

# Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas (TUDAI)

## Base de Datos

### Tema 4: Consultas SQL - Parte 1

2  
0  
2  
5

# Lenguaje de Consulta SQL

- ✓ La sentencia del lenguaje empleada para la recuperación de los datos a partir de las tablas cargadas en la base de datos es el **SELECT**.
- ✓ La sentencia básica es:

```
SELECT * | { [DISTINCT] columna | expresión [alias],...}  
FROM <lista tablas>
```

- ✓ En una consulta se especifica qué información se requiere, sin especificar cómo obtenerla (no requiere métodos de acceso a los datos).

# Escritura de Sentencia SQL

- ✓ Las cláusulas de la sentencias suelen colocarse en líneas separadas y con sangría, para mejorar la legibilidad
- ✓ Se estila escribir con MAYÚSCULAS las palabras reservadas y con minúsculas el resto
- ✓ Por ejemplo:

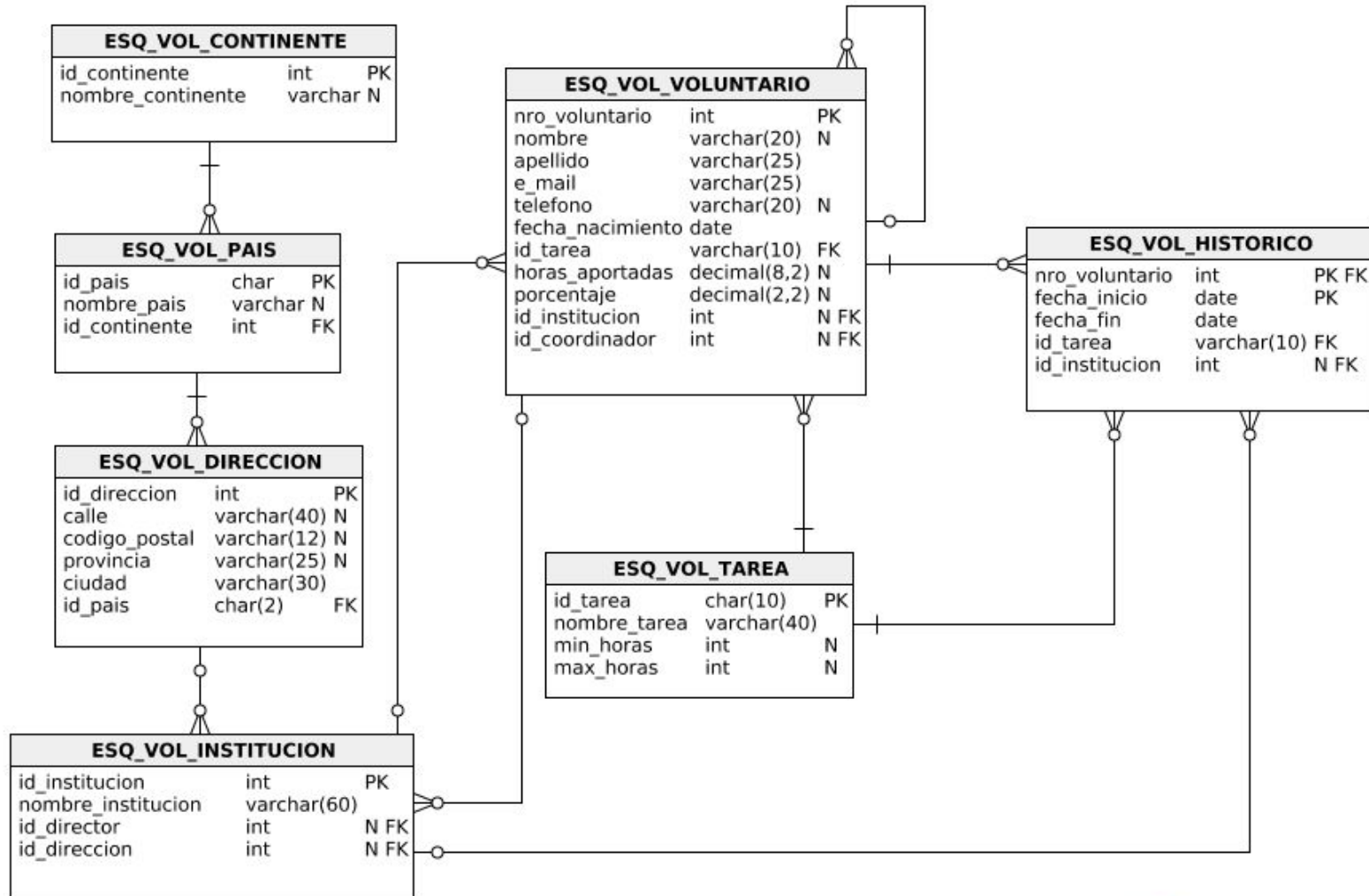
```
SELECT mi_atributo  
FROM mi_tabla;
```

# Lenguaje de Consulta SQL

```
SELECT * | { [DISTINCT] columna | expresión [alias],...}  
FROM <lista tablas>
```

- ✓ **SELECT** identifica las columnas a recuperar – **EL QUE**
- ✓ **FROM** identifica la tabla - **DE DONDE** obtener los datos
- ✓ El resultado de una consulta es una ***tabla*** (si es un número, se considera como una tabla con una fila y una columna).

# DERE del Esquema de Voluntarios



# Consultas SQL Básicas

Selección de todas las columnas.

Ejemplo: selección de los datos completos de las instituciones.

```
SELECT *  
FROM esq_vol_institucion;
```

Se usa para especificar la recuperación de todos los datos de la/s tabla/s

nombre_institucion	id_director	id_direccion	id_institucion
CASA DE LA PROVIDENCIA	200	1700	10
CORPORACION URRACAS DE EMAUS	201	1800	20
FUNDACION CIVITAS	114	1700	30
FUNDACION LAS ROSAS DE AYUDA FRATERNA	203	2400	40
FUNDACION HOGAR DE CRISTO	121	1500	50
FUNDACION MI CASA	103	1400	60
CORPORACION SOLIDARIDAD Y DESARROLLO	204	2700	70
FUNDACION REGAZO	145	2500	80
FUNDACION ALERTA BOSQUES	100	1700	90
BOSQUEDUCA	108	1700	100
COMITE NACIONAL PRO DEFENSA DE LA FLORA Y LA FAUNA	205	1700	110
CONSEJO ECOLOGICO COMUNAL	NULL	1700	120
CORPORACION AMBIENTAL	NULL	1700	130
FUNDACION VIDA RURAL	NULL	1700	140
CENTRO DE AYUDA MAPUCHE	NULL	1700	150
SIERRAS PROTEGIDAS	NULL	1700	160

27 fila(s)  
Tiempo total de ejecución: en milisegundos  
SQL ejecutada.

# Consultas SQL Básicas

Selección SÓLO de algunas columnas.

Ejemplo: seleccionar el código y el nombre de las instituciones.

```
SELECT id_institucion, nombre_institucion  
FROM esq_vol_institucion;
```

id_institucion	nombre_institucion
10	CASA DE LA PROVIDENCIA
20	CORPORACION URRACAS DE EMAUS
30	FUNDACION CIVITAS
40	FUNDACION LAS ROSAS DE AYUDA FRATERNA
50	FUNDACION HOGAR DE CRISTO
60	FUNDACION MI CASA
70	CORPORACION SOLIDARIDAD Y DESARROLLO
80	FUNDACION REGAZO
90	FUNDACION ALERTA BOSQUES
100	BOSQUEDUCA

# Filas Duplicadas

Por defecto, ante una consulta, se recuperan todas las filas, **incluidas** las filas duplicadas.

Ejemplo: Seleccionar los voluntarios que son coordinadores

```
SELECT id_coordinador  
FROM esq_vol_voluntario;
```



id_coordinador	
NULL	
100	
100	
102	
103	
103	
103	
103	
101	
108	
108	
108	
108	
108	
100	
114	
114	
114	
114	
114	
100	
100	
100	
100	
...	
201	
205	
NULL	

Para eliminar los valores repetidos se debe usar la cláusula **DISTINCT**

Ejemplo: Seleccionar los distintos voluntarios que son coordinadores

```
SELECT DISTINCT id_coordinador  
FROM esq_vol_voluntario;
```





# Eliminar Valores Repetidos

La cláusula **DISTINCT** se aplica a todas las columnas de la lista en el **SELECT**

Ejemplo: Seleccionar los voluntarios coordinadores y las distintas instituciones de los empleados coordinados

```
SELECT DISTINCT id_institucion, id_coordinador  
FROM esq_vol_voluntario;
```

id_institucion	id_coordinador
10	101
20	100
20	201
30	100
30	114
40	101
50	100
50	120
50	121
50	122
50	123
50	124
60	102
60	103
70	101
80	100
80	145
80	146
80	147
80	148
80	149
90	100
90	NULL

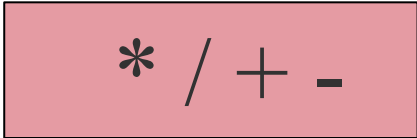
# Ejercicios 1

1. ¿Cuántas Instituciones tiene la tabla Institución y cuales son?
2. Comparar los resultados de:
  1. Distintos Identificadores de Instituciones en las que trabajan los voluntarios
  2. Distintos coordinadores
  3. Distintos identificadores de coordinadores con sus instituciones

# Uso de operadores Aritméticos

- ✓ Una expresión aritmética puede contener nombres de columnas, valores numéricos constantes y operadores aritméticos

## Precedencia de los Operadores



\* / + -

- ✓ La multiplicación y la división tienen prioridad sobre la suma y la resta
- ✓ Los operadores de idéntica prioridad se evalúan de izquierda a derecha
- ✓ Los paréntesis se utilizan para forzar evaluaciones prioritarias y para clarificar las sentencias

# Uso de operadores Aritméticos

*Ejemplo: Seleccionar los nombres de las tareas y los rangos de duración (diferencia entre horas máximas y mínimas) de cada una.*

```
SELECT nombre_tarea, max_horas – min_horas  
FROM tarea;
```

*¿Que obtendremos en estos casos?*

```
SELECT nombre_tarea, 2*max_horas – min_horas  
FROM tarea;
```

```
SELECT nombre_tarea, 2*(max_horas – min_horas)  
FROM tarea;
```

*Los paréntesis se usan para forzar evaluaciones prioritarias y para clarificar sentencias!!*

# Renombrado de Columnas

- ✓ El alias **AS** renombra un encabezamiento de columna o tabla.
- ✓ Si contiene espacios, caracteres especiales o es sensible a mayúsculas y minúsculas, se debe encerrar entre comillas dobles.

```
SELECT nombre_tarea, max_horas – min_horas AS variacion  
FROM tarea;
```

```
SELECT nombre_tarea, max_horas – min_horas variacion  
FROM tarea;
```

- ✓ Operador de concatenación || para cadenas de caracteres

```
SELECT apellido || ‘,’ || nombre AS “Apellido,nombre”  
FROM voluntario;
```

# Ejercicios 2

1. Listar todos los datos de las tareas y la diferencia entre las hs máxima y mínima de la misma.
2. Listar las direcciones de manera que se vea la calle, la ciudad y la provincia separadas por comas (,).

# Restringir las Filas Recuperadas

- ✓ La cláusula **WHERE** se usa para realizar las restricciones.
- ✓ Una cláusula **WHERE** contiene una condición lógica, la cual usa
  - operadores de comparación (<, >, =, <=, >=, <> o !=)
  - operadores lógicos (**AND**, **OR**, **NOT**).
- ✓ Las filas recuperadas son aquellas cuyos datos que satisfacen la/s condición/es lógicas

```
SELECT * | { [DISTINCT] columna | expresión [alias],...}  
FROM <lista tablas>  
[WHERE condición/es];
```

# Restringir las Filas Recuperadas

Por ejemplo recuperar el nro de voluntario, nombre y apellido de los voluntarios que trabajan en la institución cuyo identificador es 60.

```
SELECT nro_voluntario, nombre, apellido  
FROM voluntario  
WHERE id_institucion = 60;
```

## Resultado de la consulta

nro_voluntario	nombre	apellido
103	Alexander	Hunold
104	Bruce	Ernst
105	David	Austin
106	Valli	Pataballa
107	Diana	Lorentz



# ¿Cómo resuelve el DBMS una Query ?

Los pasos que sigue el DBMS son los siguientes:

- 1) Resolver el FROM (saber sobre que va operar) (el FROM)**
- 2) Filtrar ese DataSet o Conjunto de Datos (el WHERE)**
- 3) Proyectar (el select)**

# Condiciones de Comparación

- ✓ Los operadores de comparación se utilizan en la cláusula **WHERE** para comparar expresiones.
- ✓ El resultado de la comparación puede ser
  - Verdadero (T)
  - Falso (F)
  - Desconocido (U)
- ✓ Tener presente que si se comparan valores nulos usando los operadores de comparación el resultado será siempre FALSO porque un valor nulo no puede ser igual, mayor, distinto, etc. a otro valor.

```
SELECT *  
FROM voluntario  
WHERE id_tarea= 'ST_MAN';
```

```
SELECT *  
FROM voluntario  
WHERE id_tarea != 'ST_MAN';
```

# Otros Operadores de Comparación

- ✓ Además de los operadores de comparación está disponible un operador especial **[NOT] BETWEEN** :
- ✓ **BETWEEN** trata a los valores de los extremos incluidos dentro del rango.
  - **BETWEEN x AND y** es equivalente a  $a \geq x$  **AND**  $a \leq y$
  - **NOT BETWEEN x AND y** es equivalente a  $a < x$  **OR**  $a > y$

Ejemplo: Seleccionar los voluntarios cuyo número se encuentra entre 100 y 120

```
SELECT * FROM voluntario  
WHERE nro_voluntario BETWEEN 100 AND 120;
```

# Operador LIKE

- ✓ No siempre se conoce el valor exacto a buscar.
- ✓ Se puede buscar coincidencias con un patrón de caracteres mediante el operador **LIKE**. También se emplea en la forma negativa **NOT LIKE**.
- ✓ Comodines
  - %: cualquier secuencia de cero o más caracteres
  - \_ : denota un solo carácter

Ejemplo: Seleccionar los voluntarios cuya segunda letra del nombre sea a y luego tenga una n como carácter final.

```
SELECT * FROM voluntario WHERE nombre LIKE '_a%n';
```

nombre	apellido	e_mail	telefono	fecha_nacimiento	id_tarea	nro_voluntario	horas_aportadas
Karen	Colmenares	KCOLMENA	515.127.4566	1999-08-10	PU_CLERK	119	2500.00
Jason	Mallin	JMALLIN	650.127.1934	1996-06-14	ST_CLERK	133	3300.00
Karen	Partners	KPARTNER	011.44.1344.467268	1997-01-05	SA_MAN	146	13500.00
Harrison	Bloom	HBLOOM	011.44.1343.829268	1998-03-23	SA_REP	169	10000.00

# Ejercicios 3

1. ¿Cuáles son los voluntarios nacidos antes de la década del '90?
2. ¿Cuáles son los voluntarios con nombre David?
3. ¿Cuáles son los voluntarios con apellido Smith?

# Operador IS [NOT] NULL

- ✓ Si una columna en particular carece de un valor se dice que contiene un **NULL**.
- ✓ **NULL** es un valor inaccesible, sin valor, desconocido o inaplicable.
- ✓ No representa ni un cero ni un espacio en blanco (el cero es un número y el espacio en blanco es un carácter). SOLO testean valores que son nulos.

Ejemplo: **listar los datos completos de los voluntarios que no tengan coordinador.**

```
SELECT * FROM voluntario  
WHERE id_coordinador IS NULL;
```

nombre	apellido	e_mail	telefono	fecha_nacimiento	id_tarea	nro_voluntario	horas_aportadas	porcentaje
Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	100	24000.00	NULL

# Operador IS [NOT] NULL

- ✓ Si se comparan valores nulos usando los otros operadores (=, >, etc.) el resultado será siempre DESCONOCIDO porque un valor nulo no puede ser igual, mayor, menor o distinto a otro valor.
- ✓ Si se desea incluir en el resultado los datos de aquellas columnas que tengan nulos hay que hacerlo explícitamente.

# Operador IS [NOT] NULL

Ejemplo listar los datos de los voluntarios cuyo porcentaje sea menor o igual que 0,10.

- ✓ Algunos porcentajes pueden ser nulos
- ✓ Si deseo incluirlos en el resultado debo explicitar IS NULL.

```
SELECT * FROM voluntario  
WHERE porcentaje <= 0.1  
      OR porcentaje IS NULL;
```



# Condiciones de Comparación Compuestas

- ✓ Un operador lógico combina los resultados de dos condiciones para producir un único resultado basado en ellos, o invertir el resultado de una condición.
- ✓ Los operadores **AND** y **OR** se pueden usar para componer expresiones lógicas.

# Lógica Trivaluada

A	B	A OR B	A AND B
T	T	T	T
T	U	T	U
T	F	T	F
U	T	T	U
U	U	U	U
U	F	U	F
F	T	T	F
F	U	U	F
F	F	F	F

El operador **AND** retorna VERDADERO si ambas condiciones evaluadas son VERDADERAS

El operador **OR** retorna VERDADERO si alguna de las condiciones es VERDADERA.

El operador **NOT** invierte el resultado de la expresión.

# Condiciones de Comparación Compuestas

**Ejemplo: Seleccionar los voluntarios que son coordinados por los voluntarios nro 100 y 124 que están trabajando para la institución cuyo código es 50.**

```
SELECT nro_voluntario,  
       apellido,  
       id_institucion,  
       id_coordinador  
FROM voluntario  
WHERE (id_coordinador=100  
       OR id_coordinador=124)  
       AND id_institucion=50;
```

nro_voluntario	apellido	id_institucion	id_coordinador
120	Weiss	50	100
121	Fripp	50	100
122	Kaufling	50	100
123	Vollman	50	100
124	Mourgos	50	100
141	Rajs	50	124
142	Davies	50	124
143	Matos	50	124
144	Vargas	50	124
196	Walsh	50	124
197	Feeney	50	124
198	OConnell	50	124
199	Grant	50	124

**Importante:**

- Indentar las cláusulas
- Uso de paréntesis en las condiciones

¿Qué sucede si se eliminan ()?

# ...Al eliminar el ()

nro_voluntario	apellido	id_institucion	id_coordinador
101	Kochhar	90	100
102	De Haan	90	100
114	Raphaely	30	100
120	Weiss	50	100
121	Fripp	50	100
122	Kaufling	50	100
123	Vollman	50	100
124	Mourgos	50	100
141	Rajs	50	124
142	Davies	50	124
143	Matos	50	124
144	Vargas	50	124
145	Russell	80	100
146	Partners	80	100
147	Errazuriz	80	100
148	Cambrault	80	100
149	Zlotkey	80	100
196	Walsh	50	124
197	Feeney	50	124
198	OConnell	50	124
199	Grant	50	124
201	Hartstein	20	100

Estas filas no corresponden a la institución cuyo identificador es 50!!  
No es válida la consulta sin los paréntesis en la condición!!!

# Ejercicios 4

1. ¿Cuáles son los voluntarios nacidos entre 1988 y 1995?
2. ¿Cuáles son los voluntarios con nombre David o con apellido Smith y que realicen la tarea SA\_REP?

# Orden de Presentación de los Registros

- El orden de las filas listadas en una consulta es indefinido.
- **ORDER BY** puede usarse para ordenar las filas, y se debe colocar como última cláusula de la sentencia **SELECT**.
- Si no se especifica, el orden de los datos por defecto cuando se declara el **ORDER BY** es ascendente (**ASC**), pero se puede especificar también **DESC**. Se debe colocar después del nombre de la columna.

# Orden de Presentación de los Registros

Se puede ordenar el resultado de una consulta por más de una columna (pueden ser todas las de la tabla)

Ejemplo      listar      los      apellidos      ordenados  
descendentemente y nombres de los voluntarios que  
son coordinados por el voluntario 124.

```
SELECT apellido, nombre  
FROM voluntario  
WHERE id_coordinador=124  
ORDER BY apellido DESC, nombre;
```

apellido	nombre
Walsh	Alana
Vargas	Peter
Rajs	Trenna
OConnell	Donald
Matos	Randall
Grant	Douglas
Feeney	Kevin
Davies	Curtis

# LIMIT y OFFSET (POSTGRESQL)

- Permiten recuperar solamente un subconjunto de filas del total de la consulta.

```
SELECT lista de atributos  
FROM tabla/s  
[ORDER BY ... ]  
[LIMIT { número | ALL }]  
[OFFSET número];
```

- Debería ser usado siempre con la cláusula **ORDER BY**.
- La cláusula **LIMIT** limita la cantidad de filas a retornar.
- La cláusula **OFFSET** determina a partir de qué fila del resultado se retorna.



# LIMIT y OFFSET (POSTGRESQL)

Ejemplo: seleccionar los datos de los voluntarios que corresponden a los 10 primeros voluntarios.

```
SELECT *  
FROM voluntario  
ORDER BY nro_voluntario  
LIMIT 10;
```

# LIMIT y OFFSET (POSTGRESQL)

Ejemplo: seleccionar los datos de los voluntarios a partir del 15<sup>TO</sup> voluntario.

```
SELECT *  
FROM voluntario  
ORDER BY nro_voluntario  
LIMIT ALL  
OFFSET 15;
```

# Ejercicios 5

1. ¿Cuáles son los 10 voluntarios mayores?
2. En orden alfabético ¿quiénes son los 5 primeros voluntarios de la institución 80?

# ¿Qué son Funciones de Grupo o Agregación?

Estas funciones operan sobre conjuntos de filas para proporcionar un resultado por grupo.

nro_voluntario	apellido	id_institucion	coordinador
121	Fripp	50	100
196	Walsh	50	124
147	Errazuriz	80	100
103	Hunold	60	102
115	Khoo	30	114
185	Bull	50	121
158	McEwen	80	146
175	Hutton	80	149
167	Banda	80	147
187	Cabrio	50	121
193	Everett	50	123
104	Ernst	60	103
179	Johnson	80	149
153	Olsen	80	145
162	Vishney	80	147
142	Davies	50	124
109	Faviet	100	108
163	Greene	80	147
105	Austin	60	103
165	Lee	80	147

Cantidad de  
horas aportadas  
por todos los  
voluntarios

total_de_horas_aportadas
691400.00

# Funciones de Grupo

Permiten resumir el resultado de una consulta:

- **SUM( )** → *sumatoria de la columna especificada*
- **AVG( )** → *promedio de la columna especificada*
- **STDDEV()** → *desvío estándar de la columna especificada*
- **MAX( )** → *valor máximo de la columna especificada*
- **MIN ( )** → *valor mínimo de la columna especificada*
- **COUNT ( )** → *cantidad de tuplas*

```
SELECT [columna, ...] función de grupo(columna), ...  
FROM tabla/s  
[WHERE condición/es]
```

# Funciones de Grupo

- **AVG**, **SUM** y **STDDEV** se usan para datos numéricos.

```
SELECT SUM(horas_aportadas),  
       AVG(horas_aportadas),  
       MAX(horas_aportadas),  
       MIN(horas_aportadas)  
FROM voluntario;
```

- **MIN** y **MAX** se pueden usar para cualquier tipo de dato  
Ejemplo: seleccionar el voluntario más joven y el más viejo.

```
SELECT MAX(fecha_nacimiento) AS voluntario_mas_joven,  
       MIN(fecha_nacimiento) AS voluntario_mas_viejo  
FROM voluntario;
```

# Funciones de Grupo

**COUNT(\*)** devuelve el número de filas de una tabla.

```
SELECT COUNT(*)  
FROM voluntario;
```

**COUNT(expr)** devuelve el número de filas con valores no nulos para expr.

Ejemplo: Liste el número de ciudades en la tabla dirección, excluyendo los valores nulos.

```
SELECT COUNT(ciudad) AS cantidad__de_ciudades  
FROM direccion;
```

# Funciones de Grupo y Valores Nulos

Las funciones de grupo ignoran los valores nulos del atributo

```
SELECT  AVG(porcentaje) AS  
        porcentaje_promedio  
FROM voluntario;
```

porcentaje_promedio
0.22285714285714285714



# Funciones de Grupo y Valores Nulos

La función **COALESCE**(columna, valor\_reemplazo) en POSTGRESQL fuerzan a las funciones de grupo a que incluyan valores nulos, retornando un valor en ocurrencia de un nulo.

```
SELECT  AVG(COALESCE(porcentaje, 0)) AS  
        Porcentaje_promedio  
FROM    voluntario;
```

porcentaje_promedio
0.07289719626168224299

# Ejemplos de Funciones de Grupo

```
SELECT COUNT(*) AS  
    cantidad_de_voluntarios  
FROM voluntario;
```

cantidad_de_voluntarios
107

Alias de  
columna

```
SELECT SUM(horas_aportadas) AS  
    Horas_trabajadas  
FROM voluntario v WHERE  
    v.id_coordinador=120;
```

horas_trabajadas
22100.00

Alias de  
tabla


```
SELECT MAX(horas_aportadas) maximo,  
    MIN(horas_aportadas) minimo,  
    MAX(horas_aportadas) – MIN(horas_aportadas)  
    diferencia  
FROM voluntario v  
WHERE v.id_coordinador=120;
```

maximo	minimo	diferencia
3200.00	2200.00	1000.00

# Selección sobre Grupos de Datos

- ✓ Si se usa la cláusula **GROUP BY** en una sentencia **SELECT**, se dividen las filas de la tabla consultada en grupos
- ✓ Se aplican las funciones en la lista **SELECT** a cada grupo de filas y retorna una única fila por cada grupo.

```
SELECT lista columnas función de grupo (columna)
FROM <lista tablas>
[ WHERE condición ]
[ GROUP BY expresión de grupo | lista columnas
[ ORDER BY <lista atributos orden> ];
```



*La cláusula **GROUP BY** especifica cómo se deben agrupar las filas seleccionadas.*

# Creación de Grupos de Datos

nro_voluntario	apellido	id_institucion	coordinador
121	Fripp	50	100
196	Walsh	50	124
147	Errazuriz	80	100
103	Hunold	60	102
115	Khoo	30	114
185	Bull	50	121
158	McEwen	80	146
175	Hutton	80	149
167	Banda	80	147
187	Cabrio	50	121
193	Everett	50	123
104	Ernst	60	103
179	Johnson	80	149
153	Olsen	80	145
162	Vishney	80	147
142	Davies	50	124
109	Faviet	100	108
163	Greene	80	147
105	Austin	60	103
165	Lee	80	147

```
SELECT id_institucion, COUNT(*) AS  
      cantidad_voluntarios  
FROM voluntario  
GROUP BY id_institucion;
```

¿Cuántos  
voluntarios  
tiene cada  
Institución?

id_institucion	cantidad_voluntarios
90	3
NULL	1
20	2
100	6
40	1
110	2
80	34
70	1
50	45
60	5
30	6
10	1

# Sintaxis de la Cláusula **GROUP BY**

***Todas las columnas de la lista **SELECT**, excepto las funciones de grupo, deben estar en la cláusula **GROUP BY*****

Ejemplo: liste las diferentes instituciones y el máximo de horas aportadas a cada una de ellas

```
SELECT id_institucion, MAX(horas_aportadas)
FROM voluntario
GROUP BY id_institucion;
```

# Sintaxis de la Cláusula **GROUP BY**

Las columnas en la cláusula **GROUP BY** pueden no estar en la lista del **SELECT**

Ejemplo: determine los porcentajes promedio de los voluntarios por institución.

```
SELECT  AVG(porcentaje)
FROM    voluntario
GROUP BY id_institucion;
```

# Ejercicios 6

1. ¿Cuántos voluntarios realizan cada tarea?
2. ¿Cuál es el promedio de horas aportadas por tarea?

# Restringir los Resultados de los Grupos

Se puede anexar la cláusula **HAVING** para restringir grupos

- Las filas se agrupan por la/s columnas especificada/s
- Se aplica la función de grupo
- Se muestran los grupos que satisfacen la cláusula **HAVING**

nro_voluntario	apellido	id_institucion	coordinador
121	Fripp	50	100
196	Walsh	50	124
147	Errazuriz	80	100
103	Hunold	60	102
115	Khoo	30	114
185	Bull	50	121
158	McEwen	80	146
175	Hutton	80	149
167	Banda	80	147
187	Cabrio	50	121
193	Everett	50	123
104	Ernst	60	103
179	Johnson	80	149
153	Olsen	80	145
162	Vishney	80	147
142	Davies	50	124
109	Faviet	100	108
163	Greene	80	147
105	Austin	60	103
165	Lee	80	147

```
SELECT id_coordinador, COUNT(*) AS  
       cantidad_de_voluntarios  
FROM voluntario  
GROUP BY id_coordinador  
HAVING COUNT(*) > 7;
```

Coordinadores con  
más de 7 voluntarios

id_coordinador	cantidad_voluntarios
122	8
120	8
100	14
124	8
121	8
123	8



# Funciones de Grupo NO Válidas

- ✓ No se puede utilizar la cláusula **WHERE** para restringir grupos
- ✓ Se debe utilizar la cláusula **HAVING** para restringir grupos. No se pueden utilizar funciones de grupo en la cláusula **WHERE**

```
SELECT id_coordinador, COUNT(*)  
      AS cantidad_de_voluntarios  
FROM voluntario  
WHERE COUNT(*) > 7  
GROUP BY id_coordinador;
```

Sentencia **NO VALIDA**

## Error de SQL:

ERROR: aggregates not allowed in WHERE clause at character 105

## En la declaración:

```
SELECT id_coordinador, count(*) as Cantidad_voluntarios  
  
FROM unc_esq_voluntario.voluntario where count(*) > 7  
group by id_coordinador
```

;

# Ejercicios 7

1. ¿Cuáles son las tareas que tienen más de 10 voluntarios?
2. ¿Cuál es el promedio de horas aportadas por tarea solo de aquellos voluntarios nacidos a partir del año 2000?

# Ejercicios 8

1. ¿Cuáles son las tareas cuyo promedio de horas aportadas por tarea de los voluntarios nacidos a partir del año 1995 es superior al promedio general de dicho grupo de voluntarios?

# Para Recordar!!!!

- La sentencia SQL empleada para la recuperación de los datos a partir de las tablas cargadas en la base de datos es el **SELECT**
- **SELECT** identifica las columnas a recuperar – EL QUE
- **FROM** identifica la/s tabla/s - DE DONDE obtener los datos
- **WHERE** se usa para realizar las restricciones sobre los datos (filtrar)
- Para eliminar los valores repetidos se debe usar la cláusula **DISTINCT**

# Para Recordar!!!!

- Los operadores de comparación se utilizan en la cláusula WHERE para comparar expresiones. Usar paréntesis y sangrías para mejorar la legibilidad.
- **ORDER BY** puede usarse para ordenar las filas, y se debe colocar como última cláusula de la sentencia SELECT
- Las funciones de agregación operan sobre conjuntos de filas para proporcionar un resultado por grupo.
- **GROUP BY** *especifica como se deben agrupar las filas seleccionadas. Todas las columnas de la lista SELECT, excepto las funciones de grupo, deben estar en la cláusula GROUP BY.* **No** se pueden utilizar funciones de grupo en la cláusula **WHERE**.

# Recomendaciones

- Completar con textos de la Bibliografía (*en todos se explica SQL y se plantean ejemplos, incluso en algunos -Date por ej.- hay ejercicios y soluciones para chequear*)
- Consultar en Internet (*sitios confiables*)
- Consultar la Documentación del SQL estándar y de PostgreSQL



# Recomendaciones

- Resolver ejercicios (*propuestos en la práctica, laboratorio, etc.*)
- Probar ejercicios en PostgreSQL (*accesible vía web... en la página está la URL y los datos para conexión*), aprovechar las prácticas de laboratorio!!
- *Practicar...*
  - *Practicar...*
    - *Practicar... !!!*

