

# FUNCIONES AGREGADAS

# Cláusula GROUP BY y HAVING Declaración de SELECT : Sintaxis

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[HAVING     group_condition]
[ORDER BY  column] ;
```

- Las funciones de grupo operan sobre conjuntos de filas para dar un resultado por cada grupo
- Las funciones de grupo pueden utilizarse tanto en la sentencia SELECT como en la sentencia HAVING.
- GROUP BY divide filas en los grupos más pequeños.
- HAVING restringe el resultado (condición)

# Funciones de Grupo

<b>AVG</b> (DISTINCT   <u>ALL</u>   <i>n</i> )	Valor promedio de <i>n</i> , ignorando valores nulos
<b>COUNT</b> (DISTINCT   <u>ALL</u>   <i>expr</i>  *)	Nº de filas. Con * cuenta todas, incluyendo nulos
<b>MAX</b> (DISTINCT   <u>ALL</u>   <i>expr</i> )	Máximo valor de la expresión
<b>MIN</b> (DISTINCT   <u>ALL</u>   <i>expr</i> )	Mínimo valor de la expresión
<b>STDDEV</b> (DISTINCT   <u>ALL</u>   <i>n</i> )	Desviación estándar (ignora nulos)
<b>SUM</b> (DISTINCT   <u>ALL</u>   <i>n</i> )	Suma valores. Ignora nulos
<b>VARIANCE</b> (DISTINCT   <u>ALL</u>   <i>n</i> )	Varianza. Ignora nulos

# Funciones de GRUPO

- Se puede usar **AVG** y **SUM** con columnas que guardan datos numéricos exclusivamente.

```
SQL> SELECT    SUM(MONTO_TOTAL) ,  AVG(MONTO_TOTAL) ,  
2                  MIN(MONTO_TOTAL) ,  MAX(MONTO_TOTAL)  
3  FROM  B_VENTAS  
4* WHERE  ID_CLIENTE = 2 ;
```

- **MAX** y **MIN** pueden utilizarse para cualquier tipo de dato.

```
SQL> SELECT MIN(apellido) ,  MAX(apellido)  
2  FROM  b_personas  
3  WHERE  ES_CLIENTE = 'S' ;
```

# Función COUNT : Ejemplos

- COUNT (\*) retorna el número de filas en una tabla.

```
SQL> SELECT COUNT(*)
  2  FROM  b_posicion_actual
  3 WHERE  id_area = 12;
```

- COUNT(expr) retorna el número de filas no-nulas.

```
SQL> SELECT COUNT(RUC)
  2  FROM  b_personas
  3 WHERE  ES_CLIENTE = 'S';
```

# La cláusula GROUP BY

## Sintaxis

```
SELECT      column, group_function  
FROM        table  
[WHERE      condition]  
[GROUP BY   group_by_expression]  
[ORDER BY   column];
```

- Divide filas en una tabla en los grupos más pequeños usando la cláusula GROUP BY.
- La misma lista de columnas que se utilizan en la cláusula SELECT, deben ser utilizadas en la cláusula GROUP BY
- Defina el orden predefinido usando ORDER BY.

# Sin la cláusula GROUP BY

```
SQL> SELECT id, apellido, id_localidad  
2   FROM b_personas  
3  WHERE id_localidad = 12  
4  AND es_cliente= 'S';
```

ID	APELLIDO	ID_LOCALIDAD
5	BRIZUELA CUEVAS	12
6	ACHUCARRO	12
7		12

La localidad 12 se despliega **tres** veces porque está asignado a 3 clientes

# Con la cláusula GROUP BY

```
SQL> SELECT id_localidad, count(*) Cantidad  
2 FROM b_personas  
3 WHERE id_localidad = 12  
4 AND es_cliente= 'S'  
5 GROUP BY id_localidad;
```

ID_LOCALIDAD	CANTIDAD
-----	-----
12	3 ←

**Group By:** despliega una línea de datos para cada grupo recuperado por la cláusula WHERE, y COUNT (\*). En este caso corresponde al número de clientes de la localidad 12.

# GROUP BY : Ejemplos

- Despliegue los diferentes tipos de personas y la cantidad por cada tipo

```
SQL> SELECT tipo_persona, COUNT(*) Cantidad  
2  FROM b_personas  
3  GROUP BY tipo_persona;
```

- Determine todas las categorías salariales ocupadas, la cantidad de empleados que poseen dichas categorías

```
SQL> SELECT p.cod_categoria, COUNT(p.cedula)  
2  FROM B_POSICION_ACTUAL p  
3  WHERE fecha_fin IS NULL  
4  GROUP BY p.Cod_Categoría;
```

# Desplegando Grupos específicos usando la cláusula HAVING

```
SQL> SELECT p.cod_categoria CAT, COUNT(p.cedula) Cantid  
2  FROM B_POSICION_ACTUAL p  
3  WHERE fecha_fin IS NULL  
4  GROUP BY p.Cod_Categoría  
5  HAVING COUNT(p.cedula) > 1;
```

La cláusula HAVING condiciona los grupos

CAT	CANTID
42	2
43	2
48	2
49	2
56	3
66	2

Sólo se despliegan aquellas categorías que tienen más de 1 empleado

# La cláusula HAVING: Sintaxis

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column] ;
```

- Use la cláusula HAVING para restringir grupos.
  - Paso 1: Se agrupan filas.
  - Paso 2: Se aplica la función de grupo.
  - Paso 3: Se despliegan los grupos que cumplen con la condición del HAVING.

# Errores que no deben cometerte:

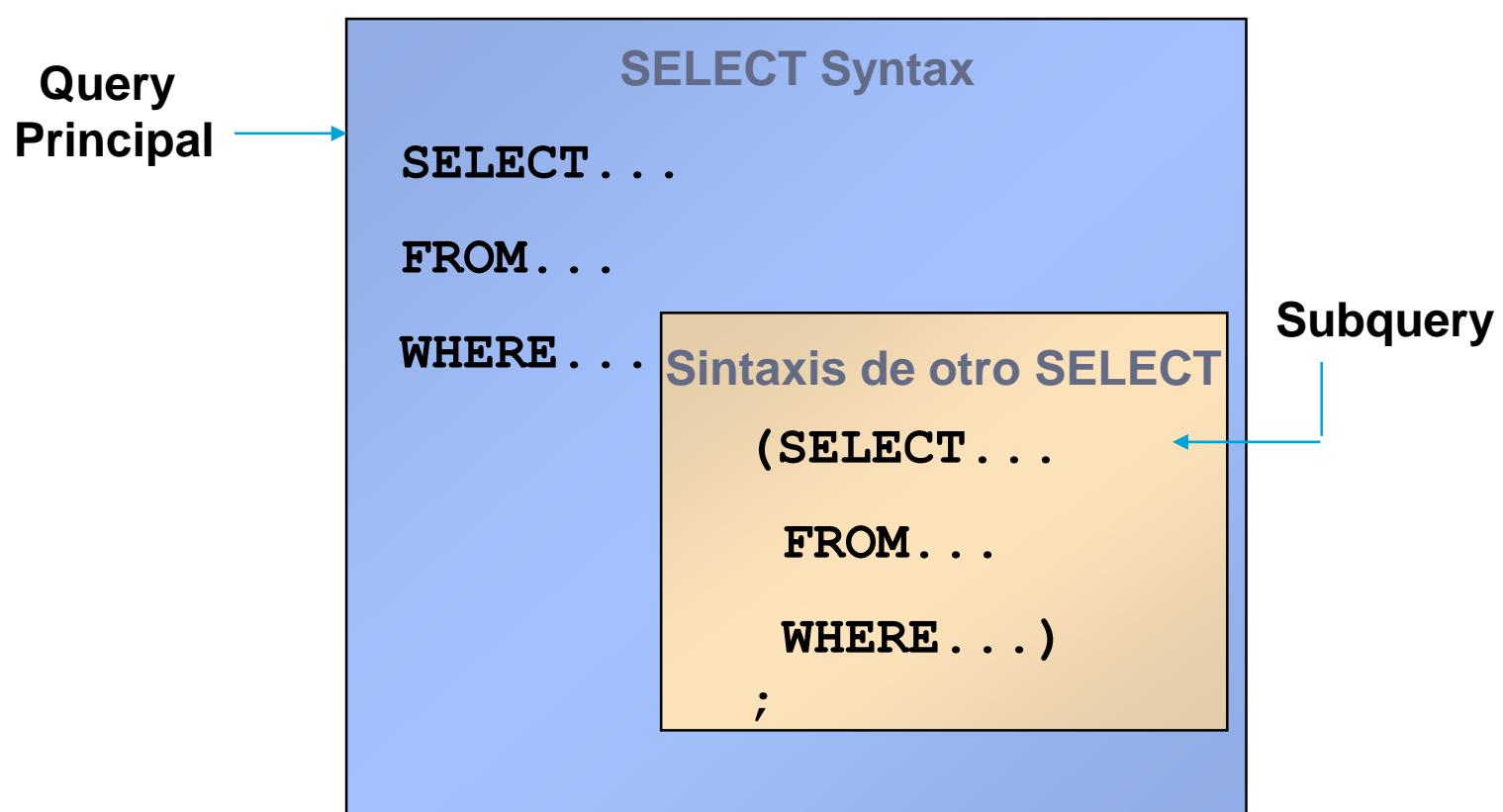
- Todas las columnas citadas en el SELECT que sean parte de la función agregada DEBEN estar en la cláusula GROUP BY, pues de lo contrario, dará un mensaje de error
- La cláusula WHERE no puede utilizarse para condicionar grupos.

# **SUBQUERIES**

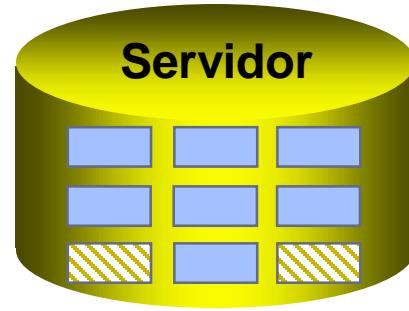
# LAS SUBCONSULTAS

- Una Subconsulta (subquery) es una sentencia SELECT ‘embebida’ en otra sentencia SQL.
- Son muy útiles cuando se trata de seleccionar filas de una tabla con una condición que depende de la tabla en sí.
- Se puede usar subqueries en SQL con las cláusulas:
  - WHERE (QUERY ANIDADO)
  - HAVING
  - FROM (INLINE VIEWS)
  - FROM (para seleccionar las filas en un sentencia CREATE o INSERT)

# Formato del Sub query



Ejemplo de la subconsulta: Seleccionar los empleados que tienen el mismo jefe que "Ferreira"



**B\_EMPLEADOS**

APELLIDO	CEDULA_JEFE
Ferreira	952160
Moreno	952160
Gonzalez	952160
Caballero	952160
Balmaceda	952160

**B\_EMPLEADOS**

CEDULA_JEFE
952160

# Ejemplo de la subconsulta

```
SQL> SELECT NOMBRE, APELLIDO, CEDULA_JEFE  
      FROM B_EMPLEADOS  
     WHERE CEDULA_JEFE = (SELECT CEDULA_JEFE FROM  
                           B_EMPLEADOS  
                          WHERE UPPER(APELLIDO) = 'FERREIRA');
```

Seleccionar los empleados que tienen el mismo jefe que “Ferreira”

# SINTAXIS DE LAS SUB CONSULTAS

```
SELECT      lista de columnas
FROM        tabla
WHERE       expresión Operador
            (SELECT      lista de columnas
             FROM        tabla);
```

- El subquery se ejecuta una vez antes del Query principal.
- El resultado del subquery es usado por el Query exterior principal.

# Indicaciones para los subquerries:

- La subconsulta debe aparecer entre paréntesis
- Se pueden utilizar subconsultas con operadores de comparación de una sola fila ( $>$ ,  $=$ ,  $\geq$ ,  $<$ ,  $\neq$ ,  $\leq$ ) y de MÚLTIPLES FILAS ( IN, NOT IN).
- La subconsulta debe aparecer en el lado derecho del operador
- NO PUEDEN TENER la cláusula ORDER BY.

# ¿Cómo se procesan las consultas anidadas?

1. La declaración SELECT anidada se ejecuta primero.
2. El resultado se pasa en la condición de pregunta principal.

Query anidado

```
(SELECT CEDULA_JEFE  
FROM  
    B_EMPLEADOS  
 WHERE UPPER(APELLIDO)=  
      'FERREIRA')
```

Query Principal

```
SELECT NOMBRE, APELLIDO,  
       CEDULA_JEFE  
  FROM B_EMPLEADOS  
 WHERE CEDULA_JEFE =
```

**Retorna cedula\_jefe= 952160**

# Consultas de una sola fila

```
SELECT NOMBRE, APELLIDO,  
CEDULA_JEFE  
FROM B_EMPLEADOS  
WHERE CEDULA_JEFE =
```

Escriba la declaración de SQL para desplegar el nombre y apellido del empleado.

```
SELECT CEDULA_JEFE FROM  
B_EMPLEADOS  
WHERE UPPER(APELLIDO)=  
'FERREIRA'
```

Escriba la consulta para averiguar el jefe de Ferreira

```
SELECT NOMBRE, APELLIDO,  
CEDULA_JEFE  
FROM B_EMPLEADOS  
WHERE CEDULA_JEFE =  
(SELECT CEDULA_JEFE FROM  
B_EMPLEADOS  
WHERE UPPER(APELLIDO)=  
'FERREIRA')
```

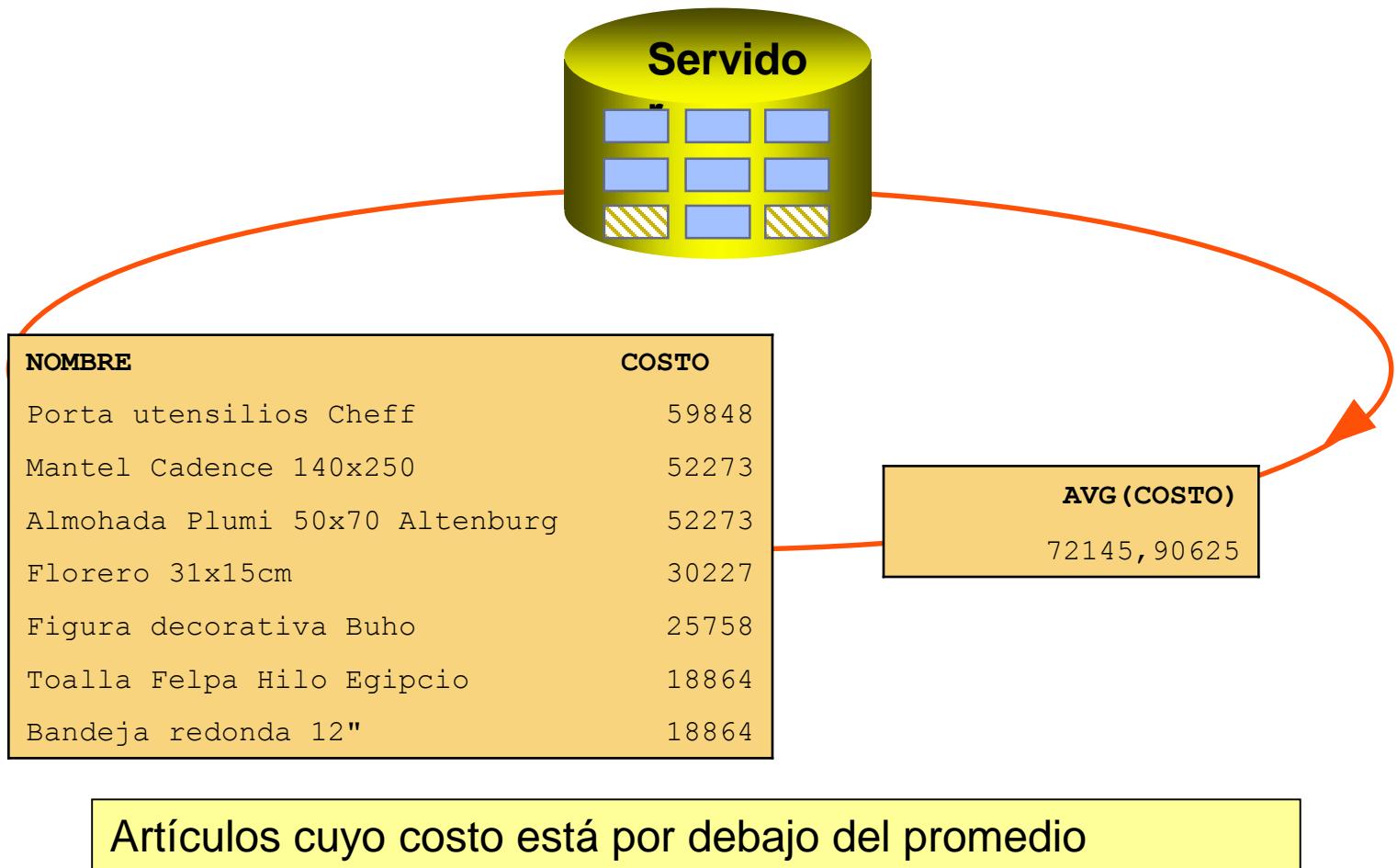
Ponga ambas declaraciones y deje que el SQL determine el valor de comparación

# Ejemplo de la utilización de funciones de grupo en la subconsulta

```
SQL> SELECT nombre, costo  
2   FROM b_articulos  
3  WHERE costo < (SELECT AVG(costo)  
4                      FROM b_articulos)  
5 ORDER by costo DESC;
```

Seleccionar el nombre y costo de los artículos cuyo costo está por debajo del promedio.

# Utilización de funciones de grupo en una subconsulta



# Subquery escalar

- Un subquery escalar es aquel que retorna exactamente una columna de una fila
- De allí que el subquery escalar puede utilizarse en expresiones tales como
  - DECODE y CASE
  - Todas las cláusulas de SELECT, con excepción de GROUP BY

# Ejemplo de uso de un subquery escalar en el CASE:

```
select nombre_cta,
(CASE
    WHEN id_tipo = (SELECT id   FROM b_tipo_cuenta
                      WHERE nombre_categoria
= ('ACTIVO'))
        THEN 'ACTIVO'
    ELSE
        'Otra clasificación'
    END) Clasificación
FROM b_cuentas;
```

# Errores con Subqueries

- Si se escribe un subquery que devuelve más de una fila y se usa un operador de comparación de fila, la sentencia dará error.

```
SQL> SELECT nombre, apellido
      FROM b_personas
      WHERE id_localidad =
        (SELECT id FROM b_localidad
         WHERE UPPER(nombre) LIKE 'CAA%');

ERROR en línea 4:
ORA-01427: la subconsulta de una sola fila
devuelve más de una fila
```

- Para corregir el error, cambie al operador de la comparación a IN, un operador de la fila múltiple.

# Ejemplo de subconsultas que retornan múltiples filas

- Un subquery de filas múltiples devuelve más de 1 fila.
- Usted debe usar a un operador de la fila múltiple en la cláusula WHERE, por ejemplo el operador IN.

```
SQL> SELECT nombre, apellido, id_localidad  
      FROM b_personas  
     WHERE id_localidad IN  
    (SELECT id FROM b_localidad  
     WHERE  UPPER(nombre)  LIKE  'CAA%' );
```

# Utilización de cláusula HAVING en una SUBCONSULTA

- También se puede utilizar una subconsulta en la cláusula HAVING
- Siempre se ejecuta primeramente la SUBCONSULTA y retorna el resultado a la cláusula HAVING de la consulta principal

```
SQL> SELECT CEDULA_VENDEDOR, AVG(MONTO_TOTAL) COSTO  
      FROM B_VENTAS  
     GROUP BY CEDULA_VENDEDOR  
    HAVING AVG(MONTO_TOTAL) <  
        (SELECT AVG(MONTO_TOTAL) FROM B_VENTAS) ;
```

# Subqueries con múltiples columnas

Se pueden comparar más de una columna, para ello la sintaxis es la siguiente:

```
SELECT columna, columna, ...
FROM tabla
WHERE (columna, columna, ...) IN
      (SELECT columna, columna, ...
       FROM tabla
       WHERE condición)
```

# Consultas correlacionadas

- Un subquery correlacionado es aquel que contiene una referencia a una tabla que aparece el query exterior (query principal). La sintaxis es la siguiente:

# Uso del operador EXISTS en consultas correlacionadas

- El operador EXISTS verifica la existencia de filas en el conjunto resultante del subquery
- Si se encuentra la fila, la búsqueda ya no continúa y se retorna TRUE.
- Ejemplo: Seleccionar todos los empleados que tienen al menos un empleado que depende de él

```
SELECT CEDULA, NOMBRE FROM  
B_EMPLEADOS E  
WHERE EXISTS (SELECT 'X'  
                FROM B_EMPLEADOS  
                WHERE     CEDULA_JEFE = E.CEDULA) ;
```

# EXISTS vs IN

**Tomando el mismo ejemplo: Seleccionar todos los empleados que tienen al menos un empleado que depende de él (Es decir, que son jefes):**

```
SELECT CEDULA, NOMBRE FROM  
B_EMPLEADOS E  
WHERE CEDULA IN (SELECT CEDULA_JEFE  
FROM B_EMPLEADOS);
```

```
SELECT CEDULA, NOMBRE FROM  
B_EMPLEADOS E  
WHERE EXISTS (SELECT 'X'  
FROM B_EMPLEADOS  
WHERE CEDULA_JEFE = E.CEDULA);
```

**Si bien generan el mismo resultado, se procesan de modo diferente. EXISTS puede ser más eficiente si el Subquery tiene muchos datos.**

# NOT EXISTS vs NOT IN

Aunque pareciera que son equivalentes puede que no generen el mismo resultado.

Ej: Ver todos los empleados que no son jefes.

```
SELECT CEDULA, NOMBRE FROM
B_EMPLEADOS E
WHERE CEDULA NOT IN (SELECT CEDULA_JEFE
FROM B_EMPLEADOS);
```

```
SELECT CEDULA, NOMBRE FROM
B_EMPLEADOS E
WHERE NOT EXISTS (SELECT 'X'
                     FROM B_EMPLEADOS
                     WHERE     CEDULA_JEFE = E.CEDULA);
```

Verifique el resultado, debido a la presencia de valores NULL en el subquery!

# INLINE VIEWS (Vistas Inlines)

- **Un sub query en la cláusula FROM es llamada vista inline, ya que el sub query representa virtualmente una vista.**

```
SELECT      lista_de_camplos  
FROM <subconsulta>;
```

```
SELECT      CLIENTE,  MONTO_VENTAS  
FROM  
(SELECT  ID_CLIENTE CLIENTE,  SUM(MONTO_TOTAL)  
MONTO_VENTAS  
      FROM B_VENTAS  
     GROUP BY ID_CLIENTE)  V  
/
```

# Uso de la cláusula WITH

- Mediante la cláusula WITH, puede definir un bloque de consulta antes de utilizarlo en una consulta. La cláusula WITH (conocida formalmente como `subquery_factoring_clause`) le permite reutilizar el mismo bloque de consulta en una sentencia SELECT cuando se produce más de una vez dentro de una consulta compleja.
- Esto resulta particularmente útil cuando una consulta tiene muchas referencias al mismo bloque de consulta y hay presentes uniones y agregaciones.

Ejemplo: Ver el id, nombre y cantidad del artículo más vendido

# Ejemplo sin la cláusula WITH

```
SELECT v.id_articulo, a.nombre, SUM(v.cantidad)
      cantidad
  FROM b_detalle_ventas v JOIN b_articulos a
    ON a.id = v.id_articulo
 GROUP BY v.id_articulo, a.nombre
 HAVING SUM(v.cantidad) =
 (SELECT MAX(v.cantidad) maximo
  FROM
 (SELECT v.id_articulo, a.nombre,   SUM(v.cantidad)
      cantidad
  FROM b_detalle_ventas v JOIN b_articulos a
    ON a.id = v.id_articulo
 GROUP BY v.id_articulo, a.nombre) v);
```

# Ejemplo con el uso de la cláusula WITH

```
WITH sum_ven AS
  (SELECT v.id_articulo, a.nombre,    SUM(v.cantidad)
   cantidad
    FROM b_detalle_ventas v JOIN b_articulos a
    ON    a.id = v.id_articulo
   GROUP BY v.id_articulo, a.nombre)
  SELECT v.id_articulo, v.nombre, v.cantidad
    FROM sum_ven v
   WHERE v.cantidad =
  (SELECT MAX(v.cantidad) maximo
    FROM sum_ven v);
```

# QUERIES JERÁRQUICOS

Si la tabla contiene datos organizados jerárquicamente, es posible recuperar los datos en orden jerárquico utilizando la **cláusula jerárquica**:

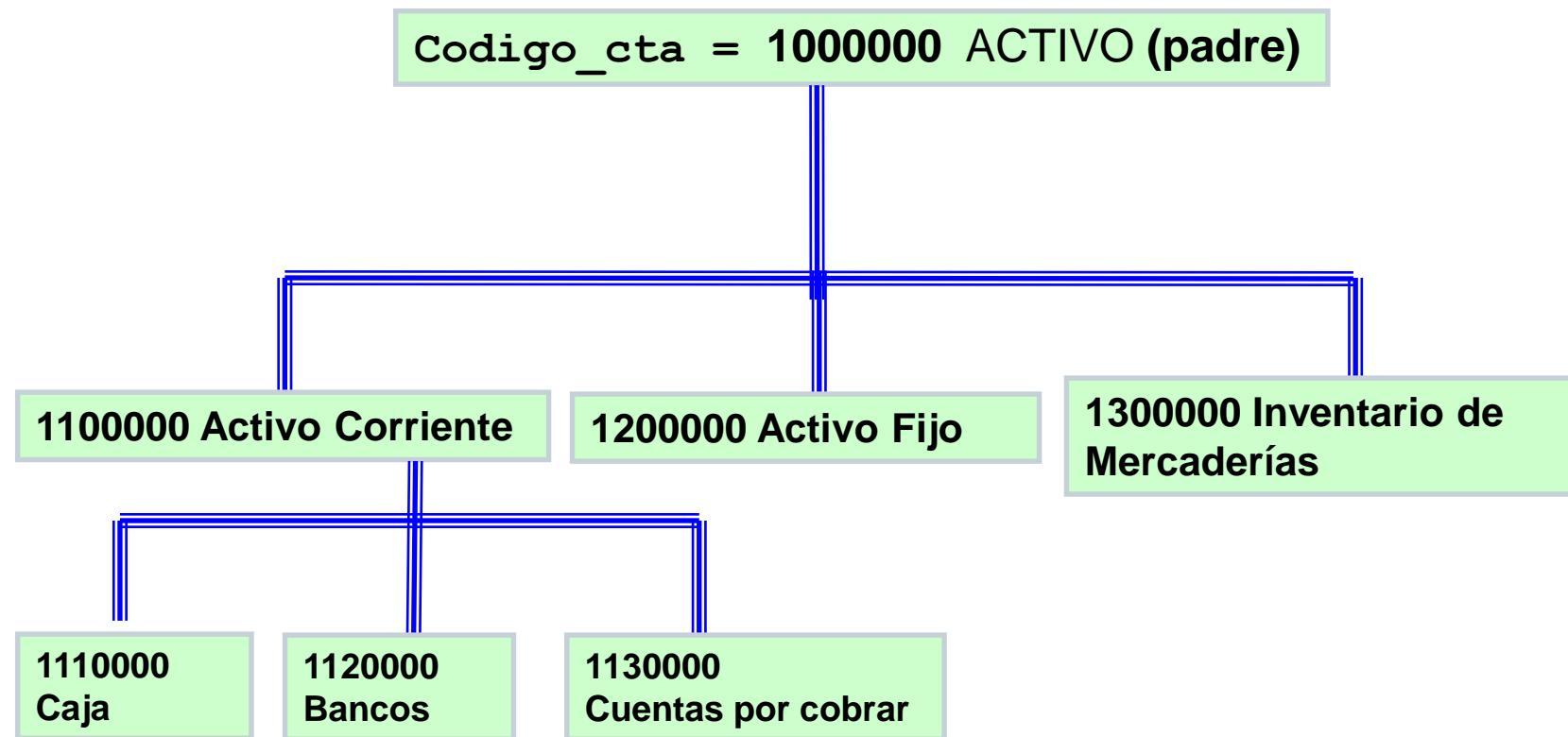
**START WITH <condición>**

**CONNECT BY <condición>**

# La cláusula jerárquica

- Permite listar los datos reflejando la ramificación de los datos cuando éstos están estructurados en orden jerárquico
- El primer elemento de una jerarquía se llama nodo raíz. Las ramificaciones son nodos, y el último elemento de una ramificación es la hoja.
- La cláusula START WITH especifica la fila raíz de la jerarquía
- La cláusula CONNECT BY es la que permite fijar la relación de dependencia entre las filas ‘padres’ y las filas ‘hijas’ dentro de la jerarquía
- PRIOR se refiere al sentido del recorrido

# Ejemplo de Estructura de árbol en la tabla b\_cuentas



# Recuerde:

- El recorrido del árbol (Top Down o Bottom Up) depende de la cláusula **PRIOR**:

## Ver la cuenta de ACTIVO (1000000) y las que dependen de ella

```
SELECT codigo_cta, nombre_cta  
FROM b_cuentas  
start with codigo_cta = 1000000  
connect BY PRIOR codigo_cta =  
cta_superior;
```

## Ver las cuentas superiores a la cuenta 1130000:

```
SELECT codigo_cta, nombre_cta  
FROM b_cuentas  
start with codigo_cta = 1130000  
connect BY codigo_cta = PRIOR  
cta_superior;
```

- Para mostrar la jerarquía en los datos recuperados:

```
SELECT LPAD(' ',2*(LEVEL-1)) ||  
TO_CHAR(codigo_cta,'0000000') codigo_cta,  
nombre_cta  
FROM b_cuentas  
start with codigo_cta = 1000000  
connect BY PRIOR codigo_cta = cta_superior;
```

CODIGO_CTA	NOMBRE_CTA
1000000	ACTIVO
1100000	ACTIVO CORRIENTE
1110000	CAJA
1110100	CAJA CHICA
1120000	BANCOS
1120100	Cta. Cte. 386881 Banco ITAU
1120200	Cta.Cte. 504 Banco Nacional de Fomento
1130000	Cuentas por cobrar CLIENTES
1130100	Cadena Hiperseis
1130200	Supermercado STOCK
1130300	Supermercado Gran Vía
1200000	ACTIVO FIJO
1210000	Rodados
1220000	Inmuebles
1300000	INVENTARIO MERCADERIAS
1310000	MERCADERIAS