Bases de datos

Tema IV – El comando SELECT

Universidad Nacional del Litoral
FACULTAD DE INGENIERÍA
Y CIENCIAS HÍDRICAS

SQL - Select

Formato

SELECT *select_list* ***Cláusulas;***

Cláusulas

Cláusula	Finalidad
FROM	Nombres de las tablas de las que se seleccionan filas
WHERE	Especifica las condiciones que deben satisfacer las filas seleccionadas
GROUP BY	Agrupa las filas seleccionadas en grupos especificados
HAVING	Indica la condición que ha de satisfacer cada grupo mostrado en el **GROUP BY**
ORDER BY	Especifica el orden en que se mostrarán las filas seleccionadas

SQL – Select - Cláusulas

FROM

- Especifica una o varias tablas a partir de las cuales se recuperarán las filas que se desean.
- Si no hay expresiones optativas en la consulta (where, group by, having u order by), entonces la tabla que se ha recuperado es la tabla completa compuesta por las columnas de la lista objeto solamente (select_list).
- Si la lista objeto contiene columnas de más de una tabla, entonces la cláusula FROM deberá nombrar todas las tablas en cualquier orden, independientemente del orden de las columnas en la lista objeto.

```
SELECT cod_post, nom_loca FROM localidad;
```


SQL - Select - Cláusulas

WHERE

- Especifica una tabla obtenida por la aplicación de una condición de búsqueda a las tablas que se listan en la cláusula FROM.

```
SELECT columna1, columna2, ....., columnaN
    FROM nombre_de_la_tabla
    WHERE condición;
```

- Esta cláusula puede contener una o más subconsultas. Si es así, cada subconsulta que sigue a la cláusula WHERE se ejecuta para cada fila recuperada por la cláusula FROM.

SQL - Select - Cláusulas

GROUP BY

- Especifica una tabla agrupada que resulta de la aplicación de la cláusula GROUP BY al resultado de cualquier cláusula especificada previamente.
- Hace referencia de manera específica a una columna o varias columnas de la tabla nombrada en la cláusula FROM y agrupa las filas sobre la base de los valores de esas columnas.
- El resultado de una cláusula GROUP BY divide el resultado de la cláusula FROM en un conjunto de grupos, de forma que para cada grupo de más de una fila, los valores de la columna agrupada son idénticos.

SQL - Select - Cláusulas

GROUP BY

```
SELECT columna1, columna2, ......, columnaN
    FROM nombre_de_la_tabla
    GROUP BY columna_de_agrupación1, ...
        columna_de_agrupaciónN;
```

SQL - Select - Cláusulas

GROUP BY

- TODAS LA COLUMNAS de la select_list deben estar en GROUP BY.
- Si el comando no contiene una cláusula WHERE, entonces la cláusula GROUP BY se coloca inmediatamente detrás de la cláusula FROM. Si el comando tiene la cláusula WHERE, entonces la cláusula GROUP BY va detrás de aquella.
- Las filas devueltas, estarán ordenadas al azar dentro de cada grupo porque esta cláusula no realiza ningún tipo de ordenamiento.

7

SQL - Select - Cláusulas

HAVING

- Especifica una restricción en la tabla agrupada que resulta de la cláusula anterior GROUP BY y elimina los grupos que no satisfagan la condición especificada.
- Si se especifica HAVING en una consulta, entonces también se tendrá que haber especificado GROUP BY.
- Se utiliza para especificar la cualidad que debe poseer un grupo para que éste sea devuelto.

8

SQL - Select - Cláusulas

HAVING

- Compara una propiedad del grupo con un valor constante. Realiza la misma función con los grupos que WHERE realiza con las filas individuales eliminando los grupos que no poseen la cualidad, de la misma manera que WHERE elimina las filas que no poseen esa cualidad.

9

SQL - Select - Cláusulas

HAVING

```
SELECT columna1, columna2, ....., columnaN
    FROM nombre_de_la_tabla
    GROUP BY columna/s_de_agrupación
    HAVING propiedad_especificada_del_grupo;
```

10

SQL - Select - Cláusulas

HAVING

Por ejemplo, suponga la tabla empleados con la siguiente estructura: (documento, nombre_empleado, nombre_puesto, sueldo), y se desea encontrar el sueldo anual medio de todos los puestos de trabajo en que hay más de un empleado, entonces la consulta será:

```
SELECT nombre_puesto, COUNT(*), 12*AVG(sueldo)
    FROM empleados
    GROUP BY nombre_puesto
    HAVING COUNT(*) > 1;
```

11

SQL - Select - Cláusulas

ORDER BY

- Especifica el orden en que aparecerán las filas en la recuperación haciendo una lista de las filas que hay en un grupo especificado de acuerdo con el valor creciente o decreciente.
- Si se usa la cláusula ORDER BY, ésta deberá ser la última cláusula del comando SELECT.
- Puede especificarse el orden ascendente (ASC) o descendente (DESC).
- Si la columna ordenada consta de letras SQL utilizará el orden alfabético ascendente (comenzando con la A) si no se especifica DESC.

12

SQL - Select - Predicados

- Los predicados son condiciones que se indican en la cláusula WHERE de una consulta SQL. Los predicados que permite SQL son:
  - la comparación
  - los cuantificadores
  - los existenciales

13

SQL - Select - Predicados

Comparación

- Compara dos valores. Consta de una expresión de valor, seguida de un operador de comparación, seguido, a su vez, ya sea de otra expresión de valor o de una subconsulta de valor único.
- Los tipos de datos de las dos expresiones de valores, o la expresión de valor y la subconsulta, deben ser comparables.

14

SQL - Select - Predicados

Comparación

- Los operadores de comparación incluidos en SQL son

  =, <> ó !=,<, >, <=, >=

- Si los valores que hay a ambos lados del operador de la comparación no son NULL, entonces el predicado de la comparación es verdadero o falso.

15

SQL - Select - Predicados

Comparación

- Si alguna de las dos expresiones de valores es un valor NULL o si la subconsulta está vacía, entonces el resultado del predicado de la comparación es desconocido;
- Cuando se usa GROUP BY, ORDER BY o DISTINCT junto con un predicado de comparación, un valor NULL es idéntico a otro valor NULL o es un duplicado del mismo.

16

SQL - Select - Predicados

Comparación

- Las cadenas de caracteres se pueden comparar por medio de los operadores de comparación mencionados antes. Esto se consigue comparando los caracteres que se encuentran en las mismas posiciones ordinales de la cadena. Así, dos cadenas de caracteres son iguales si todos los caracteres con la misma posición ordinal, son iguales.

17

SQL - Select - Predicados

BETWEEN

- Especifica la comparación dentro de un intervalo. La sintaxis es:

  . . . [NOT] BETWEEN *valor* AND *valor*

- Los tipos de datos de los valores, deben ser comparables.
- Para seleccionar los productos cuyos precios estén entre $ 5 y $ 10 de la tabla producto:

```
SELECT cod_prod, nom_prod, precio
    FROM producto
    WHERE precio BETWEEN 5 AND 10;
```

18

SQL - Select - Predicados

BETWEEN

- La respuesta que se obtendrá, viene dada en cualquier orden. Si se efectúa nuevamente la misma consulta, puede ser que el orden, sea totalmente distinto al obtenido en el primer intento.
- El término NOT se puede utilizar con BETWEEN para recuperar la información que hay fuera de un intervalo en lugar de la que hay dentro del mismo. Por ejemplo, para recuperar la información de los productos cuyo precio es inferior a $ 3 y superior a $ 6, será:

```
SELECT Cod_prod, Nom_prod, Precio
   FROM producto
   WHERE precio NOT BETWEEN 3 AND 6;
```

19

SQL - Select - Predicados

IN

- Es equivalente al uso de OR. Especifica una comparación cuantificada. Hace una lista de un conjunto de valores y prueba si un valor está en esa lista.
- La lista debe ir entre paréntesis. Por ejemplo, para recuperar los productos cuyo precio es alguno de los siguientes: $ 4, $ 5, $ 7, será:

```
SELECT cod_prod, nom_prod, precio
   FROM producto
   WHERE precio IN (4,5,7);
```

20

SQL - Select - Predicados

IN

- Es equivalente a:

```
SELECT cod_prod, nom_prod, precio
   FROM producto
   WHERE precio = 4 OR precio = 5
         OR precio = 7;
```

- La consulta también se puede escribir usando ANY:

```
SELECT cod_prod, nom_prod, precio
   FROM producto
   WHERE precio = ANY(4,5,7);
```

21

SQL - Select - Predicados

IN

- El orden en que los elementos de la lista se especifican (select_list) en la consulta determina el orden en que aparecerán las columnas en la recuperación. *No determina el orden en que aparecerán las filas*.
- En caso de desear un ordenamiento específico, debe emplearse a continuación del WHERE la cláusula ORDER BY. Como en el caso anterior, también es válido el uso de NOT IN.

22

SQL - Select - Predicados

LIKE

- El predicado LIKE especifica una comparación de caracteres pudiendo usarse substrings y comodines.
- El *guión bajo* ("_") representa un único carácter.
- El porcentaje ("%") representa una cadena de caracteres de longitud arbitraria (incluyendo cero caracteres).
- Otro elemento más que se utiliza ya propio del motor en cuestión, es el símbolo arroba ("@") usado como carácter de escape para poder utilizar símbolos especiales (caso de "_" o "%" o ").

23

SQL - Select - Predicados

NULL

- Especifica la prueba que se lleva a cabo con un valor NULL. Por ejemplo, si se busca el nombre de un producto en el que debido a la falta de información se ha puesto NULL al precio, NO se puede especificar:

```
WHERE precio = NULL;
```

… porque nada, ni incluso el mismo NULL, es igual al valor de NULL. SQL tampoco permite utilizar NULL en la cláusula SELECT.

24

SQL - Select - Predicados

NULL

- No se puede encontrar el valor NULL por exclusión, como por ejemplo, estipulando que el precio está por encima o por debajo de cualquier otro precio conocido de la lista. Por ejemplo, si $ 10 es el mayor precio especificado de la lista, la consulta:

```
SELECT nom_prod, precio
  FROM  producto WHERE precio > 10;
```

.. no devolverá el nombre del producto con el valor NULL, ni tampoco lo devolvería con:

```
SELECT nom_prod, precio FROM  producto
  WHERE precio < 0;
```

25

SQL - Select - Predicados

NULL

- El único predicado que se puede utilizar cuando se busca un valor NULL es:

```
WHERE especificación_de_columna IS
[NOT] NULL;
```

26

SQL - Select - Predicados

ALL, SOME, ANY

- Exigen que se use el predicado de la comparación aplicado a los resultados de una ***subconsulta***.
- Permiten probar un valor único frente a todos los elementos de un conjunto. Por ejemplo, podría desearse encontrar los proveedores que no pertenecen a la ciudad de Santa Fe y cuya fecha de inicio de actividades sea inferior a la de todos los proveedores que son de la ciudad de Santa Fe:

```
SELECT nom_provee FROM proveedor
   WHERE Ciudad <> "Santa Fe"
      AND fecha_inicio < ALL
         (SELECT  fecha_inicio  FROM  proveedor
            WHERE Ciudad = "Santa Fe");
```

27

SQL - Select - Predicados

ALL, SOME, ANY

- Todos los otros operadores de comparación se pueden combinar con SOME, ANY y ALL.
- Cuando el operador de comparación que se está usando es igual, el término ANY se puede intercambiar con IN, e incluso algunas veces puede parecer más lógico usar el IN.
- En muchos casos, ANY tiene el mismo significado que SOME.

28

SQL - Select - Predicados

ALL, SOME, ANY

- Proveedores que no pertenecen a la ciudad de Santa Fe y cuya fecha de inicio de actividades sea inferior a la de algún proveedor de la ciudad de Santa Fe, se puede escribir de la siguiente forma:

```
SELECT nom_provee  FROM proveedor
  WHERE  ciudad <> "Santa Fe"
     AND fecha_inicio < ANY
        (SELECT fecha_inicio FROM proveedor
           WHERE ciudad = "Santa Fe");
```

La comparación < ANY de la cláusula WHERE del SELECT exterior es verdadera, si la fecha de inicio es menor que al menos un elemento del conjunto formado por todos los proveedores de Santa Fe.

29

SQL - Select - Predicados

EXISTS

- Indica las condiciones de un conjunto vacío.
- Contiene una subconsulta que junto de EXISTS, se puede evaluar como verdadera o falsa. Si el resultado de la subconsulta no existe, entonces el conjunto descrito por la subconsulta estará vacío.
- Representa el ***cuantificador existencial de la lógica formal***.
- El predicado EXISTS se puede usar siempre que se pueda usar una consulta con el predicado IN aunque no siempre se puede usar IN en lugar de EXISTS.
- Sintaxis:

```
SELECT select_list FROM nombre_de_la_tabla
  WHERE [NOT] EXISTS(subconsulta);
```

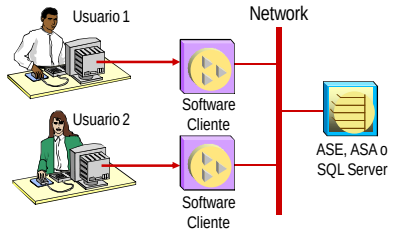
30

SQL - Select – SQL Server

Particularidades del SQL Server y Transact SQL

Ejemplos

Generalidades



- El software cliente, es utilizado para el acceso al servidor de base de datos (ASE o ASA o SQL Server) quien accede a la base de datos

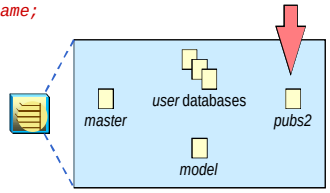
Seleccionando una base de datos

- Para conectarse con una base puede emplearse el comando **use**
  - Sintaxis

```
use database_name;
go
```

  - Ejemplo

```
use pubs2;
go
```



Recuperación simple: select / from

- Comandos para recuperar datos desde la base de datos
  - La cláusula **select** permite especificar las columnas a recuperar
  - La cláusula **from** permite especificar la o las tablas desde las cuales los datos serán recuperados
- Ejemplo:

```
select pub_name from publishers

pub_name
----------------------------------
New Age Books
Binnet & Hardley
Algodata Infosystems
```

Lista únicamente los nombres de publicación de la tabla *publishers*

(continúa…)

Recuperación Simple: select / from

- Este ejemplo, recupera más de una columna, listando los editores y los estados en los que están radicados:

```
select pub_name, state from publishers

pub_name                       state
------------------------------ ------
New Age Books                  MA
Binnet & Hardley               DC
Algodata Infosystems           CA
```

(continúa…)

Recuperación Simple: select / from

- Este ejemplo devuelve todas las columnas de la tabla *publishers*
- **"*"** es equivalente a **"todas las columnas"**

```
select * from publishers

pub_id  pub_name              city        state
------  --------------------  ----------- -----
0736    New Age Books         Boston      MA
0877    Binnet & Hardley      Washington  DC
1389    Algodata Infosystems  Berkeley    CA
```

Reordenando Columnas

- El orden de los nombres de las columnas en la cláusula **select** determina el orden de las columnas en la salida
- El siguiente ejemplo lista los valores de las columnas *state* y *pub_name* desde la tabla *publishers:*

```
select state, pub_name from publishers

state pub_name
----- ----------------------------------------
MA    New Age Books
DC    Binnet & Hardley
CA    Algodata Infosystems
```

37

Eliminación de elementos duplicados

- **distinct** elimina filas duplicadas en la salida
- El siguiente ejemplo lista los *states* que están en la tabla *stores* no utilizando **distinct**:

```
select state from stores

state
-----
WA
WA
CA
MA
CA
CA
OR
```

(continúa…)

38

Eliminación de elementos duplicados

- Este ejemplo que utiliza **distinct,** lista únicamente una ocurrencia de cada nombre de *state* encontrado en la tabla *stores*:

```
select distinct state from stores
state
-----
CA
MA
OR
WA
```

- Si la select list tiene más de una columna, **distinct** trabaja sobre todas las filas del conjunto resultado

39

Palabra clave: distinct

- Use **distinct** para:
  - Encontrar solamente los valores distintos de una columna en particular
  - Agrupar valores en categorías
- El siguiente ejemplo lista cada tipo de libro de la tabla *titles*:

```
select distinct type from titles

type
------------
UNDECIDED
business
mod_cook
popular_comp
psychology
trad_cook
```

40

Recuperación calificada: select/from /where

- La cláusula **where** limita cuales filas serán recuperadas
- Este ejemplo lista *stores* en el *state* de California unicamente:

```
select stor_name, state from stores
where state = "CA"
```

- Las condiciones de búsqueda son expresiones booleanas (true/false) que aplicadas a cada fila permiten determinar su inclusión o no en el conjunto resultado

(continúa…)

41

Recuperación calificada: select/from /where

- Predicados o calificadores de la cláusula **where**:
  - Operadores de comparación (=, >, <, >=, <=)
  - Rangos (**between** y **not between**)
  - Igualdad de caracteres (**like** y **not like**)
  - Valores desconocidos (**is null**, **is not null**)
  - Listas (**in** y **not in**)
  - Conectores lógicos (**and, or**)
- **not** niega cualquier expresión booleana y palabras claves como **like**, **null**, **between** e **in**

42

Operadores de comparación

Operador	Significado
=	Igual que
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
!=	No igual
<>	No igual
!>	No mayor que
!<	No menor que

43

Operadores de comparación: Ejemplos

- Este ejemplo, que usa el operador =, encuentra *titles de libros* que son del tipo *business*:

```
select title_id, title
from titles
where type = "business"
```

- Este ejemplo, que usa >=, encuentra los libros cuyo costo es $4.99 o más:

```
select title
from titles
where price >= $4.99
```

44

Uso de Rangos

- Use **between** para especificar un rango que incluya las ocurrencias

```
select stor_id, stor_name, city, state
from stores
where stor_id between "6380" and "7100"
```

- Use **not between** para especificar un rango que excluya las ocurrencias

```
select stor_id, stor_name, city, state
from stores
where stor_id not between "6380" and "7100"
```

45

Uso de like para igualdad entre caracteres

- Se utiliza para seleccionar filas que contengan columnas que coincidan con porciones especificadas de strings de caracteres
- Se usa con datos tipo carácter únicamente
- Pueden usarse *wildcards* o comodines

Wildcard	Significado
%	Cualquier string de cero o más caracteres
_	Cualquier carácter simple
[ ]	Cualquier carácter simple dentro del rango especificado entre corchetes
[^]	Cualquier carácter simple no incluido dentro del rango especificado entre corchetes

(continúa…)

46

Igualdad entre caracteres

- Los string de caracteres y comodines deben ir entre comillas o apóstrofos (depende del caso)
- Encontrar el nombre de *stores* que comienza con **B** :

```
select stor_name
from stores
where stor_name like "B%"
```

```
stor_name
----------------------------------------
Barnum's
Bookbeat
```

(continúa…)

47

Igualdad entre caracteres

- Encontrar el nombre de stores cuyos nombres ***no*** comiencen con la letra **B**:

```
select stor_name
from stores
where stor_name not like "B%"

stor_name
----------------------------------------
News & Brews
Doc-U-Mat: Quality Laundry and Books
Eric the Read Books
Fricative Bookshop
Thoreau Reading Discount Chain
```

(continúa…)

48

Igualdad entre caracteres

- Este ejemplo usa un conjunto de valores para encontrar aquellos nombres de *stores* que comiencen con las letras B, D, E, or R:

```
select stor_name
from stores
where stor_name like "[BDER]%"

stor_name
----------------------------------------
Barnum's
Doc-U-Mat: Quality Laundry and Books
Bookbeat
Eric the Read Books
```

(continúa…)

Igualdad entre caracteres

- Este ejemplo usa un rango de caracteres para encontrar los almacenes cuyo nombre comienza con cualquier letra mayúscula entre la D y la H inclusive:

```
select stor_name
from stores
where stor_name like "[D-H]%"

stor_name
----------------------------------------
Doc-U-Mat: Quality Laundry and Books
Eric the Read Books
Fricative Bookshop
```

(continúa…)

Igualdad entre caracteres

- Este ejemplo muestra los *stor_name*, cuyos nombres no comienzan con alguna letra que esté en el rango D - H:

```
select stor_name
from stores
where stor_name not like "[D-H]%"

stor_name
----------------------------------------
Barnum's
News & Brews
Bookbeat
Thoreau Reading Discount Chain
```

(continúa…)

Igualdad entre caracteres

- Este ejemplo usa un rango de caracteres para encontrar todos los *stores* cuyo *stor_id* tenga cuatro caracteres y que los tres primeros sean **706** y el cuarto simplemente un número:

```
select stor_id, stor_name
from stores
where stor_id like "706[0-9]"

stor_id stor_name
------- ------------------------------
7066    Barnum's
7067    News & Brews
```

(continúa…)

Igualdad entre caracteres

- El underscore (_) sirve de comodín para una única posición
- Este ejemplo encuentra los autores que viven en un estado que termina con la letra I o R:

```
select au_lname, state
from authors
where state like "_[IR]"

au_lname               state
--------------------   ----------
Blotchet-Halls         OR
del Castillo           MI
```


Uso de listas

- La palabra clave **in** se utiliza para seleccionar filas con columnas que contengan alguno de los valores declarados en la lista
- El ejemplo usa **in** para encontrar *stores* en el *state* Oregon (OR) **o** California (CA):

```
select stor_name, city, state from stores
where state in ("CA", "OR")

stor_name              city        state
------------------     ----------- -------
Barnum's               Tustin      CA
News & Brews           Los Gatos   CA
Bookbeat               Portland    OR
Fricative Bookshop     Fremont     CA
```

(continúa…)

Uso de listas

- Este ejemplo utiliza **not in** para encontrar *stores* que no están en California o en Oregon:

```
select stor_name, city, state
from stores
where state not in ("CA","OR")

stor_name                      city    state
------------------------------ -------- -----
Doc-U-Mat:Quality ...          Remulade  WA
Eric the Read Books            Seattle   WA
Thoreau Reading Discount Chain Concord   MA
```

55

Conectores lógicos: and / or

- Pueden concatenarse condiciones con los operadores lógicos **and/or**
- **AND:**
  - Conecta dos o más condiciones
  - Es verdadero sólo cuando **todas las condiciones son verdaderas**

El ejemplo encuentra *stores* que viven en la ciudad de Fremont, estado de California:

```
select stor_name, city, state from stores
where state="CA" and city="Fremont"

stor_name              city     state
-------------------    -------- -----
Fricative Bookshop     Fremont  CA
```

56

Conectores lógicos: and / or

- **OR**
  - Conecta dos o más condiciones
  - Es verdadero si **alguna de las condiciones es verdadera**

El ejemplo usa **or** para encontrar los *stores* que están en el estado de California y aquellos que son de la ciudad de Portland:

```
select stor_name, city, state
from stores
where state="CA" or city="Portland"
```

57

Conectores lógicos: and / or

- Cuando use más de un conector lógico, el orden por default en que son evaluados los operadores es **not / and / or**
- El uso de los paréntesis permite forzar el orden de la evaluación

58

Renombrar las cabeceras de columnas

- Puede renombrarse la cabecera de las columnas de tres maneras diferentes
- Puede usarse *new_column_heading = column_name.*
- Ejemplo:

```
select "Número de seguro social" = au_id,
"Apellido" = au_lname from authors
where state = "UT"

Número de seguro social Apellido
----------------------  -----------
899-46-2035             Ringer
998-72-3567             Ringer
```

59

(continúa…)

Renombrar las cabeceras de columnas

- Las otras dos formas consisten en utilizar un espacio o la palabra **as** entre *column_name* y *new_column_heading*
- Ejemplo:

```
select au_id as "Nro de Seguro Social",
au_lname "Apellido" from authors
where state = "UT"

Nro de Seguro Social   Apellido
---------------------- -----------
899-46-2035            Ringer
998-72-3567            Ringer
```

60

Cadenas de caracteres en el resultado de la consulta

- Para clarificar la salida del reporte, puede agregarse una cadena de caracteres a la cláusula **select**:

```
select "Nombre de la librería: ", stor_name
from stores
where stor_id = "7067"

                  stor_name
----------------- --------------------
Nombre de la librería: News & Brews
```

61

Expresiones Numéricas: Operadores Aritméticos

Símbolo	Operación
+	Suma
–	Resta
*	Multiplicación
/	División
%	Módulo

- Las expresiones numéricas pueden usarse:
  - Sobre cualquier columna numérica
  - En cualquier cláusula que permita una expresión
  - Ejemplo:

```
select title_id, advance + price from titles
where type = "business"
```

62

Valores NULL

- Un valor NULL es equivalente a ***desconocido***
  - **null** no implica cero o blanco
  - **null** es un valor especial que significa "información no disponible"
  - **is null** debe ser usado para determinar las columnas que contienen valores nulos
  - " **= null**" puede ser usado, pero no es una sintaxis recomendada (NO ANSI)
- Un valor NULL jamás es igual que otro valor NULL
- Los NULL se ordenan y agrupan juntos
- Las columnas pueden estar definidas para permitir valores nulos

(continúa…)

63

Valores NULL

- Primer ejemplo del uso de **is null**:

```
select title, price
from titles
where type = "popular_comp"
and price is null
```

- Segundo ejemplo (no recomendado - NO ANSI):

```
select title, price
from titles
where type = "popular_comp"
and price = null

title                          price
------------------------------ ------
Net Etiquette                  NULL
```

(continúa…)

64

Valores NULL

- Cálculo involucrando NULLs y resultados NULL

```
select discounttype, highqty, lowqty,
"high - low" = highqty - lowqty
from discounts

discounttype          highqty   lowqty   high-low
--------------------  --------  -------  ---------
Initial Customer      NULL      NULL     NULL
Volume Customer       1000      100      900
Huge Volume Customer  NULL      1001     NULL
Customer Discount     NULL      NULL     NULL
```

65

select / order by

- La cláusula **order by** ordena el resultado de la consulta (en orden ascendente por default)
- Las columnas sobre las que se ordena que aparecen en **order by**, no es necesario que estén seleccionadas
- Cuando se use **order by**, los NULLs aparecen primero
- Ejemplo:

```
select stor_name, state
from stores
order by stor_name
```

- Ejemplo (más de una columna):

```
select stor_name, state
from stores
order by state, stor_name
```

(continúa…)

66

select / order by

- El ejemplo utiliza columnas calculadas (precio de todas las ventas de libros de *type* ***psychology***) como criterio de ordenamiento :

```
select Revenue = total_sales * price, title_id
from titles
where type = "psychology"
order by total_sales * price
```

- El ejemplo muestra que no es necesario incluir en la selección el criterio de ordenamiento:

```
select title, price from titles
where pub_id = "0736"
order by pubdate
```

67

Funciones Agregadas

Función	Significado
count(*)	Cantidad de filas seleccionadas
count(col_name)	Cantidad de valores no nulos en la col.
max(col_name)	Mayor valor en la columna
min(col_name)	Menor valor en la columna
sum(col_name)	Sumatoria de valores de la columna
avg(col_name)	Promedio de valores de la columna

- Ignoran los valores NULL (excepto **count** (*))
- **sum** y **avg** sólo trabajan con datos numéricos
- Si la cláusula **group by** no es usada, sólo una fila es devuelta
- **NO** pueden ser usadas en la cláusula **where**
- Pueden ser aplicadas a todas las filas o a un subconjunto de ellas

68

Función Agregada: count

- La función **count** cuenta el número de filas que cumplen con una condición
- El ejemplo devuelve el número de filas seleccionadas:

```
select count(*) from titles

-----------
18
```

- El ejemplo cuenta el número de filas en que la columna especificada no es NULL:

```
select count(advance)from titles

-------------
16
```

69

Funciones Agregadas: max / min

- La función **max** encuentra el valor más grande en la columna:

```
select max(price) from titles

----------------------
               22.95
```

- La función **min** encuentra el valor más pequeño en la columna:

```
select min(price) from titles

----------------------
               2.99
```

70

Funciones Agregadas: sum / avg

- La función **sum** suma todos los valores de la columna:

```
select sum(total_sales) from titles
  where type = "psychology"

-------------
         9939
```

- La función **avg** suma todos los valores de la columna y lo divide por el número de filas que ha encontrado:

```
select avg(advance) from titles

-------------
    5,962.50
```

71

Funciones Agregadas en una cláusula Select

- Está permitido incluir más de una función agregada en una cláusula **select**:

```
select min(advance), max(advance) from titles
----------------------  -----------
              0.00      15,000.00
```

72

Funciones Agregadas: distinct Keyword

- La palabra clave **distinct** elimina duplicados antes que la función agregada sea aplicada
- La palabra clave **distinct** es:
  - Permitida con **sum**, **avg**, **count**, y **count(col_name)**
  - No permitida con **min**, **max**, y **count(*)**
  - Usada únicamente con nombres de columnas, no con expresiones aritméticas
- Cuántos tipos diferentes de libros se manejan?

```
select count(distinct type)  select count(type)
from titles                  from titles

---------------              ----------------
            6                            18
```

73

Trabajo con valores NULL: isnull

- **isnull** sustituye con un valor real los valores NULL
- **isnull** (*column_which_might_be_null*, *value_to_use_if_null*)
- Esta sustitución toma lugar sólo en el conjunto resultado. **NO** afecta los datos en las tablas

Ejemplo:

```
select avg(price)
from titles

-----------
    14.77
```

Ejemplo que asigna $0.00 a los libros que no poseen precio:

```
select avg(isnull(price,$0.00))
from titles

-----------
    13.13
```

74

select / group by

- La cláusula **group by** organiza los datos en conjuntos que los agrupan tomando como base el contenido de una o varias columnas
  - La cláusula generalmente contiene una función agregada en la *select_list*
  - La función agregada es calculada para cada grupo
  - Todos los valores NULL en la columna del **group by** son tratados como un grupo
- El ejemplo agrupa la filas por tipo y calcula el precio medio para cada tipo:

```
select type, average_price = avg(price)from titles
group by type
```

(continúa…)

75

select / group by

- El agrupamiento puede hacerse por una columna o por una expresión que **no** contiene una función agregada:

```
select title_id, sum(qty) from salesdetail
group by title_id
```

- El agrupamiento no puede hacerse por una columna cabecera derivada
- La cláusula **group by** generalmente contiene todas las columnas y expresiones que aparecen en la *select_list* **no** afectadas por una función agregada

76

group by con una cláusula where

- La cláusula **where**:
  - Elimina las filas **antes** que vayan a los grupos
  - Aplica una condición a la tabla **antes** que se formen los grupos
  - No acepta funciones agregadas (las condiciones de búsqueda son evaluadas de a una fila por vez)
- El ejemplo lista el número total de libros vendidos con un descuento mayor que 50%:

```
select title_id, sum(qty)from salesdetail
where discount > 50
group by title_id
```

77

group by con la cláusula having

- La cláusula **having**:
  - Restringe los grupos
  - Aplica una condición a los grupos **después** que han sido formados
- El ejemplo encuentra los tipos de libros y el precio promedio de cada tipo
  - Sólo mostrar los grupos que el precio promedio es mayor que $12

```
select avg(price), type from titles
group by type
having avg(price) > $12.00
```

- **having** usualmente es utilizada en conjunto con una función agregada

78

Renombrar nombres de columnas y tablas

LOS ALIAS

- Hacer el nombre cifrado de una columna más significativo cuando se la muestra
- Abreviar un nombre que se usa a menudo en una tabla o de una columna
- Hacer más clara una instrucción complicada de SQL
- Distinguir entre dos ocurrencias del mismo nombre de columna o nombre de tabla, en cualquier instrucción SELECT.

79

Renombrar nombres de tablas

- Se la define en la cláusula **FROM**. Después se usa el alias como un calificador tanto en la cláusula **SELECT** como en la **WHERE**.
- Ej: si se desea abreviar con **E** el nombre de la tabla de Empleados, y **C** a la tabla de Clientes con el fin de combinar ciertos empleados con ciertos clientes, se escribirá:

```
SELECT E.*, C.*
   FROM   empleado E, cliente C
   WHERE E.id_zona = C.id_zona
      AND E.comision  >  0.15;
```

80

Renombrar nombres de tablas

- El uso del calificativo **E.** en la cláusula **AND** no es necesario si la tabla de empleados es la única que contiene la columna *comision*.
- Se obtienen todos los campos de las tablas empleado y cliente recuperando los datos de aquellos clientes que viven en la misma zona asignada a los empleados (E.id_zona = C.id_zona) pero considerando solamente las de aquellos empleados que obtengan una comisión por ventas mayor a un 15% (E.comision > 0.15).

81

Subconsultas o selecciones anidadas

- Se usan para obtener la información que se necesita para completar la consulta principal.
- El uso de una subconsulta produce la escritura de una única consulta compuesta en vez de dos o más consultas sencillas → proporciona un método para aumentar la eficacia del usuario.
- En el procesamiento de una instrucción SQL compuesta, se evalúa primero la subconsulta, y a continuación, se aplican los resultados a la consulta principal.

82

Subconsultas o selecciones anidadas

- SQL no impone límites al número de subconsultas que se pueden anidar dentro de una consulta, aun cuando el implementador puede imponer un límite.
- Si se sabe que la subconsulta deberá devolver como máximo un valor, o si se quiere estar seguro de que el resultado es único, entonces la sintaxis es:

```
SELECT   select_list
    FROM tablas
    WHERE condición (predicado de la
                     comparación) (subconsulta);
```

83

Subconsultas o selecciones anidadas

- La sintaxis anterior, devolverá un mensaje de error si **hay más de una fila** que cumpla tal condición.
- Si la subconsulta puede devolver (o devolverá) más de un valor, entonces la sintaxis exige el predicado **IN** en la forma siguiente:

```
SELECT select_list
    FROM tablas
    WHERE nombre_de_la_columna IN
          (subconsulta);
```

84

Subconsultas o selecciones anidadas

- Las consultas anidadas son elegantes pero consumen muchos recursos del servidor de datos.
- Se aconseja en consecuencia,- sobre todo en un ambiente de desarrollo en el que las instrucciones SQL estarán embebidas en las aplicaciones - utilizar las subconsultas complejas solamente en los casos en los que sea imprescindible, para evitar de esta manera, una gran pérdida de *performance* en la respuesta del motor de base de datos.

85

Subconsultas correlacionadas

- Sea la tabla **item** (nro_fac, precio_item) que representa el detalle de una factura. Obtener el **número de factura** y el **precio total** de la misma pero solamente de aquellas facturas en que el precio total sea mayor que el doble del precio del artículo más barato de su detalle.

86

Subconsultas correlacionadas

- Sea la tabla **item** (nro_fac, precio_item) que representa el detalle de una factura. Obtener el **número de factura** y el **precio total** de la misma pero solamente de aquellas facturas en que el precio total sea mayor que el doble del precio del artículo más barato de su detalle.

```
SELECT nro_fac, SUM(precio_item)
      FROM item A
      GROUP BY nro_fac
      HAVING SUM(precio_item) >
            (SELECT 2 * MIN(precio_item)
                  FROM item
                  WHERE nro_fac = A.nro_fac)
```

correlación

87

Subconsultas correlacionadas

```
SELECT nro_fac, SUM(precio_item)
      FROM item A
      GROUP BY nro_fac
      HAVING SUM(precio_item) >
            (SELECT 2 * MIN(precio_item)
                  FROM item
                  WHERE nro_fac = A.nro_fac)
```

correlación

- Una subconsulta que contiene una referencia a una consulta externa es llamada subconsulta correlacionada.
- Su resultado está correlacionado con cada fila individual de la consulta principal.
- Una subconsulta está correlacionada cuando sus valores dependen del valor de una variable recibida desde el SELECT externo.

88

LAS UNIONES (join)

- Una unión en SQL es una consulta en la que los datos se recuperan de dos o más tablas. La finalidad de unir, es recuperar información que no está en una única tabla.
  - Uniones equidistantes.
  - Uniones NO equidistantes.
  - Uniones naturales.
  - Uniones externas.
  - Uniones de múltiples tablas.
  - Uniones de tablas consigo mismas.

89

LAS UNIONES EQUIDISTANTES

- La unión equidistante en SQL se representa por:

```
SELECT tabla1.*, tabla2.*
      FROM tabla1, tabla2
      WHERE (conjunto de condiciones);
```

- El conjunto de condiciones es el grupo de ***comparaciones de igualdad*** entre las columnas de la tabla1 y las columnas de la tabla2. Las columnas comparadas, deben ser del mismo tipo y tamaño.
- Si el conjunto de condiciones está vacío, vale decir que no exista cláusula **WHERE**, entonces el resultado, es el producto cartesiano de las tablas:

```
SELECT * FROM tabla1, tabla2;
```

90

LAS UNIONES EQUIDISTANTES

- Si se considera que la tabla1 tiene las columnas A, B y C, y la tabla2, que contiene las columnas C, D y E siguientes:

SELECT * FROM tabla1, tabla2;

Tabla1			Tabla2		
A	B	C	C	D	E
1	1	1	1	4	5
1	2	1	2	6	7
2	2	1	3	8	9
1	2	2			

TABLA3					
T1A	T1B	T1C	T2C	T2D	T2E
1	1	1	1	4	5
1	1	1	2	6	7
1	1	1	3	8	9
1	2	1	1	4	5
1	2	1	2	6	7
1	2	1	3	8	9
2	2	1	1	4	5
2	2	1	2	6	7
2	2	1	3	8	9
1	2	2	1	4	5
1	2	2	2	6	7
1	2	2	3	8	9

91

LAS UNIONES EQUIDISTANTES con una condición

- Se utilizan para navegar entre las tablas. Se obtiene una unión equidistante, cuando existen dos tablas que tienen una columna (o más) en común, y el conjunto de condiciones de la cláusula **WHERE** indicará que los valores de las columnas en común son **iguales**.
- La o las columna en común, se denominan columnas de unión. La sintaxis para obtener tal resultado es:

SELECT tabla1.*, tabla2.* FROM tabla1, tabla2
WHERE tabla1.columna_de_union = tabla2.columna_de_union;

- Suele ser necesario usar el nombre de la tabla con cada una de las columnas de unión para especificar de donde proceden.

92

LAS UNIONES EQUIDISTANTES con una condición

- Los resultados serán:

SELECT * FROM tabla1, tabla2
WHERE tabla1.c = tabla2.c;

Tabla1			Tabla2		
A	B	C	C	D	E
1	1	1	1	4	5
1	2	1	2	6	7
2	2	1	3	8	9
1	2	2			

TABLA4					
T1A	T1B	T1C	T2C	T2D	T2E
1	1	1	1	4	5
1	2	1	1	4	5
2	2	1	1	4	5
1	2	2	2	6	7

93

LAS UNIONES NATURALES

- O simplemente UNION. Es una unión equidistante con la columna común de dos tablas no repetida:

Tabla1			Tabla2		
A	B	C	C	D	E
1	1	1	1	4	5
1	2	1	2	6	7
2	2	1	3	8	9
1	2	2			

TABLA 5				
A	B	C	D	E
1	1	1	4	5
1	2	1	4	5
2	2	1	4	5
1	2	2	6	7

- Se puede obtener la unión natural de las tablas1 y 2 con:

SELECT tabla1.*, D, E FROM tabla1, tabla2
WHERE tabla1.C = tabla2.C;

94

LAS UNIONES EQUIDISTANTES

Vendedores	
Vendedor	Zona
Baudino	Sur
Scrutiño	Guadalupe
Gauderio	Costanera
Deyro	Guadalupe

Clientes	
Cliente	Zona
Cortese	Guadalupe
Anchaval	Costanera
Ingaramo	Sur
Rossi	Guadalupe
Ortiz	Sur
López	Sur

SELECT vendedores.*, clientes.*
FROM vendedores, clientes
WHERE vendedores.zona = clientes.zona
ORDER BY zona;

Vendedor	Zona	Zona	Cliente
Baudino	Sur	Sur	Ortiz
Baudino	Sur	Sur	Ingaramo
Baudino	Sur	Sur	López
Scrutiño	Guadalupe	Guadalupe	Cortese
Deyro	Guadalupe	Guadalupe	Rossi
Gauderio	Costanera	Costanera	Anchaval

95

LAS UNIONES EQUIDISTANTES

- Sin la columna duplicada:

Vendedores	
Vendedor	Zona
Baudino	Sur
Scrutiño	Guadalupe
Gauderio	Costanera
Deyro	Guadalupe

Clientes	
Cliente	Zona
Cortese	Guadalupe
Anchaval	Costanera
Ingaramo	Sur
Rossi	Guadalupe
Ortiz	Sur
López	Sur

SELECT vendedores.*, clientes.cliente
FROM vendedores, clientes
WHERE vendedores.zona = clientes.zona
ORDER BY zona;

Vendedor	Zona	Zona	Cliente
Baudino	Sur	Sur	Ortiz
Baudino	Sur	Sur	Ingaramo
Baudino	Sur	Sur	López
Scrutiño	Guadalupe	Guadalupe	Cortese
Deyro	Guadalupe	Guadalupe	Rossi
Gauderio	Costanera	Costanera	Anchaval

96

LAS UNIONES NO EQUIDISTANTES

- La columna de unión de una tabla **no es igual** a la columna de unión correspondiente de la otra tabla:

```
SELECT tabla1.columnas, tabla2.columnas
    FROM tabla1, tabla2
    WHERE tabla1.columna_de_union
(cualquier operador de comparación excepto =)
            tabla2.columna_de_union;
```

97

LAS UNIONES NO EQUIDISTANTES

```
SELECT vendedores.*, clientes.*
    FROM  vendedores, clientes
    WHERE  NOT (vendedores.zonas = clientes.zonas);
```

- **La cláusula WHERE también se podría expresar:**

```
    WHERE vendedores.zonas <> clientes.zonas;
```

El resultado haría que se correspondiesen los vendedores con los clientes de forma impredecible, con la excepción de que no habría filas en las que la zona del vendedor fuese la misma que la zona del cliente.

98

CONDICIONES ADICIONALES EN CONSULTAS DE UNIONES

- Se usa **AND** para agregar condiciones adicionales a la condición de vinculación entre las tablas. Dentro de estos adicionales puede utilizarse cualquier conector lógico.

```
SELECT empleado.nombre, cliente.nombre
    FROM empleado, cliente
    WHERE empleado.zona = cliente.zona
        AND (empleado.nom_funcion = "vendedor"
            OR
                empleado.nom_funcion = "gerente");
```

99

UNIONES ENTRE DOS O MAS TABLAS

- Se debe indicar las tablas que hay que unir en la cláusula **FROM**, y usando el **AND** para añadir las condiciones de ***join*** más cualquier otra que sea necesaria.

```
SELECT nombres_de_las_columnas
    FROM tabla1, tabla2, . . . , tablaN
    WHERE (condicion1)
        AND (condicion2)  ……
        AND (condicionN);
```

- No existe límite teórico respecto al número posible de uniones.
- **En una consulta que trabaje con n tablas, se requieren al menos n-1 condiciones de join.**

100

UNIONES ENTRE DOS O MAS TABLAS

- **Uniones consigo misma**
- **Uniones externas**

101