SQL

Introducción a las Bases de Datos

Informática Básica

SQL

Introducción a las Bases de Datos

Informática Básica

Introducción

- **SQL** es un lenguaje que permite el acceso a las funcionalidades de básicas de un **RDMBS** (Manejador de Bases de Datos Relacionales).
- Típicamente se puede usar desde una línea de comandos o desde otros lenguajes de programación como Java o C.
- Se van a estudiar las características fundamentales del Standard.

Sublenguajes

- **DDL**
  - Create Table
  - Alter Table
- **DML**
  - Insert
  - Delete
  - Update
- **QL**
  - Select
- También hay instrucciones para manejo de seguridad, definición de restricciones, etc.

SQL vs Cálculo de Tuplas

- Estructura de SQL

  De i en adelante, Ti no aparece en el **select** pero sí en el **from**

  Select $T_1.a_2, T_2.a_5, ...T_{i-1}.a_2$
  from $T_1, T_2, ...T_i, ...$
  where φ;

- En Cálculo

  $$\{< x_1.a_2, x_2.a_5, ..., x_{i-1}.a_2 > / T_1(x_1) \wedge T_2(x_2) \wedge ... \wedge T_i(x_i) \wedge \varphi\}$$

- Observar Que:
  - Sin importar la estructura de φ, la expresión en cálculo es segura porque todas las variables libres toman valores de las tablas involucradas.
  - Cualquier consulta se puede escribir de esta forma.

Ejemplos

- $Fabs(\#f, Nombre, Dir), Prods(\#p, Desc),$
  $Ventas(\#f, \#p, Precio).$
- Devolver los nombres de los fabricantes que venden el producto con nro 7848
- Cálculo de tuplas

$$\{t.nombre / Fabs(t) \wedge \exists v.(Ventas(v) \wedge v.\#p = 7848 \wedge v.\#f = t.\#f)\}$$

- SQL

```
Select nombre
from Fabs t
where exists (select *
  From ventas v
  Where v.#p=7848 and t.#f=v.#f );
```

```
Select nombre
from Fabs t, Ventas v
where #p=7848 and t.#f=v.#f;
```

Ejemplos

- Devolver los nombres de los vendedores que venden el producto 7848 y también el producto 8747
- Cálculo de tuplas

$$\{t.nombre/Fabs(t) \wedge \exists v.(Ventas(v) \wedge v.\#p = 7848 \wedge v.\#f = t.\#$$
$$\exists v1.(Ventas(v1) \wedge v1.\#f = t.\#f \wedge v1.\#p = 874$$

- SQL

```
Select nombre
from Fabs t, Ventas v
where #p=7848 and t.#f=v.#f and
exists( select *
    from Ventas v1
    where v1.#f=t.#f and v1.#p = 8747) );
```

Ejemplos

- Devolver los nombres de los vendedores que venden el producto 7848 y también el producto 8747
- SQL (Otra versión)

```
Select nombre
from Fabs t, Ventas v
where #p=7848 and t.#f=v.#f and
t.#f in ( select #f
          from Ventas v1
          where v1.#p = 8747) );
```

- SQL tiene los operadores $\in$ y $\notin$ (**in** y **not in**)

Ejemplos

- Devolver los nombres de los vendedores que no venden el producto 7848
- Cálculo de tuplas

$$\{t.nombre / Fabs(t) \wedge \neg \exists v.(Ventas(v) \wedge v.\#f = t.\#f \wedge v.\#p = 7848)\}$$

- SQL

```
Select nombre
from Fabs t
where not exists(
        select *
        from Ventas v1
        where v1.#f=t.#f and v1.#p = 7848 );
```

Ejemplos

- Devolver los nombres de los fabricantes que venden todos los productos con descripción "d1"
- Cálculo de tuplas

$$\{t.nombre/Fabs(t) \wedge \forall(p.(Prod(p) \wedge p.desc =' d1' \rightarrow \exists v1.(Ventas(v1) \wedge v1.\#f = t.\#f \wedge v1.\#p = p.\#p)) \wedge \exists p.(Prod(p) \wedge p.desc =' d1')\}$$

- SQL

```
Select f.nombre
From   fabs f
Where exists (select *
  from prod p
  where p.desc"="d1 ) and not exists (select *
  from prod p
  where p.desc"="d1
      and not exists( select *
              from venta v1
              where v1.#f=f.#f and v1.#p=p.#p) )
```

SQL como Lenguaje de Consultas

- SQL permite hacer operaciones que no son posibles con la versión de CR vista en el curso.
- El standard, actualmente permite:
  - recursividad
  - manejo temporal
  - consultas sobre tablas modificadas “al vuelo”.
  - Etc.
- También provee DDL y DML:
  - Ejs: Create table, insert.,delete, grant, etc.
- Cada proveedor hace lo que quiere, por lo que:
  - Sólo se verá la parte central de todo el lenguaje.
  - Luego hay que ir a los manuales.

SQL: DDL

- Create Table.

```
Create table <nombre> (
  <nomatt> <tipo> [[<restricción att>] [,… ]
  [<restricción tab> [, …] ]
);
```

- Ejemplos:

```
Create table Fabs(
  numF integer primary key,
  nombre varchar[128] not null,
  dir varchar[256]
);
Create table Prods(
  numP integer primary key,
  desc text
);
```

```
Create table ItemVta(
  numF integer references Fabs.Numf
  numP integer references Prods.Numl
  fecha date,
  cantV integer,
  precioUnit number(12,2),
  primary key (numF,numP,fecha)
)
```

Estructura General de una Consulta SQL

Select <Expresiones de Salida>
[**From** <Expresiones de Tabla>]
[**Order by** <Expresiones de Orden>]

- Donde:
  - **<Expresiones de Salida>**: es una lista de nombres de atributos o expresiones aritméticas.
  - **<Expresiones de Tabla>**: es un conjunto de indicaciones sobre cómo se construye una tabla de base para la consulta.
  - **<Expresiones de Orden>**: es una lista de nombres de columnas que figuran en el `select`, con una indicación `asc` o `desc` para cada una.

Expresiones de Salida

- Pueden ser:
  - Nombres de funciones.
    ```
    Select sysdate();
    ```
  - Nombres de atributos y/o expresiones aritméticas complejas.

    ```
    Select fecha, numP, F.nombre,
       precioUnit * cantV
    From ...
    ```
  - Funciones de agregación (Funciones Agregadas).

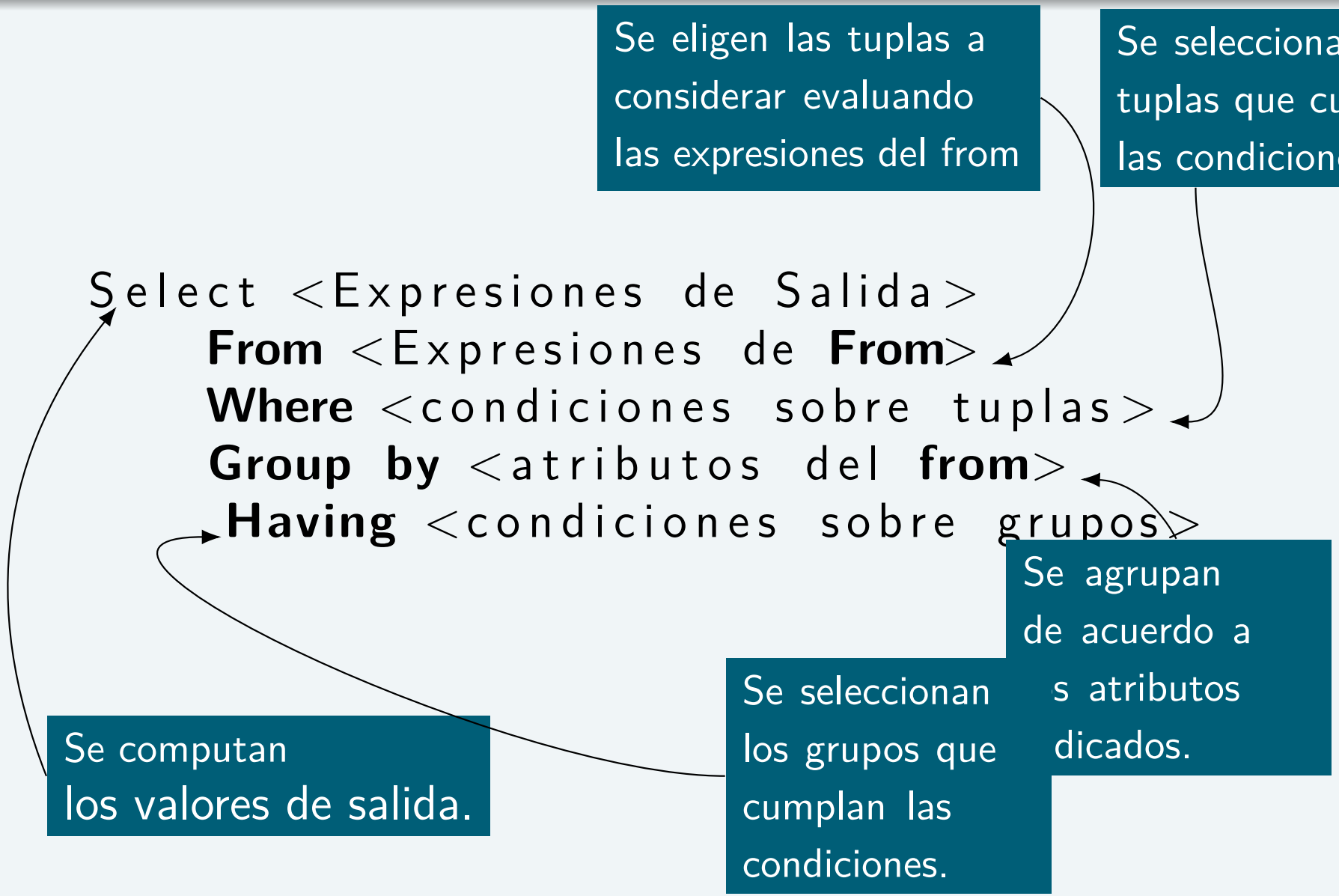
    ```
    Select count( * )
    From ...
    ```

Expresiones de Tablas

- Son las que describen la tabla (auxiliar) de la cual se toman valores en el resultado.
- Incluyen el contenido de la línea del **From** y las cláusulas **Where**, **Group By** y **Having**.
- De esta forma, la estructura de una consulta queda de la siguiente forma:

```
Select ...
 From <Expresiones de From>
 Where <condiciones sobre tuplas>
 Group by <atributos del from>
 Having <condiciones sobre grupos>
```

Ejecución de un Consulta



Expresiones en el From

- Las expresiones más simples que se pueden poner en **from** son listas de tablas separadas por ",". En este caso, calcula el producto cartesiano de las tablas.
- Además pueden aparecer expresiones de Join:
  - `<expr. de from> [<tipo>] join <expr. de from> <using o on>`
- El tipo de join puede ser:
  - `Natural`
  - `(Left|Right|Full) Outer`

Ejemplos: Natural Join

- La siguiente consulta:

```
Select *
from Prods natural join ItemVta
where numF=5
```

- es equivalente a:

```
Select Prods.*, ItemVta.numF, ItemVta.Fecha, ...
from Prods, ItemVta
where numF=5 and
      ItemVta.numP = Prods.numP
```

- **Natural Join** agrega la condición de igualdad para los atributos de igual nombre en las tablas indicadas.
- Sólo devuelve las tuplas que cumplen la condición y proyecta una vez por cada atributo

Ejemplos: Natural Join y Cálculo

- La siguiente consulta:

```
Select *
from Prods natural join ItemVta
where numF=5
```

- es equivalente a:

$$\{\langle p.numP, p.desc, i.numF, i.fecha, i.cantV, i.precioUnit\rangle / Prods(p) \wedge ItemVta(i) \wedge p.numP = i.nu$$

Ejemplos: Join Using y Join on

- La cláusula **Using** recibe la lista de atributos que deben participar en igualdades.

```
Select *
from Prods join ItemVta using (numP);
where numF=5
```

- La cláusula **On** recibe la condición de join que puede no ser una igualdad.

```
Select Prods.*, ItemVta.numF, ItemVta.Fecha, ...
from Prods join ItemVta
        on (ItemVta.numP = Prods.numP)
where numF=5
```

- **Using** proyecta una vez por cada atributo pero **On** no lo hace.

Outer Join

- Los casos anteriores, siempre devuelven las tuplas que cumplen la condición.
- Los **Outer Join** devuelve las que cumplen las condiciones y también las algunas de las que no las cumplen completadas con nulos.
- Los nombres (Left,Right y Full) se refieren al lado del que devuelven datos.
  - **Left Outer Join**: todas las tuplas de la tabla de la izquierda. Cuando la condición falla, completa la derecha con nulos.
  - **Right Outer Join**: todas las tuplas de la tabla de la derecha . Cuando la condición falla, completa la izquierda con nulos.

Ejemplos: Outer Join

Prods	
numP	**desc**
5	Tornillos 1/4“
4	Martillo

ItemVta		
numf	**numP**	**pUnit**
1	5	5
1	3	20

```
select *
from Prods as p left outer join ItemVta I
     using(numP);
```

Resultado			
numP	**desc**	**numf**	**pUnit**
5	Tornillos 1/4“	1	5
4	Martillo		

Funciones Agregadas

- En SQL se pueden usar diferentes funciones de agregación. Entre ellas:
  - Count(exp): cuenta la cantidad de tuplas del resultado.
  - Sum(exp): suma la expresión a través de todas las tuplas del resultado.
  - Avg(exp): hace el promedio de la expresión a través de todas las tuplas del resultado.
- Se colocan en la lista de salida (**select**) o en condiciones de selección de grupos (**having**)
- Devolver la cantidad de productos vendidos por el vendedor 1.

```
select numF, count(numP)
from ItemVta
where numF=1
```

Funciones Agregadas y **Distinct**

- El modificador **Distinct** hace que las funciones agregadas trabajen sobre valores en vez de trabajar sobre tuplas.
- Devolver la cantidad de productos **diferentes** vendidos por el vendedor 1.

```
select numF, count(distict numP)
from ItemVta
where numF=1
```

- Esto cuenta la cantidad de valores diferentes en el atributo **numP** y no la cantidad de tuplas.

Group By y Having

- La lista de atributos en el **Group By** indica sobre qué atributos se deben agrupar las tuplas de la tabla base. El resultado de la consulta es una tupla por grupo.
- Las funciones de agregación se computan sobre cada grupo.
- Hay que garantizar que los valores de salida (**Select**) son únicos para cada grupo.
- La condición **Having** selecciona los grupos que quedan en el resultado.

Ejemplos: Group By y Having

ItemVta		
numf	**numP**	**pUnit**
1	5	5
1	3	20
2	3	19
2	4	20
2	5	10

Para cada vendedor obtener la cantidad de productos que vende.

```
select numF, count( * )
from ItemVta
group by numF
```

ItemVta	
numf	**count(*)**
1	2
2	3

Ejemplos: Group By y Having

- Para cada vendedor que vende productos con promedio de precios unitarios mayor que 14, obtener la cantidad de productos que vende

```
select numF, count( * )
from ItemVta
group by numF
having avg( pUnit ) > 14
```