

# Lenguaje SQL (Structured Query Language)

UNIDAD 7: Lenguaje SQL en PostgreSQL

# SQL

2

Tipos de Operaciones que permite especificar

1. **Definición de Datos (DDL)**
2. **Manipulación de Datos (DML): Consultas y Actualizaciones**
3. **Definición de vistas**
4. **Definición de restricciones de integridad**
5. **Especificación de aspectos de seguridad y autorización**
6. Especificación de control de transacciones
7. Reglas para inclusión en lenguajes (C, PASCAL, etc.)

# SQL: DDL y DML

3

## DDL

- Incluye sentencias que permiten definir los objetos de la Base de Datos
- Las sentencias DDL generan cambios en el diccionario de datos, el cual contiene los metadatos

## DML

- Es un lenguaje que permite a los usuarios acceder o manipular los datos de la base de datos

# SQL

4

## DDL

- ☐ Create
- ☐ Alter
- ☐ Drop

## DML

- ☐ Select
- ☐ Insert
- ☐ Update
- ☐ Delete

# 1 - DDL

5

## CREATE TABLE

```
CREATE TABLE nombre_tabla
( nombre_columna      tipo_dato/dominio [NOT NULL]
  [DEFAULT valor] [CHECK (condicion)],
  [nombre_columna      tipo_dato/dominio [NOT NULL]
  [DEFAULT valor], ...],

  [ PRIMARY KEY (nombre_columna [, nombre_columna,...]), ]

  [ FOREIGN KEY (nombre_columna [, nombre_columna,...]),
  REFERENCES nombre_tabla (nombre_columna [,
                           nombre_columna,...]) ]

);
```

# 1 – DDL (Integridad Referencial)

6

## Integridad referencial

- **<definición de restricción referencial> ::=**  
**FOREIGN KEY** (<lista columnas referenciantes> ) <esp.de ref>
- <esp. de ref> ::= REFERENCES <nombre tabla referenciada>  
[( <column.ref.> ) ] [ <acc. ref> ]
- <acc.ref> ::=  
{ <regla de modificación> [ <regla de borrado> ]  
| <regla de borrado> [ <regla de modificación> ] }
- <regla de modificación> ::=  
ON UPDATE { CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION }
- <regla de borrado> ::=  
ON DELETE { CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION }

# 1 - DDL

7

## DROP TABLE

- DROP TABLE nombre\_tabla [RESTRICT/CASCADE];
  - ▣ RESTRICT: Elimina la tabla solo si no esta referenciada.
  - ▣ CASCADE: Elimina la tabla y la restriccion.

# 1 - DDL

8

## ALTER TABLE

- ALTER TABLE nombre\_tabla  
ADD nombre\_columna  
tipo\_dato/dominio  
[DEFAULT valor];
- ALTER TABLE nombre\_tabla  
DROP nombre\_columna;



# 2 - DML

9

Lenguaje de  
Manipulación  
del SQL

- DML implementa (al menos) las Operaciones del Algebra Relacional
- Funcionalidad

DML “=” Algebra Relacional



Multiconjunto o  
Bolsa



Conjunto



Provoca diferencias importantes entre ambos

## 2 - DML

10

SELECT

(Forma  
general)

```
SELECT lista_atributos  
FROM nombre_tabla/s  
[WHERE condición]  
[GROUP BY lista_atributos  
  [HAVING condición]]  
[ORDER BY lista_atributos];
```

Donde las tablas especificadas en la clausula FROM pueden ser:

- Tablas almacenadas (es decir, tablas base creadas por la sentencia *create table*)
- Tablas temporales (es decir, filas devueltas por un suquery)
- Tablas virtuales (es decir, creadas por un *create view*)

## 2 - DML: Tablas Ejemplo

11

Tabla DEPT

Tabla DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

4 filas

# 2 - DML: Tablas Ejemplo

12

Tabla  
DEPT

## Tabla EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	800		20
7499	ALLEN	SALESMAN	7698	20/02/81	1600	1400	30
7521	WARD	SALESMAN	7698	22/02/81	1250		30
7566	JONES	MANAGER	7839	02/04/81	2975		20
7654	MARTIN	SALESMAN	7698	28/09/81	1250		30
7698	BLAKE	MANAGER	7839	01/05/81	2850		30
7782	CLARK	ANALYST	7839	09/06/81	2450	0	10
7788	SCOTT	PRESIDENT	7566	19/04/87	3000		20
7839	KING	SALESMAN		17/11/81	5000		10
7844	TURNER	CLERK	7698	08/09/81	1500		30
7876	ADAMS	CLERK	7788	23/05/87	1100		20
7900	JAMES	ANALYST	7698	03/12/81	950		30
7902	FORD	CLERK	7566	03/12/81	3000		20
7934	MILLER		7782	23/01/82	1300		10

14 filas

## 2 - DML: Proyección (Atributos)

13

*Número y nombre de los empleados.*

**Algebra**

$\pi_{\text{epno, ename}} \text{ EMP}$

**SQL**

```
SELECT epno, ename  
FROM EMP;
```

## 2 - DML: Proyección (Atributos)

14

*Trabajos (Jobs) realizados.*

**Algebra**

$\pi_{\text{job}} \text{ EMP}$

**SQL**

```
SELECT job  
FROM EMP;
```

```
SELECT distinct job  
FROM EMP;
```

## 2 - DML: Proyección (Todos los atributos)

15

*Todos los atributos de todos los empleados.*

**Algebra**

EMP

**SQL**

```
SELECT *  
FROM EMP;
```

Proyecta todos los  
atributos de la tabla

## 2 - DML: Proyección (Atributo renombrado)

16

*Nombre (ename) de todos los empleados (mostrado como nombre) .*

**Algebra**

$\rho_{\text{nombre}} (\pi_{\text{ename}} \text{ EMP } )$

**SQL**

```
SELECT ename AS  
       nombre  
FROM EMP;
```



## 2- DML: Proyección

17

### Proyección

A diferencia del Algebra, SQL permite proyectar:

- Atributos
- **Constantes**
- **Expresiones**
- **Funciones**

## 2 - DML: Proyección

18

Expresiones y  
Constantes

*Nombre (ename) de todos los empleados junto al salario quincenal.*

```
SELECT ename as nombre, sal/2 as salario_quincena  
FROM EMP;
```

*Nombre (ename) de todos los empleados junto al salario quincenal.*

```
SELECT ename as nombre,  
       ' cobra quincenalmente ',  
       sal/2 as salario_quincena,  
       ' pesos'  
FROM EMP;
```

## 2 - DML: Proyección

19

Funciones de  
cadenas de  
caracteres

*Los tres primeros caracteres del nombre (ename) de todos los empleados.*

```
❑ SELECT substr(ename,1,3)
      FROM emp;
```

*La concatenación del nombre y job de todos los empleados.*

```
❑ SELECT concat(ename, job)
      FROM emp;
```

# 2 - DML: Proyección con Funciones

20

## Funciones de Agregacion

### □ COUNT

- ▣ SELECT Count(nombre\_atributo/distinct nombre\_atributo/\*)  
FROM nombre\_tabla;

### □ SUM

- ▣ SELECT Sum(nombre\_atributo)  
FROM nombre\_tabla;

### □ AVG

- ▣ SELECT Avg(nombre\_atributo)  
FROM nombre\_tabla;

### □ MAX (MIN)

- ▣ SELECT Max(nombre\_atributo)  
FROM nombre\_tabla;

**NOTA:** Todas estas funciones se llaman funciones de RESUMEN o de AGREGACION

## 2 - DML: Proyección

21

### Funciones de resumen

*Cantidad de empleados.*

```
❑ SELECT count(*)  
FROM emp;
```

*Monto total pagado en sueldos.*

```
❑ SELECT sum(sal)  
FROM emp
```

*Monto promedio de sueldos.*

```
❑ SELECT avg(sal)  
FROM emp
```

## 2 - DML: Restricciones de las Funciones de Agregación

22

### Funciones de Agregación

- No se pueden componer funciones agregadas, por ej. **max(avg())**
- Para toda consulta que utiliza alguna función de agregación, cada elemento de la lista del SELECT debe ser:
  - ▣ Una función agregada, o
  - ▣ Un atributo presente en la lista de atributos del GROUP BY ([Se vera mas adelante](#)).
- Los valores NULL son ignorados por cualquier función de agregación con excepción de count(\*).
- Los valores NULL son tratados como un valor convencional en los agrupamientos por atributos ([Se vera mas adelante](#)).

## 2 - DML: Alias

23

Alias

```
SELECT lista_atributos  
FROM nombre_tabla nombre_alias  
.....;
```

*Nro. y nombre de empleados (alias aux).*

```
select aux.empno, aux.ename  
from emp aux;
```

```
select aux.empno, aux.ename  
from emp as aux;
```

## 2 - DML: Selección

24

*Nro. y nombre de los empleados que trabajan en el departamento 20.*

### Algebra

$\pi_{\text{empno,ename}}(\sigma_{\text{deptno}=20} \text{ EMP})$

### SQL

```
SELECT empno, ename  
FROM EMP  
WHERE DEPTNO=20;
```

*Nro. y nombre de los empleados que trabajan en el departamento 20 y tienen un salario menor a 1000.*

$\pi_{\text{empno,ename}}$   
 $(\sigma_{\text{deptno}=20 \text{ and } \text{sal}<1000} \text{ EMP})$

```
SELECT empno, ename  
FROM EMP  
WHERE DEPTNO=20 and  
SAL<1000;
```



## 2- DML: Selección

25

### Otros Operadores de comparación

Además de los operadores utilizados en el Algebra (<, >, =, and, etc.) SQL provee los siguientes:

#### 1. Operadores de valores nulos

- <columna> IS [NOT] NULL

#### 2. Operador LIKE

- <columna> LIKE <patrón>

#### 3. Operadores de condiciones de dominios

- <columna> [NOT] BETWEEN <valor inferior> AND <valor superior>

#### 4. Operadores de comparación de conjuntos

- <columna> [NOT] IN (<lista de valores>)
- [NOT] EXISTS (<lista de valores>)

## 2 - DML: Selección (Nulos)

26

Condiciones  
con nulos

### 1. Operadores de valores nulos

El valor NULL no es una constante, no se puede utilizar explícitamente como un operando.

```
SELECT empno, ename, comm
FROM EMP
WHERE comm is null;
```

El valor nulo no es igual ni distinto a ningún valor

```
SELECT empno, ename, comm
FROM EMP
WHERE comm <> 300;
```

```
SELECT empno, ename, comm
FROM EMP
WHERE comm = 300;
```

## 2 - DML: Selección (Nulos)

27

Condiciones  
con nulos

Los valores nulos pueden representar semánticamente:

- ▣ **Valor desconocido:** No disponible por el momento. Por ej. Si se desconoce la fecha de nacimiento.
- ▣ **Valor inaplicable:** Ningún valor tiene sentido. Ej. Un atributo “ nombre cónyuge ” para un soltero/a.
- ▣ **Valor retenido:** No se dispone de la suficiente jerarquía para conocer el valor. Por ej. Un número de teléfono, pin, etc.

## 2 - DML: Selección (Nulos)

28

### Condiciones con nulos

Operando con valores nulos:

1. Operadores aritméticos (+,\*,etc.): Cuando al menos uno de los operandos es null, el resultado de la operación es **null**.
2. Operadores relacionales (<,>=,etc.): Cuando al menos uno de los operandos es null, el resultado de la operación es **unknown (desconocido)**. El valor **DESCONOCIDO** es otro valor de verdad.

Por ejemplo, si x tiene valor NULL:

- $x + 3$  es NULL
- $NULL + 3$  no es válido (No se puede usar como una constante)
- $x = 3$  es UNKNOWN
- $NULL = 3$  no es válido (No se puede usar como una constante)

## 2 - Valor de Verdad DESCONOCIDO

Dado que las comparaciones expresadas en la cláusula WHERE pueden ser combinadas usando los operadores lógicos AND, OR y NOT...

Analicemos el resultado que se obtiene cuando alguno/s de las comparaciones devuelve el valor de verdad DESCONOCIDO (UNKNOWN )

## 2 – Operadores Lógicos con el Valor de Verdad DESCONOCIDO

Tabla de Verdad para los operadores OR, AND y NOT

P	Q	P OR Q
V	V	V
V	NULL	V
V	F	V
F	V	V
F	NULL	NULL
F	F	F
NULL	V	V
NULL	NULL	NULL
NULL	F	NULL

P	Q	P AND Q
V	V	V
V	NULL	NULL
V	F	F
F	V	F
F	NULL	F
F	F	F
NULL	V	NULL
NULL	NULL	NULL
NULL	F	F

P	NOT P
V	F
NULL	NULL
F	V

## 2 – Operadores Lógicos con el Valor de Verdad DESCONOCIDO

### Tabla de Verdad para el Operador OR

P	Q	P OR Q
V	V	V
V	NULL	V
V	F	V
F	V	V
F	NULL	NULL
F	F	F
NULL	V	V
NULL	NULL	NULL
NULL	F	NULL

Analicemos nuevamente el resultado de P or Q, dándole los siguientes valores numéricos a los valores de verdad:

$V = 1$

$F = 0$

$D = 0,5$

$A \text{ or } B = \max\{A, B\}$



## 2 – Operadores Logicos con el Valor de Verdad DESCONOCIDO

### Tabla de Verdad para el Operador AND

P	Q	P AND Q
V	V	V
V	NULL	NULL
V	F	F
F	V	F
F	NULL	F
F	F	F
NULL	V	NULL
NULL	NULL	NULL
NULL	F	F

Analicemos nuevamente el resultado de P and Q, dándole los siguientes valores numéricos a los valores de verdad:

**V = 1**

**F = 0**

**D = 0,5**

**Entonces...**

**A and B = min{A,B}**





## 2 – Operadores Logicos con el Valor de Verdad DESCONOCIDO

### Tabla de Verdad para el Operador NOT

P	NOT P
V	F
NULL	NULL
F	V

Analicemos nuevamente el resultado de Not P, dándole los siguientes valores numéricos a los valores de verdad:

$$V = 1$$

$$F = 0$$

$$D = 0,5$$

$$\text{Not } A = 1 - A$$



# 2 - DML: Selección

34

Condiciones  
con operador  
el “like “

## 2. Operador LIKE

*Todos los datos de los empleados cuyo nombre comienza con la letra A.*

```
SELECT *  
FROM scott.emp  
WHERE ename LIKE 'A%';
```

*El operador **like** compara parte de una cadena de caracteres, donde:*

- *% sustituye a una cantidad arbitraria de caracteres*
- *\_ sustituye a un solo carácter*

## 2 - DML: Selección

35

Condiciones  
con operador  
de rango

### 3. Operador de rango

*Todos los datos de los empleados cuyo salario oscila entre 1000 y 2000.*

SELECT \*

FROM scott.emp

WHERE salary BETWEEN 1000 and 2000;

## 2 - DML: Selección

36

Condiciones  
comparación  
con conjuntos

### 4. Operadores de comparación con conjuntos

*Todos los datos de los empleados que trabajan en el departamento 20 y/o 30.*

```
Select *  
FROM scott.emp  
WHERE deptno IN (20,30);
```

## 2 - DML: Producto Cartesiano

37

Algebra

EMP  $\times$  DEPT

SQL

SELECT \*  
FROM EMP,DEPT;

## 2 – DML: Join

38

join

[join] :: =

nom\_tabla1 NATURAL JOIN nom\_tabla2 /

nom\_tabla1 JOIN nom\_tabla2 ON condición /

nom\_tabla1 INNER JOIN nom\_tabla2 ON condición /

nom\_tabla1 LEFT [OUTER] JOIN nom\_tabla2 ON  
condición /

nom\_tabla1 RIGHT [OUTER] JOIN nom\_tabla2 ON  
condición

## 2 - DML: Natural Join

39

*Nombre, trabajo que realiza y nombre del departamento donde trabaja, de todos los empleados.*

### Algebra

$\pi_{ename, job, dname} (EMP \bowtie DEPT)$

### SQL

```
SELECT ename, job, dname  
FROM EMP NATURAL JOIN  
DEPT;
```

(Los atributos sobre los cuales se hace el JOIN deben tener el mismo nombre en ambas tablas)

## 2 - DML: Natural Join

40

Natural Join  
construido con  
producto  
cartesiano  
Algebra

```
SELECT epno,ename, job, mgr, hiredate, sal, comm,  
       deptno, dname, loc  
FROM   EMP, DEPT  
WHERE  EMP.deptno= DEPT.deptno;
```



## 2 - DML: Join

41

### Algebra

$\pi_{ename, job, dname}$   
 $(EMP \bowtie_{EMP.deptno = DEPT.deptno} DEPT)$

### SQL

- `SELECT ename, job, dname  
FROM EMP JOIN DEPT ON  
EMP.deptno = DEPT.deptno;`
- `SELECT ename, job, dname  
FROM EMP INNER JOIN  
DEPT ON EMP.deptno =  
DEPT.deptno;`

## 2 - DML: Join

42

*Nombre, trabajo que realiza y nombre del departamento donde trabaja, de todos los empleados del departamento 10.*

Join con  
restricción  
extra

```
SELECT ename, job, dname  
FROM EMP JOIN DEPT ON EMP.deptno =  
      DEPT.deptno  
WHERE EMP.deptno = 10;
```

## 2 - DML: Outer Join

43

Outer Join

- *LEFT OUTER JOIN*
- *RIGHT OUTER JOIN*
- *FULL OUTER JOIN*

## 2 - DML: Outer Join

44

### Left Outer Join

- `SELECT deptno,dname, ename, job`  
`FROM DEPT NATURAL JOIN EMP;`  
*El departamento 40 no sale informado porque no existe ningún empleado asignado a él*
- `SELECT DEPT.deptno,dname, ename, job`  
`FROM DEPT LEFT OUTER JOIN EMP ON`  
`DEPT.DEPTNO=EMP.DEPTNO;`  
*El departamento 40 sale informado a pesar de que no existen empleados asignados a él*

## 2 - DML: Outer Join

45

### Right Outer Join

- `SELECT deptno,dname, ename, job`  
`FROM EMP NATURAL JOIN DEPT;`

*El departamento 40 no sale informado porque no existe ningún empleado asignado a él.*

- `SELECT ename, job, dept.deptno,dname`  
`FROM EMP RIGHT OUTER JOIN DEPT ON`  
`DEPT.DEPTNO=EMP.DEPTNO;`

*El departamento 40 sale informado a pesar de que no existen empleados asignados a él*

## 2 - DML: Outer Join

46

Full Outer Join

```
SELECT lista_atributos  
FROM nombre_tabla FULL OUTER JOIN  
    nombre_tabla  
ON condición;
```

## 2 - DML: Union

47

*Nombre de empleados que trabajan en los departamentos 20 y/o 30.*

### Algebra

- $\pi_{\text{ename}} (\sigma_{\text{deptno}=20 \text{ or } \text{deptno}=30} \text{EMP})$
- $\pi_{\text{ename}} (\sigma_{\text{deptno}=20} \text{EMP}) \cup \pi_{\text{ename}} (\sigma_{\text{deptno}=30} \text{EMP})$

### SQL

- ```
SELECT ename FROM EMP
WHERE deptno=20
or deptno=30 ;
```
- ```
SELECT ename FROM EMP
WHERE deptno=20
UNION
SELECT ename
FROM EMP
WHERE deptno=30;
```

## 2 - DML: Union

48

### Union

- En este caso, si existen tuplas repetidas se eliminan

### Union All

- En este caso, si existen tuplas repetidas no se eliminan



## 2 - DML: Intersección

49

*Departamentos que tienen administrativos y analistas entre sus empleados (CLERK y ANALYST)*

### Algebra

$\pi \text{ deptno}(\sigma \text{ job}=\text{'CLERK'} \text{ EMP})$

$\cap$

$\pi \text{ deptno}(\sigma \text{ job}=\text{'ANALYST'} \text{ EMP})$

### SQL

Select deptno  
from emp  
where job='CLERK'

**INTERSECT**

Select deptno  
from emp  
where job='ANALYST';

## 2 - DML: Minus

50

*Departamentos que no tienen empleados*

**Algebra**

$\pi_{\text{deptno}} \text{Dept} - \pi_{\text{deptno}} \text{EMP}$

**SQL**

```
select deptno from dept  
minus  
select deptno from emp;
```

## 2 - DML: División

51

### Algebra

$\pi_{x,y}A \text{ DIVIDE BY } \pi_yB$

### SQL

- La operación DIVIDE no tiene un operador equivalente en SQL, pero puede expresarse como un doble NOT EXISTS
- Se analizará mas adelante, cuando se explique el operador NOT EXISTS

```
Select A.x
from A
where not exists
    (select *
     from B
     where not exists
         (select *
          from A AX
          where AX.x=A.x
            and      AX.y=B.y));
```

52

## Subconsultas

# Subconsultas

53

Existen diferentes tipos de subconsultas, clasificadas según diferentes criterios (algunas de ellas):

- En base al lugar donde se encuentra:
  - En la cláusula FROM
  - En la cláusula WHERE o HAVING
- En base al resultado que obtienen:
  - Valores escalares
  - Conjuntos

## 2 – DML: Subconsultas Clausula FROM

54

Subconsultas en la clausula FROM

*Nombre del depto. en el que trabaja SCOTT.*

```
select DNAME
from dept, (select *
            from emp
            where ename='SCOTT') AUX
where dept.DEPTNO=AUX.DEPTNO;
```

```
select DNAME
from dept natural join (select *
                       from emp
                       where ename='SCOTT');
```

- Se puede observar en la cláusula FROM una subconsulta en lugar de una tabla base.
- Dado que no tiene nombre si se necesita referenciar alguna atributo, deberá colocarse un alias (en este caso, AUX).

## 2 – DML: Subconsultas en WHERE

55

Operadores  
de  
comparación  
para consultas  
más complejas

### Operador IN/ NOT IN

- <expresión> IN (<subconsulta>)
  - ▣ En este caso <expresión> es comparada con el conjunto que devuelve la subconsulta:
    - El resultado es Verdadero si el elemento de la expresion esta dentro del conjunto devuelto por la subconsulta. Y Falso si el elemento no se encuentra en el.
    - Los atributos colocados en <expresion> deben coincidir con los atributos de la tabla que genera la subconsulta.
- <expresión> NOT IN (<subconsulta>)

## 2 – DML: Subconsultas en WHERE

56

Operadores  
de  
comparación  
para consultas  
más complejas

Operador =, <, > con subconsultas

□ Ej: <expresión> = (<subconsulta>)

■ En este caso <expresión> es comparada con el VALOR que devuelve la subconsulta



## 2 – DML: Subconsultas en WHERE

57

Operadores  
de  
comparación  
para consultas  
más complejas

### Operador =, <, > con subconsultas: Uso de ANY y ALL

- **<expresión>** <operador de comparación> [**ANY/ALL**] (<subconsulta>)
- <expresión> puede ser una columna o un valor computado
- En este caso <expresión> es comparada con cada valor seleccionado por la subconsulta
  - **Any:** Evalúa verdadero si existe al menos una tupla seleccionada en el subquery que satisfaga la condición. Si el subquery da vacío, la condición es falsa.
  - **All:** Evalúa verdadero si todas las tuplas seleccionadas por el subquery satisfacen la comparación. Si el subquery es vacío, la condición es verdadera.

## 2 - DML: Subconsultas en WHERE

58

Operadores  
de  
comparación  
para consultas  
más complejas

*Mostrar los empleados que ganan lo mismo que el empleado nro. 7902.*

```
SELECT *  
FROM emp  
WHERE EMPNO<>7902  
AND sal =  
  (SELECT sal  
   FROM emp  
   WHERE EMPNO=7902);
```

## 2 - DML: Subconsultas en WHERE

59

Operadores  
de  
comparación  
para consultas  
más complejas

*Mostrar todos los empleados que trabajan en el depto 10 y que ganan al menos tanto como algún empleado del departamento 30*

```
SELECT *  
FROM emp  
WHERE deptno=10  
and sal >= any  
  (SELECT sal  
   FROM emp  
   WHERE deptno=30);
```

## 2 - DML: Subconsultas en WHERE

60

Operadores  
de  
comparación  
para consultas  
más complejas



*Mostrar todos los empleados que no trabajan en el depto 30 y que ganan más que todos los empleados que trabajan en el departamento 30*

```
SELECT *  
FROM semp  
WHERE deptno <> 30  
and sal >= all  
  (SELECT sal  
   FROM emp  
   WHERE deptno=30);
```

## 2 – DML: Subconsultas

61

Equivalencias  
de  
operadores  
para consultas  
más complejas  
(con  
subqueries)

- IN  = ANY
- NOT IN  <> ALL

## 2 – DML: Subconsultas en WHERE

62

Operadores  
de  
comparación  
para consultas  
más complejas

### Operador EXISTS/ NOT EXISTS

- where [NOT] EXISTS (<subconsulta>)
  - ▣ En este caso, se evalúa el conjunto que devuelve la subconsulta:
    - El resultado es Verdadero si el conjunto generado es no vacío. Es Falso si el conjunto es vacío.
- where NOT EXISTS (<subconsulta>)

## 2 - DML

63

### Subconsultas

- Algunas operaciones del Algebra también se pueden implementar con subconsultas y operadores de conjunto

## 2 - DML: Intersección

64

Intersección  
con subquery

*Mostrar departamentos que tienen empleados que trabajan como 'CLERK' y 'ANALYST'.*

```
SELECT dept.deptno, dname
from dept
where deptno in
    (Select deptno
     from emp
     where job='CLERK')
and deptno in
    (Select deptno
     from emp
     where job='ANALYST');
```



## 2 - DML: Minus

65

Minus con  
subquery

*Mostrar departamentos que no tienen empleados.*

- ```
select deptno
from dept
where deptno not in
      (select deptno
       from emp);
```
- ```
select deptno
from dept
where not exists
      (select *
       from emp
       where dept.deptno=emp.deptno);
```

## 2 - DML: Exists vs In

66

### Exists vs. In

- ❑ Las subconsultas vinculadas con IN pueden seleccionar más de un atributo, obviamente compatible con los atributos expresados en el where.
- ❑ Todas las consultas resueltas con IN pueden ser resueltas con EXISTS
- ❑ Idem para los negados (NOT IN – NOT EXISTS)

## 2 - DML: División

67

### Algebra

$(\pi_{x,y} A) / (\pi_y B)$

### SQL

```
Select A.x  
from A  
where not exists  
  (select *  
   from B  
   where not exists  
     (select *  
      from A aux  
      where aux.x=A.x  
            and aux.y=B.y));
```

## 2 - DML: División (Tomemos el ejemplo de la BD de deportes)

68

*Deportes (todos los datos) practicado en todos los clubes.*

$$\left( (\pi_{codd, codc} \text{ **prac** } / (\pi_{codc} \text{ **club**})) \text{ NJ } \text{ **depo** } \right)$$

select \*  
from **depo** ←  
where not exists  
(select \*  
from **club** ←  
where not exists  
(select \*  
from **prac** ←  
where **prac.codd=depo.codd**  
and **prac.codc=club.codc**));

## 2 - DML: Grupos

69

Group BY

SELECT ....

....

[GROUP BY lista\_atributos];

- Agrupa las tuplas que tienen igual valor del/de los atributos especificados en la lista\_atributos.
- Muestra una tupla representativa del grupo (resume el grupo). Este valor debe ser único al grupo y además si es un atributo, debe estar especificado en la cláusula group by.

## 2 - DML: Grupos

70

Group by

*Departamentos (nro.) junto a su cantidad de empleados.*

- ❑ `select deptno, count(*)  
from emp  
group by deptno;` (genera grupo con valor nulo)
  
- ❑ `select deptno, count(*)  
from emp natural join dept  
group by deptno;` (no genera el grupo con valor nulo)

## 2 - DML: Grupos

71

Group by

*Departamentos (nro.) junto al sueldo mayor dentro de él.*

- ❑ `select deptno, max(sal)`  
`from emp`  
`group by deptno;` (genera grupo con valor nulo)
  
- ❑ `select deptno, max(sal)`  
`from emp natural join dept`  
`group by deptno;` (no genera el grupo con valor nulo)

## 2 - DML: Grupos

72

Group y  
Having

SELECT ....  
[GROUP BY lista\_atributos  
[HAVING condición]];

- Agrupa las tuplas que tienen igual valor del/ de los atributos especificados en lista\_atributos.
- Muestra una tupla representativa del grupo (resume el grupo). Este valor debe ser único al grupo y además si es un atributo, debe estar especificado en la cláusula group by.
- El HAVING permite seleccionar los grupos que cumplen la condición indicada



## 2 - DML: Grupos

73

Group y  
Having

*Departamentos (nro.) junto a la cantidad de empleados, para aquellos que superen los 4 empleados.*

```
select deptno, count(*)  
from emp natural join dept  
group by deptno  
having count(*)>4;
```

## 2 – DML: Ordenamiento

74

Ordenamiento

SELECT ....

....

[ORDER BY lista\_atributos ASC/DESC];

Ordena la salida (tabla resultante)  
según el/los atributos especificados

## 2 – DML: Ordenamiento

75

### Ordenamiento

*Empleados (nro., nombre, job y salario) ordenado segun nombre, luego por job, y por ultimo por ambos.*

- `select empno, ename, job, sal`

`from emp`

`order by ename`

- `select empno, ename, job, sal`

`from emp`

`order by job`

- `select empno, ename, job, sal`

`from emp`

`order by job, ename`

# Ejemplo de agrupamiento, having, ordenamiento y limit (subquery en un having)

76

Consulta con  
group, having,  
order y limit

*Departamento que tiene mas empleados (nro. y cantidad)*

```
□ select deptno, count(*)  
   from emp natural join dept  
  group by deptno  
 having count(*)=  
        (select count(*)  
         from emp natural join dept  
        group by deptno  
       order by 1 desc  
       limit 1);
```

*La clausula **limit x**, se usa para quedarse con x cantidad de tuplas del resultado obtenido. En este caso se queda con una, y como está ordenada descendetemente es la que tiene el mayor valor.*

## 2 – DML: Actualización

77

Actualización

- Inserción de tuplas
- Modificación de tuplas
- Eliminación de tuplas

## 2 – DML: Inserción

78

Insert con  
values

```
INSERT INTO <tabla> [<column_i, ..., column_j>]  
VALUES (<valor_i, ..., valor_j>);
```

- Para cada columna debe ser especificado un valor.
- Si no se especifica una columna de la tabla, se almacenará en ella un valor nulo.
- Si no se especifican las columnas se asumirá el orden especificado en el create.

# 3 – DML: Inserción

79

Insert con  
subquery

```
INSERT INTO <tabla> [<column_i, ..., column_j>]  
<query>;
```

Ejemplo:

```
insert into hr.jobs  
select *  
from otrohr.jobs;
```

## 2 – DML: Modificación

80

### Update

```
UPDATE <tabla>  
SET <column_i>=<expr_i>, ..., <expr_j> = <expr_j>  
[WHERE <condición>];
```

Donde:

<expr> puede ser una constante, una expresión aritmética o un query

Ejemplo:

```
UPDATE EMP  
SET sal = (select min(sal) from EMP where  
          job='MANAGER')  
WHERE job='SALESMAN' and deptno=20;
```



## 2 – DML: Modificación

81

Update

```
UPDATE <tabla>  
SET (<columna i, ..., columna j>) = <query>  
[WHERE <condición>];
```

## 2 – DML: Eliminación

82

DELETE

DELETE FROM <tabla> [WHERE <condición>];

# 3 - Vistas

83

## Definición de Vistas

- Una vista es definida como una consulta sobre una o más relaciones, guardadas con un nombre en el diccionario de datos
- ¿Por qué usar vistas?
  - ▣ Ocultar información a algunos usuarios
  - ▣ Hacer más simples algunas consultas
  - ▣ Modularidad en el acceso a la base de datos

# 3 - Vistas

84

## Creación de Vistas

```
CREATE VIEW <nombre de la vista> [<columna(s)>] AS  
<sentencia select> [ WITH CHECK OPTION ];
```

- Si [<columna(s)>] no está especificado, las columnas toman el mismo nombre de los correspondientes a la tabla generada en la sentencia select.
- Una vista es evaluada cada vez que es accedida.
- La definición de una vista puede usar otra vista.
- La cláusula WITH CHECK OPTION provoca que si se intenta insertar en la vista una tupla que no satisface la condición de la vista, esa inserción es rechazada. Los Updates son similarmente rechazados si el nuevo valor no satisface la condición.

# 3 - Vistas

85

## Uso de Vistas

- Una vista puede ser usada en una sentencia SELECT en forma idéntica que una tabla.
- No toda vista puede ser actualizada automáticamente (insert, update, delete).
- La dificultad de la actualización de vistas radica en que éstas deben ser traducidas a actualizaciones sobre las tablas bases sobre las que está definida la vista (*create trigger ... instead of ...* )

# 3 – Actualización usando Vistas

86

Condiciones para que puedan **actualizarse datos** a través de una vista de manera automática

- La cláusula from debe usar una sola tabla
- La clausula select contiene sólo nombres de atributos, es decir, no tiene:
  - ▣ expresiones
  - ▣ funciones de agregación
  - ▣ especificación de distinct
- No debe incluir la cláusula group by en su definición
- No se ha incluido en la clausula select algún atributo que en la tabla base tiene la restricción de not null

# 4 – Restricciones de Integridad

87

## Tipos

1. **NOT NULL** - Indica que la columna no puede contener valores nulos
2. **UNIQUE** - Asegura que el valor de una columna/s debe ser único en toda la tabla
3. **PRIMARY KEY** - Asegura la Integridad de Entidad
4. **FOREIGN KEY** - Asegura la Integridad Referencial
5. **CHECK** - Asegura que se satisfaga la condición especificada
6. **DEFAULT** - Especifica un valor por defecto

# 4 – Restricciones de Integridad

88

## Tipos

(En cuanto a la forma de definirse)

Una restricción puede definirse sintácticamente de dos formas:

- Como parte de la definición de una columna
  - El nombre de la restricción es asignada automáticamente por el motor de base de datos.
- Como parte de la definición de la tabla.
  - El nombre de la restricción puede ser especificado por el usuario.



# 4 – Restricciones de Integridad

89

## Comprobación

- Todas las restricciones son comprobadas con los mismos mecanismos y procedimientos:
- Cada restricción tiene una condición que se evalúa a cierto o a falso.
- Cada restricción tiene un modo (constraint mode):
  - ▣ DEFERRED => diferido

Las restricciones con el modo DEFERRED son comprobadas al final de la ejecución de un grupo lógico de sentencias SQL (transacción).

- ▣ IMMEDIATE => inmediato

Las restricciones con el modo IMMEDIATE son comprobadas después de la ejecución de cada sentencia SQL.

# 4 – Restricciones de Integridad

90

Restricciones  
de Tabla  
/Columna

## □ Nulidad

- ▣ Las filas de la tabla no pueden tener valores nulos

# 4 – Restricciones de Integridad

91

Restricciones  
de Tabla  
/Columna

- Unicidad (unique)
  - ▣ Dos filas de la tabla no pueden tener el mismo valor no nulo en las columnas indicadas.
  - ▣ Puede aplicarse al conjunto completo de columnas

# 4 – Restricciones de Integridad

92

Restricciones  
de Tabla /  
Columna

- Clave primaria (primary key)
  - Es una restricción de unicidad pero en la cual las columnas no pueden tomar valor nulo (integridad de entidades)

# 4 – Restricciones de Integridad

93

Restricciones  
de Tabla /  
Columna

- Integridad referencial (foreign key-references) (**VER CREATE TABLE**)
  - ▣ Representa una clave ajena o foránea.
  - ▣ Incluye una o más columnas referenciantes (referencing columns) y las correspondientes columnas referenciadas (referenced columns) de una tabla base referenciada (referenced table), que puede ser la referenciante.
  - ▣ Las columnas referenciadas deben tener una restricción de unicidad en la tabla referenciada.

# 4 – Restricciones de Integridad

94

Restricciones  
de Tabla /  
Columna

- Control (check)
  - ▣ Incluye una condición que sólo puede referir a elementos de la tabla que la incluye.

# 4 – Restricciones de Integridad

95

Restricciones  
de Tabla /  
Columna

- Valor por defecto
  - ▣ Especifica un valor por defecto que tomara la columna si no se indica un valor explicito

# 4 – Restricciones de Integridad

96

## Sintaxis

□ <definición de restricción de tabla> ::=  
[ CONSTRAINT <nombre de restricción>  
<restricción de tabla> [ <modo de  
restricción> ]



# 5 – Seguridad

97

## Seguridad

- Debemos proteger la base de datos de actualizaciones malintencionadas
  - ▣ Cada usuario debe ver solo los datos que ellos necesiten
  - ▣ Los usuarios tienen privilegios, y deben solo podran operar segun el modo y el conjunto de datos que se les haya concedido
    - Sentencias Grant and Revoke
    - Tablas Virtuales: View

# 5 – Seguridad

98

## Usuarios

PostgreSQL permite conceder permisos a usuarios o roles de la base de datos:

- **create user** <nombre usuario> **with password** <password>;
- **drop user** <nombre usuario>;

Ejemplo:

- *create user ud10 with password 'ud10';*

# 5 – Seguridad

99

## Privilegios

- Obtención de privilegios:
  - ▣ El creador de un objeto es el dueño
  - ▣ El dueño tiene todos los privilegios y puede conceder privilegios a otros usuarios

# 5 – Seguridad

100

Conceder/  
Revocar  
permisos

□ **grant** <lista de privilegios>  
    [**on** <objetos>]  
    **to** <usuarios o roles>  
    [**with grant option**];

Ejemplos:

- *create user ud10 with password 'ud10';*
- *grant connect on database postgres to ud10;*
- *grant select on table emp to ud10;*
- *grant select, insert, update, delete on table emp to ud10;*
- *grant all privileges on emp to ud10;*

# 5 – Seguridad

101

Conceder/  
Revocar  
permisos

□ **revoke** <lista de privilegios>  
    [**on** <objetos>]  
    **from** <usuarios o roles>  
    [**restrict** | **cascade**];

Ejemplo:

□ *revoke all privileges on emp from ud10;*

# Lenguaje SQL



**Terminamos!!!**