

Descripción: devuelve el significado de la clave k en d .

Aliasing: res es modificable si y sólo si d es modificable.

$\text{BORRAR}(\text{in/out } d : \text{dicc}(\text{string}, \sigma), \text{in } k : \text{string})$

Pre $\equiv \{d = d_0 \wedge \text{def?}(d, k)\}$

Post $\equiv \{d =_{\text{obs}} \text{borrar}(d_0, k)\}$

Complejidad: $O(|k|)$

Descripción: elimina la clave k y su significado de d .

$\# \text{CLAVE}(\text{in } d : \text{dicc}(\text{string}, \sigma)) \rightarrow \text{res} : \text{nat}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \#claves(d)\}$

Complejidad: $O(1)$

Descripción: devuelve la cantidad de claves del diccionario.

CLAVES(in $d : \text{dicc}(string, \sigma) \rightarrow res : \text{conj}(string)$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} claves(d)\}$

Complejidad: $O(n * |k|)$, donde $n = \#claves(d)$

Descripción: devuelve el conjunto de claves del diccionario.

COPIAR(in $d : \text{dicc}(string, \sigma) \rightarrow res : \text{dicc}(string, \sigma)$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} d\}$

Complejidad: $O(n * (|k| + \text{copy}(\text{obtener}(k, d))))$, $\forall k \in claves(d)$, donde $n = \#claves(d)$

Descripción: genera una copia nueva del diccionario.