Bases de Datos I
22/09/2020
Resumen Cap 2 22/09
IS-501
José Inestroza

Capítulo 02 SQL Complete Reference
Conceptos básicos de SQL
Declaraciones
El cuerpo principal de SQL consta de aproximadamente 40 declaraciones. Cada declaración solicita una acción específica del DBMS, como crear una nueva tabla, recuperar datos o insertar nuevos datos en la base de datos. Cada declaración SQL comienza con un verbo, una palabra clave que describe lo que hace la declaración. CREAR, INSERTAR, ELIMINAR y COMPROMETER son verbos típicos. Cada cláusula también comienza con una palabra clave, como WHERE, FROM, INTO y HAVING. Algunas cláusulas son opcionales; otros son necesarios.

Tabla de declaraciones

Statement	Description
Data Manipulation	
SELECT	Retrieves data from the database
INSERT	Adds new rows of data to the database
UPDATE	Modifies existing database data
MERGE	Conditionally inserts/updates/deletes new and existing rows
DELETE	Removes rows of data from the database
Data Definition	
CREATE TABLE	Adds a new table to the database
DROP TABLE	Removes a table from the database
ALTER TABLE	Changes the structure of an existing table
CREATE VIEW	Adds a new view to the database
DROP VIEW	Removes a view from the database
CREATE INDEX	Builds an index for a column
DROP INDEX	Removes the index for a column
CREATE SCHEMA	Adds a new schema to the database
DROP SCHEMA	Removes a schema from the database
CREATE DOMAIN	Adds a new data value domain
ALTER DOMAIN	Changes a domain definition
DROP DOMAIN	Removes a domain from the database
Access Control	
GRANT	Grants user access privileges
REVOKE	Removes user access privileges
CREATE ROLE	Adds a new role to the database
GRANT ROLE	Grants role containing user access privileges
DROP ROLE	Removes a role from the database
Transaction Control	
COMMIT	Ends the current transaction
ROLLBACK	Aborts the current transaction
SET TRANSACTION	Defines data access characteristics of the current transaction
START TRANSACTION	Explicitly starts a new transaction
SAVEPOINT	Establishes a recovery point for a transaction

Statement	Description
Programmatic SQL	
DECLARE	Defines a cursor for a query
EXPLAIN	Describes the data access plan for a query
OPEN	Opens a cursor to retrieve query results
FETCH	Retrieves a row of query results
CLOSE	Closes a cursor
PREPARE	Prepares a SQL statement for dynamic execution
EXECUTE	Executes a SQL statement dynamically
DESCRIBE	Describes a prepared query

Tipos de datos
En la actualidad, los productos DBMS comerciales pueden procesar una gran variedad de datos y existe una diversidad considerable en los tipos de datos particulares admitidos en las diferentes marcas de DBMS. Los tipos de datos típicos incluyen los siguientes

• **Enteros** Las columnas que contienen este tipo de datos suelen almacenar recuentos, cantidades, edades, etc. Las columnas de números enteros también se utilizan con frecuencia para contener números de identificación, como clientes, empleados y números de pedido.
• **Números decimales** Las columnas con este tipo de datos almacenan números que tienen partes fraccionarias y que deben calcularse con exactitud, como tasas y porcentajes. También se utilizan con frecuencia para almacenar cantidades de dinero.
• **Números de coma flotante** Las columnas con este tipo de datos se utilizan para almacenar números científicos que se pueden calcular aproximadamente, como pesos y distancias. Los números de coma flotante pueden representar un rango de valores mayor que los números decimales, pero pueden producir errores de redondeo en los cálculos.
• **Cadenas de caracteres de longitud fija** Las columnas que contienen este tipo de datos suelen almacenar cadenas de caracteres que siempre tienen la misma longitud, como códigos postales, abreviaturas de estado / provincia, descripciones breves, etc. Siempre que la cadena que se va a almacenar sea más pequeña que la longitud definida para una columna de longitud fija, se rellena con espacios para que se ajuste a la longitud exacta de almacenamiento.
• **Cadenas de caracteres de longitud variable** Este tipo de datos permite que una columna almacene cadenas de caracteres que varían en longitud de una fila a otra, hasta una longitud máxima. (El estándar SQL1 solo permitía cadenas de caracteres de longitud fija, que son más fáciles de procesar para el DBMS pero pueden desperdiciar un espacio considerable).
El tipo de datos suele almacenar nombres de personas y empresas, direcciones, descripciones, etc. A diferencia de las cadenas de caracteres de longitud fija, las cadenas de longitud variable no se rellenan con espacios; el número exacto de caracteres proporcionados se almacena junto con
la longitud de la cadena de datos.
• **Cantidades monetarias** Algunos productos SQL admiten un tipo MONEY o CURRENCY, que generalmente se almacena como un número decimal o de coma flotante. Tener un tipo de dinero distinto permite al DBMS formatear correctamente las cantidades de dinero cuando se muestran. Sin embargo, el estándar SQL no especifica tal tipo de datos.

• **Fechas y horas** El soporte para valores de fecha / hora también es común en los productos SQL, aunque los detalles pueden variar considerablemente de un producto a otro, en gran parte porque los proveedores implementaron estos tipos de datos antes de que se desarrollara el estándar SQL. Generalmente se admiten varias combinaciones de fechas, horas, marcas de tiempo, intervalos de tiempo y aritmética de fecha / hora. El estándar SQL incluye una especificación elaborada para los tipos de datos DATE, TIME, TIMESTAMP e INTERVAL, incluida la compatibilidad con la hora.
• **Datos booleanos** Algunos productos SQL, como Microsoft SQL Server, admiten valores lógicos (VERDADERO o FALSO) como un tipo explícito y algunos permiten operaciones lógicas (comparación, Y / O, etc.) en los datos almacenados dentro de declaraciones SQL.
• Objetos de caracteres grandes El estándar SQL: 1999 agregó el tipo de datos CLOB que admite el almacenamiento de cadenas de caracteres grandes, hasta una cantidad específica con una longitud máxima típica en el rango de varios gigabytes. Esto permite que la base de datos almacene documentos completos, descripciones de productos, documentos técnicos, currículums y datos de texto no estructurados similares. Varias bases de datos basadas en SQL admiten tipos de datos patentados (agregados antes del estándar SQL: 1999) capaces de almacenar cadenas de texto largas (típicamente hasta 32,000 o 65,000 caracteres, y en algunos casos incluso más grandes). El DBMS generalmente restringe el uso de columnas de caracteres grandes en consultas y búsquedas interactivas.
• **Objetos binarios grandes** El estándar SQL: 1999 también agregó el tipo de datos BLOB que admite el almacenamiento de secuencias de bytes de longitud variable no estructuradas. Las columnas que contienen estos datos se utilizan para almacenar imágenes de video comprimidas, código ejecutable y otros tipos de datos no estructurados. Antes de la publicación del estándar, los proveedores implementaron sus propias soluciones patentadas, como los tipos de datos IMAGE de SQL Server y LONG RAW de Oracle, que pueden almacenar hasta 2 gigabytes de datos.
• **Caracteres no romanos** A medida que las bases de datos crecieron para admitir aplicaciones globales, DBMS
los proveedores agregaron soporte para cadenas de longitud fija y longitud variable de caracteres multibyte que se utilizan para representar Kanji y otros caracteres asiáticos y árabes utilizando tipos patentados como los tipos de datos GRAPHIC y VARGRAPHIC en SQL Server. El estándar ANSI / ISO ahora especifica versiones de juegos de caracteres nacionales de los diversos tipos de datos de caracteres (NCHAR, NVARCHAR y NCLOB). Si bien la mayoría de las bases de datos modernas admiten el almacenamiento y la recuperación de dichos caracteres (a menudo se utiliza la convención de UNICODE para representarlos), la compatibilidad con la búsqueda y clasificación de estos tipos varía ampliamente.

Data Type	Abbreviation(s)	Description
CHARACTER(*len*)	CHAR	Fixed-length character strings
CHARACTER VARYING(*len*)	CHAR VARYING, VARCHAR	Variable-length character strings
CHARACTER LARGE OBJECT(*len*)	CLOB	Large fixed-length character strings
NATIONAL CHARACTER(*len*)	NATIONAL CHAR, NCHAR	Fixed-length national character strings
NATIONAL CHARACTER VARYING(*len*)	NATIONAL CHAR VARYING, NCHAR	Variable-length national character strings
NATIONAL CHARACTER LARGE OBJECT(*len*)	NCLOB	Large variable-length national character strings
BIT(*len*)		Fixed-length bit strings
BIT VARYING(*len*)		Variable-length bit strings
INTEGER	INT	Integers
SMALLINT		Small integers
NUMERIC(*precision*, *scale*)		Decimal numbers
DECIMAL(*precision*, *scale*)	DEC	Decimal numbers
FLOAT(*precision*)		Floating point numbers
REAL		Low-precision floating point numbers
DOUBLE PRECISION		High-precision floating point numbers
DATE		Calendar dates
TIME(*precision*)		Clock times
TIME WITH TIME ZONE (*precision*)		Clock times with time zones
TIMESTAMP(*precision*)		Dates and times
TIMESTAMP WITH TIME ZONE (*precision*)		Dates and times with time zones
INTERVAL		Time intervals
XML(*type modifier* [*secondary type modifier*])		Character data formatted as Extensible Markup Language (XML)

Se pueden especificar fechas y horas específicas como constantes de cadena, y se admite la aritmética de fechas. A continuación, se muestra un ejemplo de una consulta válida que utiliza fechas de DB2, asumiendo que la columna HIRE_DATE contiene datos de DATE:

```
SELECT NAME, HIRE_DATE
  FROM SALESREPS
 WHERE HIRE_DATE >= '05/30/2007' + 15 DAYS;
```

SQL Server se introdujo con un solo tipo de datos de fecha / hora, llamado DATETIME, que se parece mucho al tipo de datos DB2 TIMESTAMP. Si HIRE_DATE contenía datos DATETIME, SQL Server podría aceptar esta versión de la consulta (sin la aritmética de fecha):

```
SELECT NAME, HIRE_DATE
  FROM SALESREPS
 WHERE HIRE_DATE >= '06/14/2007 12:00AM';
```

Dado que en la consulta no se especifica una hora específica el 14 de junio de 2007, SQL Server tiene como valor predeterminado la medianoche de esa fecha. Por lo tanto, la consulta de SQL Server realmente significa

```
SELECT NAME, HIRE_DATE
  FROM SALESREPS
 WHERE HIRE_DATE >= '06/14/2007 12:00AM';
```

SQL Server también admite aritmética de fechas a través de un conjunto de funciones integradas. Por lo tanto, la consulta de estilo DB2 también se puede especificar de esta manera:

```
SELECT NAME, HIRE_DATE
  FROM SALESREPS
 WHERE HIRE_DATE >= DATEADD(DAY, 15, '05/30/2007')
```

Oracle también admite aritmética de fecha limitada, por lo que la consulta de estilo DB2 también se puede especificar, pero sin la palabra clave DAYS:

```
SELECT NAME, HIRE_DATE
  FROM SALESREPS
 WHERE HIRE_DATE >= '30-MAY-07' + 15;
```

Afortunadamente, con el advenimiento de la conversión del año 2000, la mayoría de los proveedores de DBMS agregaron soporte universal para fechas en declaraciones SQL con años de cuatro dígitos en un formato estándar AAAAMM-DD, que usamos para la mayoría de los ejemplos de este libro. En el caso de Oracle, el formato predeterminado sigue siendo el que se muestra en los ejemplos anteriores, pero se puede cambiar en la base de datos o en la sesión del usuario con un simple comando. Si está utilizando Oracle y prueba alguno de los ejemplos de este libro, simplemente ingrese este comando para cambiar su fecha predeterminada
formato:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD';
```

Se debe tener cuidado al realizar consultas que busquen coincidencias de fechas exactas utilizando el operador igual (=), y las fechas tienen componentes de tiempo almacenados en ellas. Considere el siguiente ejemplo:

```
SELECT NAME, HIRE_DATE
  FROM SALESREPS
 WHERE HIRE_DATE = '06/14/2007';
```

Constantes

En algunas sentencias SQL, un valor de datos numérico, de caracteres o de fecha debe expresarse en forma de texto. Por ejemplo, en esta declaración INSERT que agrega un vendedor a la base de datos:

```
INSERT INTO SALESREPS (EMPL_NUM, NAME, QUOTA, HIRE_DATE, SALES)
     VALUES (115, 'Dennis Irving', 175000.00, '2008-06-21', 0.00);
```

El valor de cada columna en la fila recién insertada se especifica en la cláusula VALUES. Los valores de datos constantes también se utilizan en expresiones, como en esta instrucción SELECT:

```
SELECT CITY
  FROM OFFICES
 WHERE TARGET > (1.1 * SALES) + 10000.00;
```

Constantes numéricas
Las constantes enteras y decimales (también llamadas literales numéricas exactas) se escriben como números decimales ordinarios en declaraciones SQL, con un signo más o menos a la izquierda opcional:
21-375 2000,00 +497500,8778
No debe poner una coma entre los dígitos de una constante numérica y no todos los dialectos SQL permiten el signo más inicial, por lo que es mejor evitarlo. Para los datos monetarios, la mayoría de las implementaciones de SQL simplemente usan constantes enteras o decimales, aunque algunas permiten la constante
a especificar con un símbolo de moneda:
$0.75 $5000.00 $-567.89
Las constantes de coma flotante (también llamadas literales numéricas aproximadas) se especifican utilizando la notación E que se encuentra comúnmente en lenguajes de programación como C y FORTRAN. Aquí hay algunas constantes de punto flotante SQL válidas:
1.5E3 -3.14159E1 2.5E-7 0.783926E21
La E se lee "multiplicada por diez a la potencia de", por lo que la primera constante se convierte en "1,5 veces diez a la tercera potencia", o 1500.

Constantes de cadena
El estándar ANSI / ISO especifica que las constantes de SQL para datos de caracteres deben incluirse entre comillas simples ('...'), Como en estos ejemplos:
'Jones, John J.' 'Nueva York' 'Occidental' Si se va a incluir una comilla simple en el texto constante, se escribe dentro de la constante como
dos caracteres consecutivos de comillas simples. Por lo tanto, este valor constante:
"No puedo" se convierte en la cadena de siete caracteres "No puedo". Algunas implementaciones de SQL, como SQL Server, aceptan cadenas de caracteres entre comillas dobles ("..."):
"Jones, John J." "New York" "Western" Desafortunadamente, las comillas dobles pueden plantear problemas de portabilidad con otros productos SQL. El estándar SQL proporciona la capacidad adicional de especificar constantes de cadena de un conjunto de caracteres nacionales específico (por ejemplo, francés o alemán) o de un conjunto de caracteres definido por el usuario. Las capacidades del juego de caracteres definido por el usuario no se han implementado normalmente en los productos SQL convencionales.

Constantes de fecha y hora

Format Name	Date Format	Date Example	Time Format	Time Example
American	mm/dd/yyyy	5/19/2008	hh:mm am/pm	2:18 PM
European	dd.mm.yyyy	19.5.2008	hh.mm.ss	14.18.08
Japanese	yyyy-mm-dd	2008-5-19	hh:mm:ss	14:18:08
ISO	yyyy-mm-dd	2008-5-19	hh.mm.ss	14.18.08

y aquí hay algunas constantes de tiempo legales:

15:30:25 3:30:25 PM 3:30:25 pm 3 PM
Las fechas y horas de Oracle también se escriben como constantes de cadena, usando este formato:
15-MAR-90
También puede utilizar la función TO_DATE () incorporada de Oracle para convertir constantes de fecha escritas en otros formatos, como en este ejemplo:

```
SELECT NAME, AGE
  FROM SALESREPS
 WHERE HIRE_DATE = TO_DATE('JUN 14 2007', 'MON DD YYYY');
```

El estándar SQL2 especifica un formato para las constantes de fecha y hora, excepto que las constantes de tiempo se escriben con dos puntos en lugar de períodos que separan las horas, minutos y segundos. El tipo TIMESTAMP estándar de SQL, que no se muestra en la tabla, tiene el formato aaaa-mm-dd-hh.mm.ss.nnnnnn, por ejemplo, **"1960-05-19-14.18.08.048632 "** representa el 19/5/60 aproximadamente a las 2:18 p.m.

Function	Returns
BIT_LENGTH (*string*)	The number of bits in a bit string
CAST (*value* AS *data_type*)	The value, converted to the specified data type (e.g., a date converted to a character string)
CHAR_LENGTH (*string*)	The length of a character string
CONVERT (*string* USING *conv*)	A string converted as specified by a named conversion function
CURRENT_DATE	The current date
CURRENT_TIME (*precision*)	The current time, with the specified *precision*
CURRENT_TIMESTAMP (*precision*)	The current date and time, with the specified *precision*
EXTRACT (*part* FROM *source*)	The specified part (DAY, HOUR, etc.) from a DATETIME value
LOWER (*string*)	A string converted to all lowercase letters
OCTET_LENGTH (*string*)	The number of 8-bit bytes in a character string
POSITION (*target* IN *source*)	The position where the *target* string appears within the *source* string
SUBSTRING (*source* FROM *n* FOR *len*)	A portion of the *source* string, beginning at the *n*th character, for a length of *len*
TRANSLATE (*string* USING *trans*)	A string translated as specified by a named translation function
TRIM (BOTH *char* FROM *string*)	A string with both leading and trailing occurrences of *char* trimmed off
TRIM (LEADING *char* FROM *string*)	A string with any leading occurrences of *char* trimmed off
TRIM (TRAILING *char* FROM *string*)	A string with any trailing occurrences of *char* trimmed off
UPPER (*string*)	A string converted to all uppercase letters

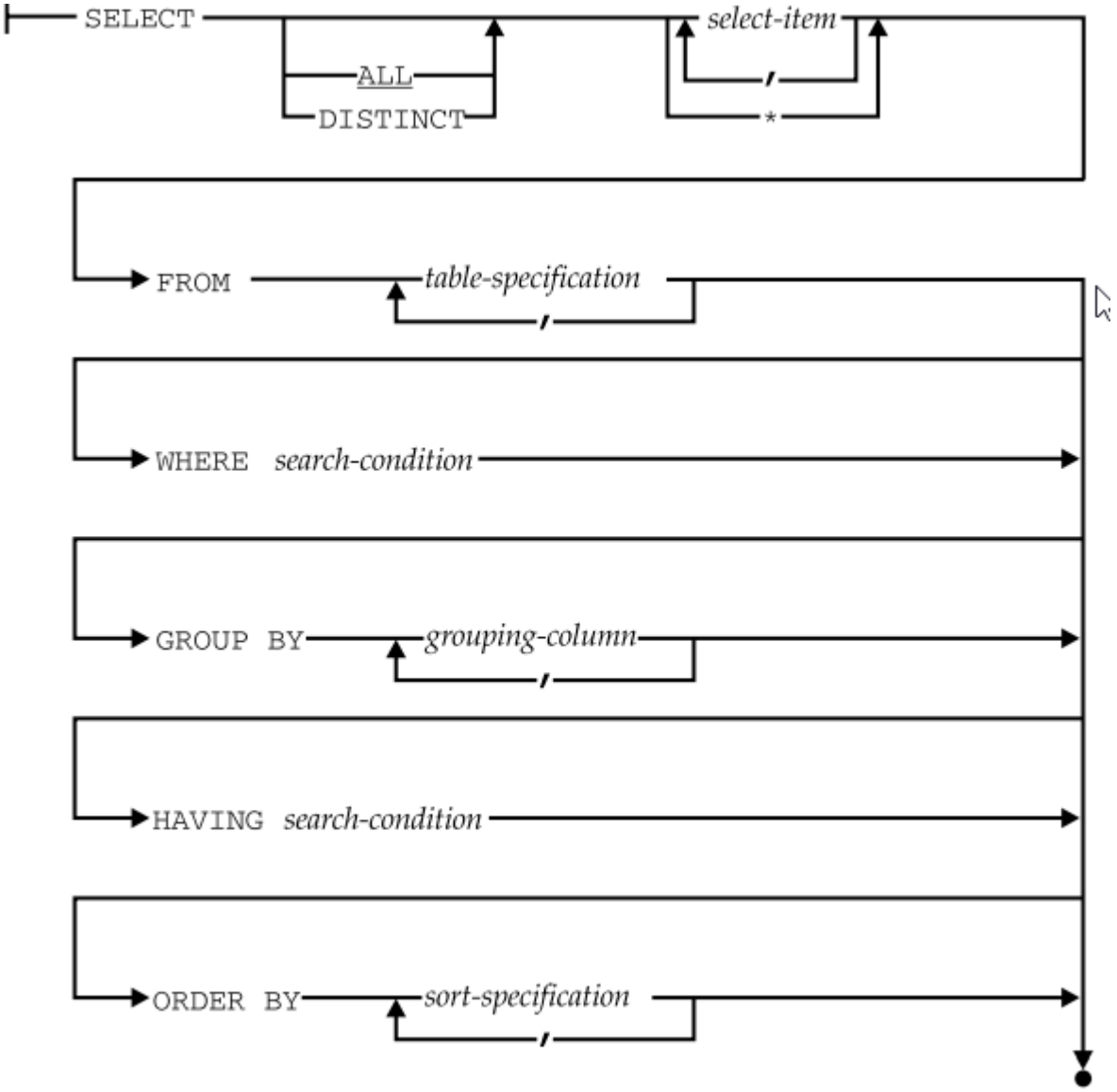
Resumen
Este capítulo describe los elementos básicos de SQL. La estructura básica de SQL se puede resumir de la siguiente manera:
• El SQL de uso común incluye alrededor de 30 declaraciones, cada una de las cuales consta de un verbo y una o más cláusulas. Cada declaración realiza una función única y específica.
• Las bases de datos basadas en SQL pueden almacenar varios tipos de datos, incluidos texto, números enteros, números decimales, números de coma flotante y, por lo general, muchos más tipos de datos específicos del proveedor.
• Las declaraciones SQL pueden incluir expresiones que combinen nombres de columnas, constantes y funciones integradas, utilizando aritmética y otros operadores específicos del proveedor.

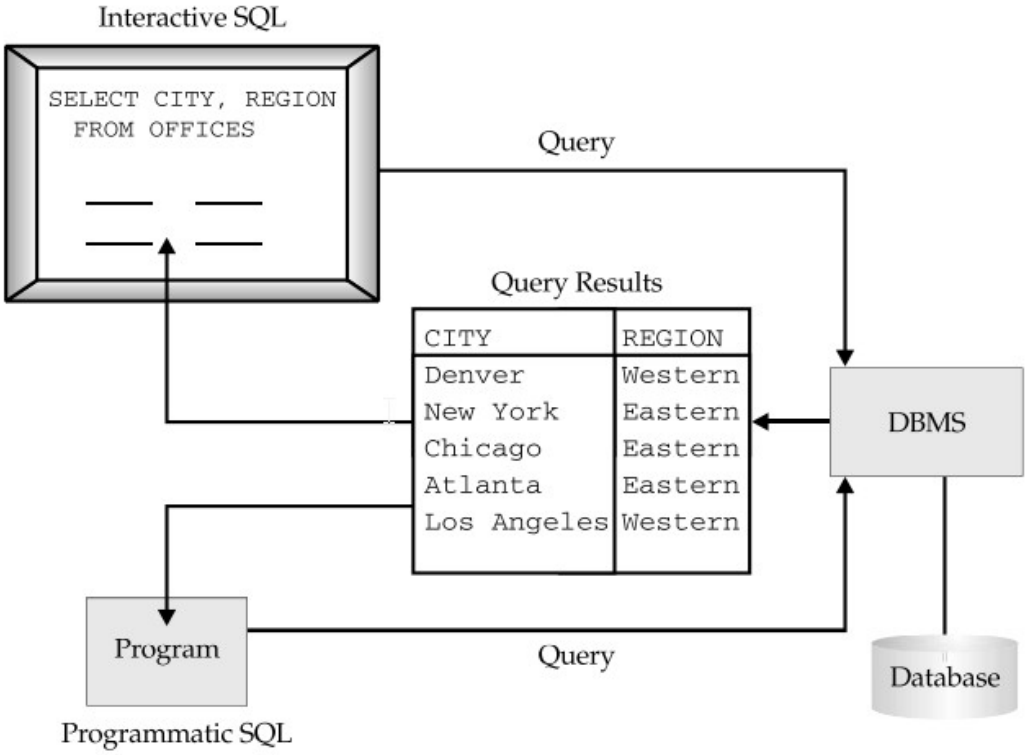
• Las variaciones en los tipos de datos, las constantes y las funciones integradas hacen que la portabilidad de las declaraciones SQL sea más difícil de lo que parece al principio.
• Los valores NULL proporcionan una forma sistemática de manejar datos faltantes o inaplicables en SQL.

Consultas simples

La declaración SELECT

• La cláusula SELECT enumera los elementos de datos que se recuperarán mediante la instrucción SELECT. Los elementos pueden ser columnas de la base de datos o columnas que SQL calculará mientras realiza la consulta. La cláusula SELECT se describe en la siguiente sección.
• La cláusula FROM enumera las tablas y vistas que contienen los datos que debe recuperar la consulta. (Las vistas se analizan en detalle en el Capítulo 14.) Las consultas que extraen sus datos de una sola tabla se describen en este capítulo.
• La cláusula WHERE le dice a SQL que incluya solo ciertas filas de datos en los resultados de la consulta. Se utiliza una condición de búsqueda para especificar las filas deseadas. Los usos básicos de la cláusula WHERE se describen en la sección “Selección de filas (cláusula WHERE)” más adelante en este capítulo.
• La cláusula GROUP BY especifica una consulta de resumen. En lugar de producir una fila de resultados de consulta para cada fila de datos en la base de datos, una consulta de resumen agrupa filas similares y luego produce una fila de resumen de resultados de consulta para cada
grupo.
• La cláusula **HAVING** le dice a SQL que incluya solo ciertos grupos producidos por la cláusula GROUP BY en los resultados de la consulta. Al igual que la cláusula WHERE, utiliza una condición de búsqueda para especificar los grupos deseados.
• La cláusula ORDER BY ordena los resultados de la consulta en función de los datos de una o más columnas. Si se omite, los resultados de la consulta no se ordenan. La cláusula ORDER BY se describe en la sección "Clasificación de resultados de consultas (cláusula ORDER BY)".


SELECT
ALL
DISTINCT
select-item
,
*
FROM
table-specification
,
WHERE search-condition
GROUP BY
grouping-column
,
HAVING search-condition
ORDER BY
sort-specification
,


Interactive SQL
SELECT CITY, REGION
FROM OFFICES
Query
Query Results
CITY
REGION
Denver
Western
New York
Eastern
Chicago
Eastern
Atlanta
Eastern
Los Angeles
Western
DBMS
Program
Query
Database
Programmatic SQL

Columnas calculadas
Si intenta sumar, restar, multiplicar o dividir columnas que contienen datos de texto, SQL informará un error. Esta consulta muestra una columna calculada simple:
Enumere la ciudad, la región y la cantidad por encima o por debajo del objetivo para cada oficina.

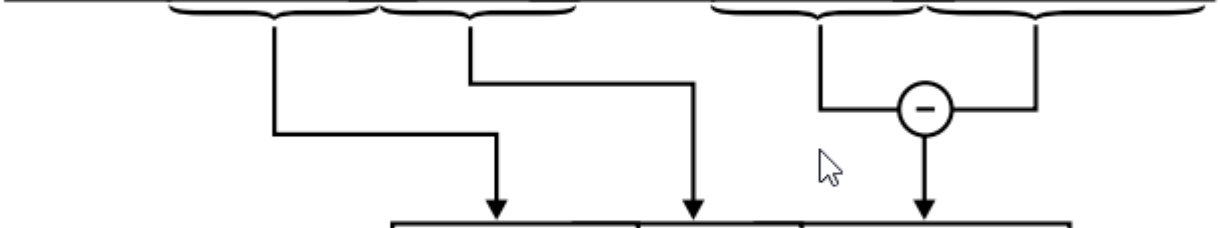
```
SELECT CITY, REGION, (SALES - TARGET)
  FROM OFFICES;
```

Muestre el valor del inventario para cada producto. (Solo se muestran las primeras 8 filas del conjunto de resultados).

```
SELECT MFR_ID, PRODUCT_ID, DESCRIPTION, (QTY_ON_HAND * PRICE)
  FROM PRODUCTS;
```

OFFICES Table

OFFICE	CITY	REGION	MGR	TARGET	SALES
22	Denver	Western	108	$300,000.00	$186,042.00
11	New York	Eastern	106	$575,000.00	$692,637.00
12	Chicago	Eastern	104	$800,000.00	$735,042.00
13	Atlanta	Eastern	NULL	$350,000.00	$367,911.00
21	Los Angeles	Western	108	$725,000.00	$835,915.00



Query Results

CITY	REGION	SALES-TARGET
Denver	Western	-$113,958.00
New York	Eastern	$117,637.00
Chicago	Eastern	-$ 64,958.00
Atlanta	Eastern	$ 17,911.00
Los Angeles	Western	$110,915.00

Muéstreme el resultado si aumenté la cuota de cada vendedor en un 3 por ciento de sus ventas del año hasta la fecha.

```
SELECT NAME, QUOTA, (QUOTA + (.03*SALES))
  FROM SALESREPS;
```

Indique el nombre, mes y año de contratación de cada vendedor. (Para las bases de datos Oracle, se debe utilizar la función TO_CHAR en lugar de las funciones MES y AÑO).

```
SELECT NAME, MONTH(HIRE_DATE), YEAR(HIRE_DATE)
  FROM SALESREPS;
```

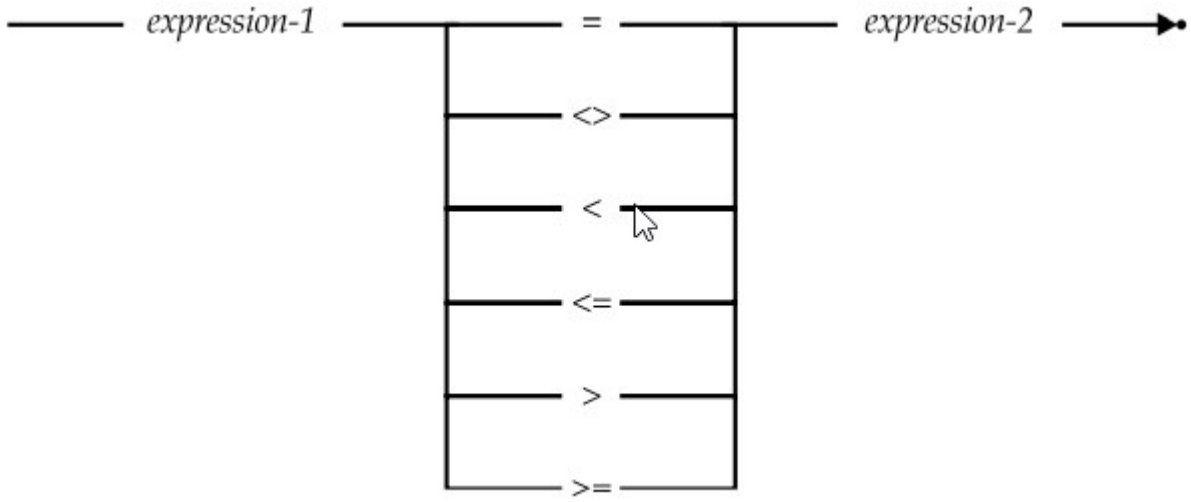
Enumere las ventas de cada ciudad.

```
SELECT CITY, 'has sales of', SALES
  FROM OFFICES;
```

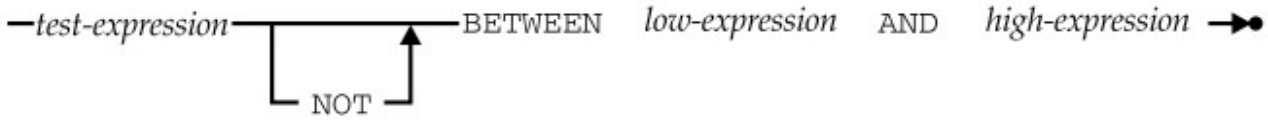
Filas duplicadas (DISTINCT)
Si una consulta incluye la clave principal de una tabla en su lista de selección, entonces cada fila de resultados de la consulta será única (porque la clave principal tiene un valor diferente en cada fila). Si la clave principal no se incluye en los resultados de la consulta, pueden producirse filas duplicadas. Por ejemplo, suponga que hizo esta solicitud:
Enumere los números de empleados de todos los gerentes de oficinas de ventas.

```
SELECT DISTINCT MGR
  FROM OFFICES;
```

Condiciones de búsqueda
1. La prueba de comparación (=, <>, <, <=,>,> =)



2. La prueba de rango (BETWEEN)



La prueba de rango verifica si un valor de datos se encuentra entre dos valores especificados. Implica tres expresiones SQL. La primera expresión define el valor a probar; la segunda y tercera expresiones definen los extremos inferior y superior del rango a comprobar. Los tipos de datos de las tres expresiones deben ser comparables. Este ejemplo muestra una prueba de rango típica:
Encuentre pedidos realizados en el último trimestre de 2007

```
SELECT ORDER_NUM, ORDER_DATE, MFR, PRODUCT, AMOUNT
  FROM ORDERS
 WHERE ORDER_DATE BETWEEN '2007-10-01' AND '2007-12-31';
```

Encuentre los pedidos que caen en varios rangos de cantidad.

```
SELECT ORDER_NUM, AMOUNT
  FROM ORDERS
 WHERE AMOUNT BETWEEN 20000.00 AND 29999.99;
```

La versión negada de la prueba de rango (NO ENTRE) busca valores que se encuentran fuera del rango, como en este ejemplo:
Enumere a los vendedores cuyas ventas no estén entre el 80 y el 120 por ciento de la cuota.

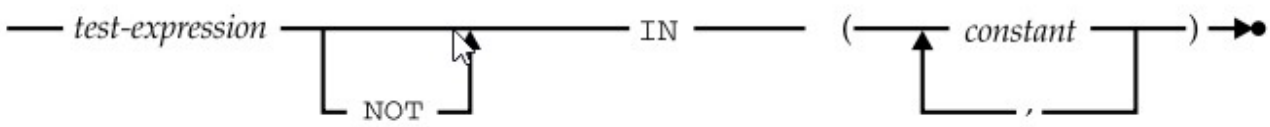
```
SELECT NAME, SALES, QUOTA
  FROM SALESREPS
 WHERE SALES NOT BETWEEN (.8 * QUOTA) AND (1.2 * QUOTA);
```

• Si la expresión de prueba produce un valor NULO, o si ambas expresiones que definen el rango producen valores NULL, entonces la prueba BETWEEN devuelve un resultado NULO.
• Si la expresión que define el extremo inferior del rango produce un valor NULO, la prueba BETWEEN devuelve FALSE si el valor de la prueba es mayor que el límite superior y NULL en caso contrario.
• Si la expresión que define el extremo superior del rango produce un valor NULO, entonces la prueba BETWEEN devuelve FALSE si el valor de prueba es menor que el límite inferior y NULL en caso contrario.

La prueba de pertenencia al conjunto (IN)

Prueba si un valor de datos coincide con uno de una lista de valores objetivo. A continuación, se muestran varias consultas que utilizan la prueba de pertenencia de conjuntos:
Enumere los vendedores que trabajan en Nueva York, Atlanta o Denver.

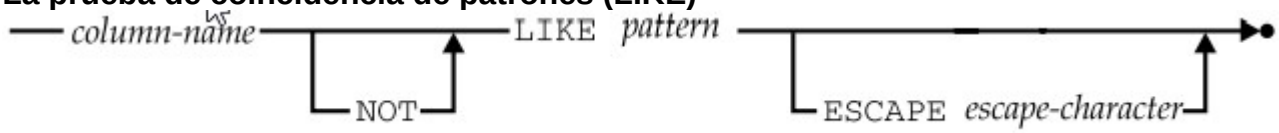
```
SELECT NAME, QUOTA, SALES
  FROM SALESREPS
```



Encuentre todos los pedidos realizados un viernes de enero de 2008.

```
SELECT ORDER_NUM, ORDER_DATE, AMOUNT
  FROM ORDERS
 WHERE ORDER_DATE IN ('2008-01-04', '2008-01-11',
                      '2008-01-18', '2008-01-25');
```

la condición de búsqueda
X EN (A, B, C)
es completamente equivalente a
(X = A) O (X = B) O (X = C)
La prueba de coincidencia de patrones (LIKE)



Puede utilizar una prueba de comparación simple para recuperar filas donde el contenido de una columna de texto coincide con algún texto en particular. **Personajes comodín** El carácter comodín del **signo de porcentaje (%)** coincide con cualquier secuencia de cero

o más caracteres. Por ejemplo, esta consulta recupera una fila de la tabla CUSTOMERS por nombre:

Muestre el límite de crédito para Smithson Corp.

```
SELECT COMPANY, CREDIT_LIMIT
  FROM CUSTOMERS
 WHERE COMPANY LIKE 'Smith% Corp.';
```

El carácter **comodín de subrayado (_)** coincide con cualquier carácter individual. Si está seguro de que el nombre de la empresa es "Smithson" o "Smithsen", por ejemplo, puede utilizar esta consulta:

```
SELECT COMPANY, CREDIT_LIMIT
  FROM CUSTOMERS
 WHERE COMPANY LIKE 'Smiths_n Corp.';
```

Puede localizar cadenas que no coincidan con un patrón utilizando la forma NOT LIKE de la prueba de coincidencia de patrones. La prueba LIKE debe aplicarse a una columna con un tipo de datos de cadena. Si el valor de los datos en la columna es NULL, la prueba LIKE devuelve un resultado NULL. Si ha utilizado computadoras a través de una interfaz de línea de comandos (como el shell de UNIX), probablemente haya visto coincidencias de patrones de cadenas antes. Con frecuencia, se usa el asterisco (*) en lugar del signo de porcentaje de SQL (%), y el signo de interrogación (?) Se usa en lugar del guión bajo de SQL (_), pero las capacidades de coincidencia de patrones en sí mismas son similares en la mayoría de las situaciones donde un La aplicación informática ofrece la capacidad de hacer coincidir partes seleccionadas de una palabra o texto.

Caracteres de escape *

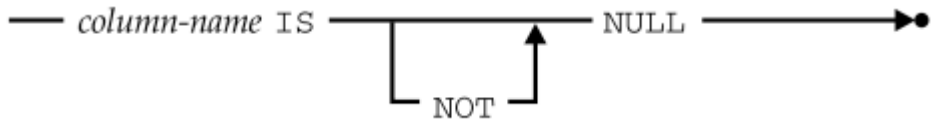
Uno de los problemas con la coincidencia de patrones de cadena es cómo hacer coincidir los propios caracteres comodín como caracteres literales. Para probar la presencia de un carácter de signo de porcentaje en una columna de datos de texto, por ejemplo, no puede simplemente incluir el signo de porcentaje en el patrón
porque SQL lo tratará como un comodín. Con algunos productos SQL más simples, no puede hacer coincidir literalmente los dos caracteres comodín.

A continuación se muestra un ejemplo que utiliza un signo de dólar ($) como carácter de escape:

Busque productos cuyos ID de producto comiencen con las cuatro letras "A% BC".

```
SELECT ORDER_NUM, PRODUCT
  FROM ORDERS
 WHERE PRODUCT LIKE 'A$%BC%' ESCAPE '$';
```

La prueba de valor nulo (ES NULO)

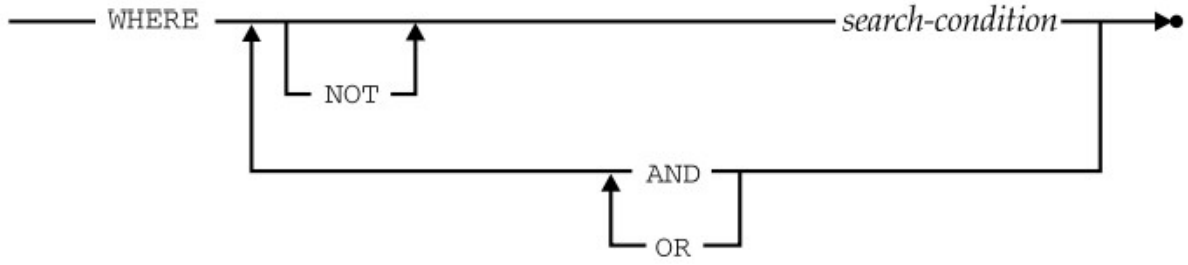


Los valores NULL crean una lógica de tres valores para las condiciones de búsqueda de SQL. Para cualquier fila dada, el resultado de una condición de búsqueda puede ser VERDADERO o FALSO, o puede ser NULO porque una de las columnas utilizadas para evaluar la condición de búsqueda contiene un valor NULO. A veces es útil verificar explícitamente valores NULL en una condición de búsqueda y manejarlos directamente.

Encuentra al vendedor aún no asignado a una oficina.

```
SELECT NAME
  FROM SALESREPS
 WHERE REP_OFFICE IS NULL;
```

Condiciones de búsqueda compuestas (AND, OR, and NOT)



La palabra clave OR se utiliza para combinar dos condiciones de búsqueda cuando una o la otra (o ambas) deben ser verdaderas:

Encuentre vendedores que estén por debajo de la cuota o con ventas por debajo de $ 300,000.

```
SELECT NAME, QUOTA, SALES
  FROM SALESREPS
 WHERE SALES < QUOTA
    OR SALES < 300000.00;
```

Encuentre vendedores que estén por debajo de la cuota y con ventas por debajo de $ 300,000.

```
SELECT NAME, QUOTA, SALES
  FROM SALESREPS
 WHERE SALES < QUOTA
   AND SALES < 300000.00;
```

Encuentre todos los vendedores que están por debajo de la cuota, pero cuyas ventas no son inferiores a $ 150,000.

```
SELECT NAME, QUOTA, SALES
  FROM SALESREPS
 WHERE SALES < QUOTA
   AND NOT SALES < 150000.00;
```

Encuentre a todos los vendedores que (a) trabajen en Denver, Nueva York o Chicago; o (b) no tienen gerente y fueron contratados desde junio de 2006; o (c) están por encima de la cuota, pero tienen ventas de $ 600,000 o menos.

```
SELECT NAME
  FROM SALESREPS
 WHERE (REP_OFFICE IN (22, 11, 12))
    OR (MANAGER IS NULL AND HIRE_DATE >= '2006-06-01')
    OR (SALES > QUOTA AND NOT SALES > 600000.00);
```

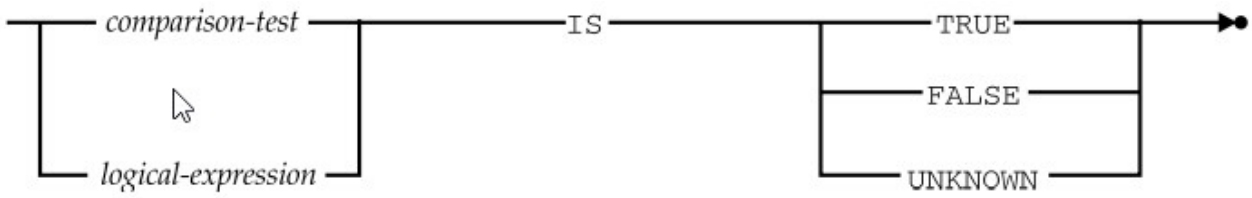
AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

TABLE 6-1 The AND Truth Table

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

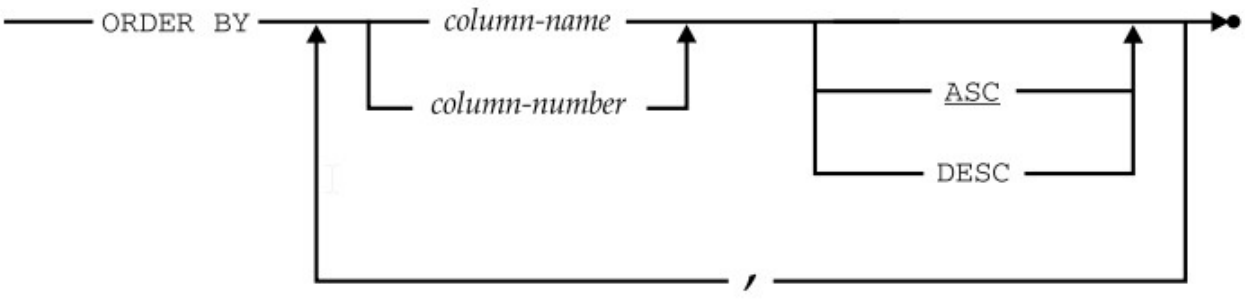
TABLE 6-2 The OR Truth Table

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL



Se puede utilizar para buscar filas en las que no se puede realizar la comparación porque SALES o QUOTA tienen un valor NULL. Del mismo modo, la prueba IS
((VENTAS - CUOTA)> 10000.00) IS FALSE

Clasificación de resultados de consultas (cláusula ORDER BY)



Al igual que las filas de una tabla en la base de datos, las filas de los resultados de la consulta no están organizadas en ningún orden en particular. Puede pedirle a SQL que ordene los resultados de una consulta incluyendo la cláusula ORDER BY en la instrucción SELECT que consta de las palabras clave ORDER BY, seguida de una lista de especificaciones de clasificación separadas por comas. Por ejemplo, los resultados de esta consulta están ordenados en dos columnas, REGIÓN y CIUDAD:
muestra las ventas de cada oficina, ordenadas alfabéticamente por región y dentro de cada región por ciudad.

```
SELECT CITY, REGION, SALES
  FROM OFFICES
 ORDER BY REGION, CITY;
```

Enumere las oficinas, clasificadas en orden descendente por ventas, de modo que las oficinas con las mayores ventas aparezcan primero.

```
SELECT CITY, REGION, SALES
  FROM OFFICES
 ORDER BY SALES DESC;
```

Enumere las oficinas, clasificadas en orden descendente por desempeño de ventas, para que aparezcan primero las oficinas con el mejor desempeño.

```
SELECT CITY, REGION, (SALES - TARGET)
  FROM OFFICES
 ORDER BY 3 DESC;
-or-
SELECT CITY, REGION, (SALES - TARGET)
  FROM OFFICES
 ORDER BY (SALES - TARGET) DESC;
```

Los resultados de la consulta se ordenan en la tercera columna, que es la diferencia calculada entre VENTAS y OBJETIVO para cada oficina. Al combinar números de columna, nombres de columna, ordenaciones ascendentes y descendentes, puede especificar una ordenación bastante compleja de los resultados de la consulta, como en el siguiente ejemplo final:
Enumere las oficinas, ordenadas alfabéticamente por región, y dentro de cada región en orden descendente por desempeño de ventas.

```
SELECT CITY, REGION, (SALES - TARGET)
  FROM OFFICES
 ORDER BY REGION ASC, 3 DESC;

CITY         REGION    (SALES-TARGET)
------------ -------- ---------------
New York     Eastern      $117,637.00
Atlanta      Eastern       $17,911.00
Chicago      Eastern      -$64,958.00
Los Angeles  Western      $110,915.00
Denver       Western     -$113,958.00
```

Reglas para el procesamiento de consultas de una sola tabla
Para generar los resultados de la consulta para una instrucción SELECT de una sola tabla, siga estos pasos:
1. Comience con la tabla nombrada en la cláusula FROM.
2. Si hay una cláusula WHERE, aplique su condición de búsqueda a cada fila de la tabla, conservando aquellas filas para las que la condición de búsqueda es TRUE y descartando aquellas filas para las que es FALSE o NULL.
3. Para cada fila restante, calcule el valor de cada elemento en la lista de selección para producir una sola fila de resultados de la consulta. Para cada referencia de columna, use el valor de la columna en la fila actual.
4. Si se especifica SELECT DISTINCT, elimine las filas duplicadas de los resultados de la consulta que se produjeron.
5. Si hay una cláusula ORDER BY, ordene los resultados de la consulta como se especifica.

Combinar resultados de consultas (UNION) *
Enumere todos los productos en los que el precio del producto excede los $ 2,000 o donde se ha pedido más de $ 30,000 del producto en un solo pedido.

```
SELECT MFR_ID, PRODUCT_ID
  FROM PRODUCTS
 WHERE PRICE > 2000.00
 UNION
SELECT DISTINCT MFR, PRODUCT
  FROM ORDERS
 WHERE AMOUNT > 30000.00;
```

Existen severas restricciones en las tablas que pueden combinarse mediante una operación UNION:
• Las dos cláusulas SELECT deben contener el mismo número de columnas.
• El tipo de datos de cada columna seleccionada de la primera tabla debe ser el mismo que el tipo de datos de la columna correspondiente seleccionada de la segunda tabla.
• Ninguna de las dos tablas se puede ordenar con la cláusula ORDER BY. Sin embargo, los resultados de la consulta combinados se pueden ordenar, como se describe en la siguiente sección.}

Uniones y filas duplicadas *
Debido a que la operación **UNION** combina las filas de dos conjuntos de resultados de consulta, tenderá a producir resultados de consulta que contengan filas duplicadas.
Enumere todos los productos en los que el precio del producto excede los $ 2,000 o donde se ha pedido más de $ 30,000 del producto en un solo pedido.

```
SELECT MFR_ID, PRODUCT_ID
  FROM PRODUCTS
 WHERE PRICE > 2000.00
 UNION ALL
SELECT DISTINCT MFR, PRODUCT
  FROM ORDERS
 WHERE AMOUNT > 30000.00;
```

La razón de la inconsistencia es que los valores predeterminados de SQL se eligieron para producir el comportamiento correcto la mayor parte del tiempo:
• En la práctica, la mayoría de las sentencias SELECT simples no producen filas duplicadas, por lo que el valor predeterminado es la eliminación de duplicados.
• En la práctica, la mayoría de las operaciones de UNION producirían filas duplicadas no deseadas, por lo que el valor predeterminado es la eliminación de duplicados.

Uniones y ordenamiento o clasificación *
La cláusula ORDER BY no puede aparecer en ninguna de las dos sentencias SELECT combinadas por una operación UNION. De todos modos, no tendría mucho sentido ordenar los dos conjuntos de resultados de la consulta, ya que se introducen directamente en la operación UNION y nunca son visibles para el usuario. Sin embargo, el conjunto combinado de resultados de la consulta producidos por la operación UNION se puede ordenar especificando una cláusula ORDER BY después de la segunda instrucción SELECT.
Enumere todos los productos en los que el precio del producto supera los $ 2,000 o donde se han pedido más de $ 30,000 del producto en un solo pedido, ordenados por fabricante y número de producto.

```
SELECT MFR_ID, PRODUCT_ID
  FROM PRODUCTS
 WHERE PRICE > 2000.00
 UNION
SELECT DISTINCT MFR, PRODUCT
  FROM ORDERS
 WHERE AMOUNT > 30000.00
 ORDER BY 1, 2;
```

Múltiples UNIONs*

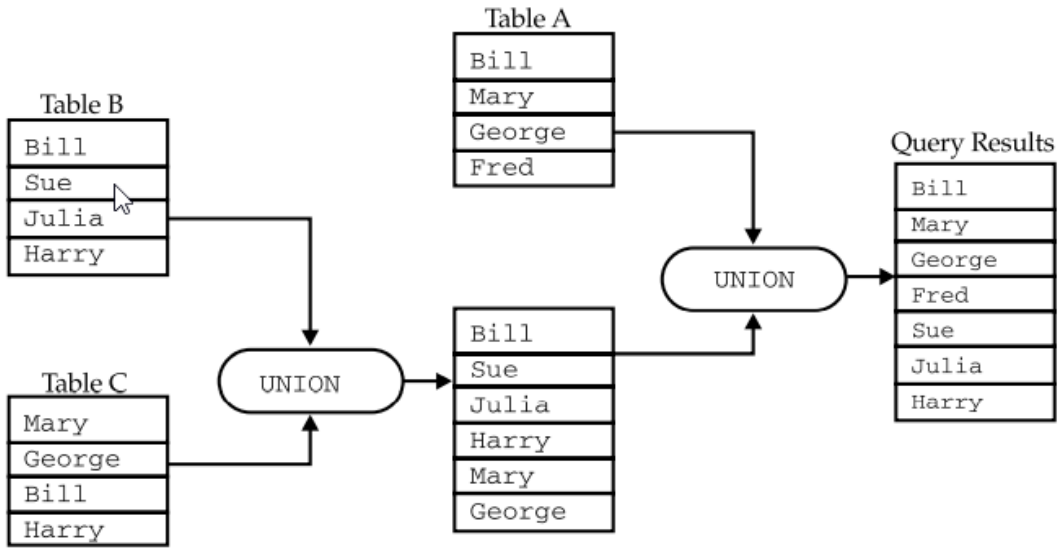
```
SELECT *
  FROM A
 UNION (SELECT *
          FROM B
         UNION
        SELECT *
          FROM C);
```

```
Bill
Mary
George
Fred
Sue
Julia
Harry
```

```
A UNION (B UNION C)

(A UNION B) UNION C

(A UNION C) UNION B
```



Resumen

Sobre consultas SQL. Describió las siguientes funciones de consulta:

• La instrucción SELECT se utiliza para expresar una consulta SQL. Cada instrucción SELECT produce una tabla de resultados de consulta que contiene una o más columnas y cero o más filas.

• La cláusula FROM especifica la (s) tabla (s) que contienen los datos que se recuperarán mediante una consulta.

• La cláusula SELECT especifica la (s) columna (s) de datos que se incluirán en los resultados de la consulta, que pueden ser columnas de datos de la base de datos o columnas calculadas.

• La cláusula WHERE selecciona las filas que se incluirán en los resultados de la consulta aplicando una condición de búsqueda a las filas de la base de datos.

• Una condición de búsqueda puede seleccionar filas comparando valores, comprobando un valor con un rango o conjunto de valores, haciendo coincidir un patrón de cadena y comprobando valores NULL.

• Las condiciones de búsqueda simples se pueden combinar con AND, OR y NOT para formar condiciones de búsqueda más complejas.

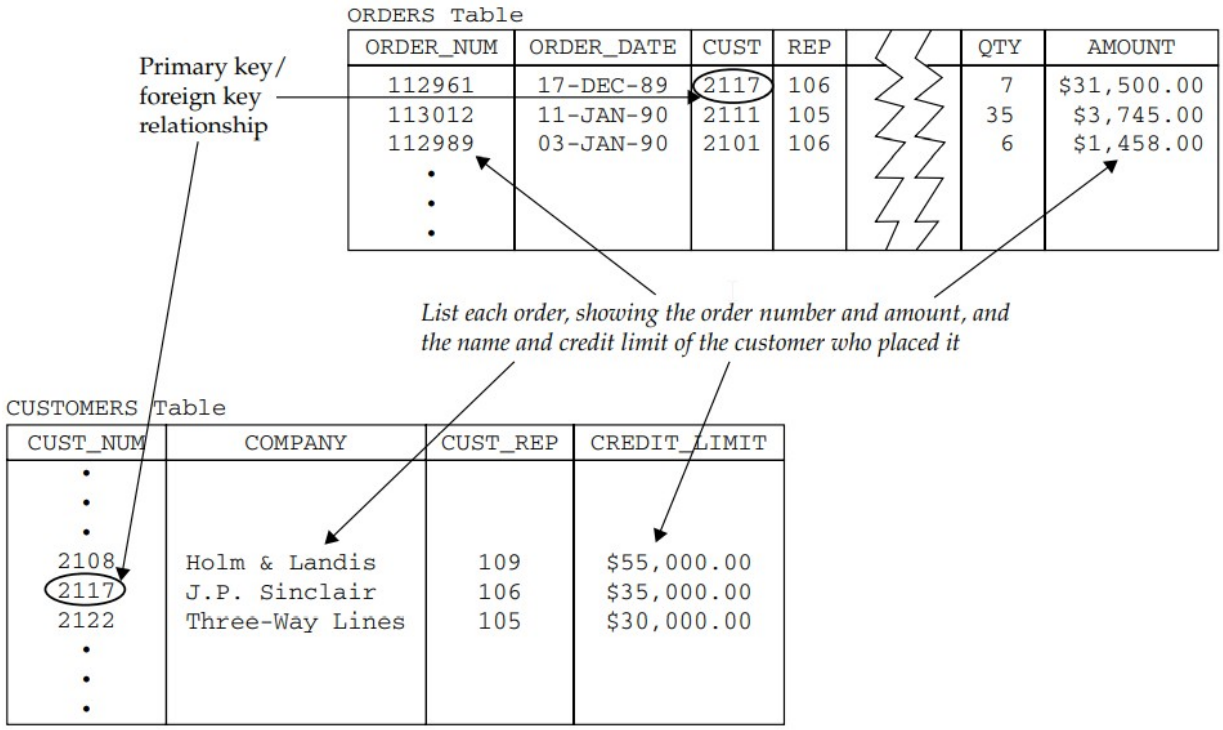
• La cláusula ORDER BY especifica que los resultados de la consulta deben ordenarse en orden ascendente o descendente, según los valores de una o más columnas.

• La operación UNION se puede utilizar dentro de una instrucción SELECT para combinar dos o más conjuntos de resultados de consultas en un solo conjunto.

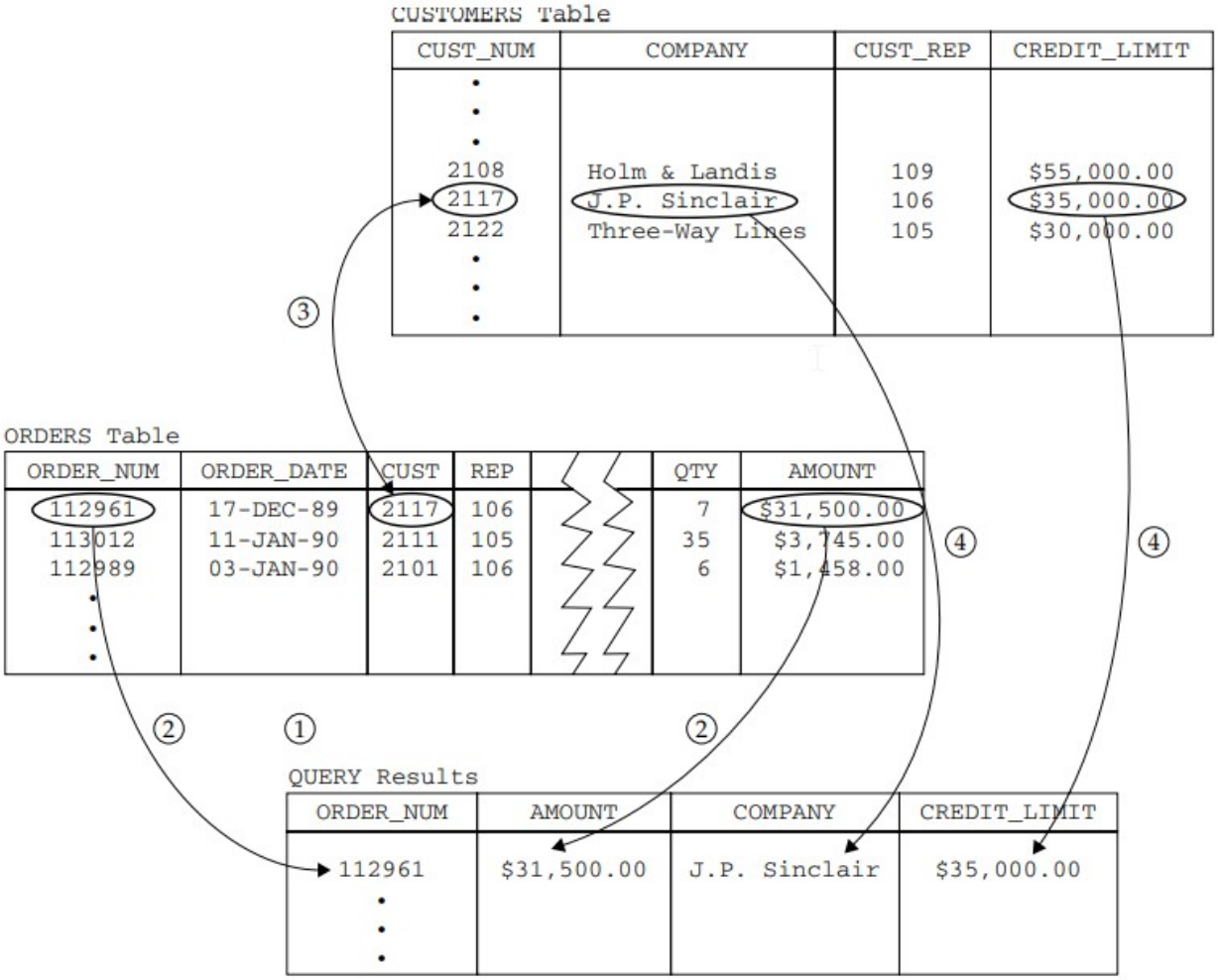
Consultas de varias tablas (JOINS)

SQL le permite recuperar datos que responden a estas solicitudes a través de consultas de múltiples tablas que unen datos de dos o más tablas.

Un ejemplo de consulta de dos tablas
"Enumere todos los pedidos, mostrando el número y el monto del pedido, y el nombre y el límite de crédito del cliente que lo realizó".
• La tabla PEDIDOS contiene el número de pedido y el monto de cada pedido, pero no tiene nombres de clientes ni límites de crédito.
• La tabla CLIENTES contiene los nombres de los clientes y los límites de crédito, pero carece de información sobre los pedidos.



1. Empiece por escribir los nombres de las cuatro columnas para los resultados de la consulta. Luego muévase a la tabla PEDIDOS y comience con el primer pedido.
2. Mire en la fila para encontrar el número de pedido (112961) y el monto del pedido ($ 31,500.00) y copie ambos valores en la primera fila de resultados de la consulta.
3. Mire en la fila para encontrar el número del cliente que hizo el pedido (2117) y vaya a la tabla CLIENTES para encontrar el número de cliente 2117 buscando en la columna CUST_NUM.
4. Desplácese por la fila de la tabla CLIENTES para encontrar el nombre del cliente ("J.P. Sinclair") y el límite de crédito ($ 35,000.00) y cópielos en la tabla de resultados de la consulta.
5. ¡Ha generado una fila de resultados de consultas! Vuelve a la tabla PEDIDOS y pasa a la siguiente fila. Repite el proceso, comenzando con el Paso 2, hasta que te quedes sin pedidos.



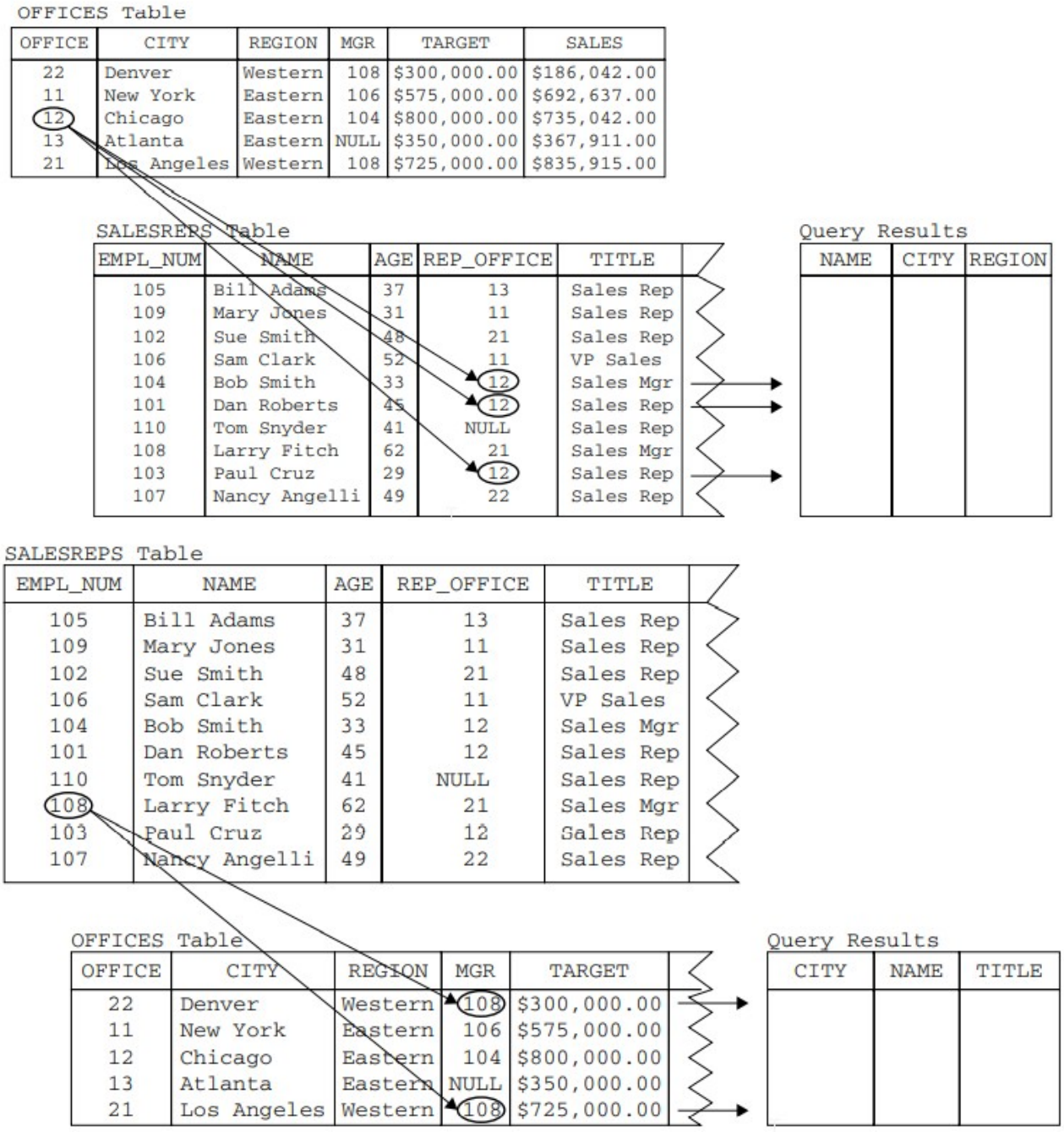
Enumere todos los pedidos que muestren el número de pedido, el monto, el nombre del cliente ("empresa") y el límite de crédito del cliente.

```
SELECT ORDER_NUM, AMOUNT, COMPANY, CREDIT_LIMIT
  FROM ORDERS, CUSTOMERS
 WHERE CUST = CUST_NUM;
```

Consultas principales / secundarias

Las consultas de múltiples tablas más comunes involucran dos tablas que tienen una relación principal / secundaria natural.

```
SELECT NAME, CITY, REGION
  FROM SALESREPS, OFFICES
 WHERE REP_OFFICE = OFFICE;
```



Enumere las oficinas y los nombres y cargos de sus gerentes.

```
SELECT CITY, NAME, TITLE
  FROM OFFICES, SALESREPS
 WHERE MGR = EMPL_NUM;
```

Una forma alternativa de especificar JOINS

La forma más sencilla de especificar las tablas que se unirán en una consulta de varias tablas es nombrarlas en una lista separada por comas, en la cláusula FROM de la instrucción SELECT, como se muestra en los ejemplos anteriores. Este método para especificar tablas unidas apareció en los primeros IBM Implementaciones SQL. Se incluyó en el estándar SQL original y es compatible con todas las bases de datos basadas en SQL. Las versiones posteriores del estándar expandieron significativamente la capacidad de unión y agregaron nuevas opciones a la cláusula

FROM. Con el formulario más reciente, los dos ejemplos de consulta anteriores también se podrían escribir así:

Enumere a cada vendedor y la ciudad y región donde trabaja.

```
SELECT NAME, CITY, REGION
  FROM SALESREPS JOIN OFFICES
    ON REP_OFFICE = OFFICE;
```

Enumere las oficinas y los nombres y cargos de sus gerentes.

```
SELECT CITY, NAME, TITLE
  FROM OFFICES JOIN SALESREPS
    ON MGR = EMPL_NUM;
```

En lugar de una lista de nombres de tablas separados por comas, la cláusula FROM de estos ejemplos usa la palabra clave JOIN para describir específicamente la operación de combinación. Además, las columnas coincidentes que se utilizarán en la combinación se especifican en la cláusula ON, que se encuentra al final de la cláusula FROM. Para estos ejemplos simples, la sintaxis SQL estándar expandida agrega muy poco a la forma anterior de la instrucción SELECT. Pero el rango de combinaciones que se pueden expresar usando la forma expandida es mucho más amplio, como se describe en secciones posteriores de este capítulo.

JOIN con criterios de selección de filas

La condición de búsqueda que especifica las columnas coincidentes en una consulta de varias tablas se puede combinar con otras condiciones de búsqueda para restringir aún más el contenido de los resultados de la consulta. Suponga que desea volver a ejecutar la consulta anterior, mostrando solo las oficinas con grandes objetivos de ventas:

Enumere las oficinas con un objetivo de más de $ 600 000 y la información de su gerente.

```
SELECT CITY, NAME, TITLE
  FROM OFFICES, SALESREPS
 WHERE MGR = EMPL_NUM
   AND TARGET > 600000.00;
```

Con la condición de búsqueda adicional, las filas que aparecen en los resultados de la consulta están aún más restringidas. La primera prueba (MGR = EMPL_NUM) selecciona solo pares de filas de OFICINAS y REPUESTOS DE VENTAS que tienen la relación padre / hijo adecuada; la segunda prueba además selecciona solo aquellos pares de filas donde el objetivo de la oficina es superior a $ 600 000. En esta forma de consulta, la condición de coincidencia para la combinación y la condición de búsqueda que restringe qué oficinas se seleccionan aparecen en la cláusula WHERE. Usando la sintaxis SQL estándar más nueva,

La condición de coincidencia aparece en la cláusula ON y la condición de búsqueda aparece en la cláusula WHERE, lo que hace que la consulta sea un poco más fácil de entender:

```
SELECT CITY, NAME, TITLE
  FROM OFFICES JOIN SALESREPS
    ON MGR = EMPL_NUM
 WHERE TARGET > 600000.00;
```

Varias columnas coincidentes

La tabla ORDERS y la tabla PRODUCTS en la base de datos de muestra están relacionadas por un par compuesto de clave externa / clave principal.

Enumere todos los pedidos, mostrando cantidades y descripciones de productos.

```
SELECT ORDER_NUM, AMOUNT, DESCRIPTION
  FROM ORDERS, PRODUCTS
 WHERE MFR = MFR_ID
   AND PRODUCT = PRODUCT_ID;
```