SQL: Manipulación de datos

SQL es un lenguaje de manipulación de bases de datos que ha sido implementado por prácticamente todos los sistemas de gestión de bases de datos relacionales (DBMS) pensados para múltiples usuarios, en parte porque ha sido aceptado por ANSI (Instituto Nacional Estadounidense de Estándares) y la ISO (Organización Internacional de Estándares) como lenguaje de consulta estándar para bases de datos relacionales.

Introducción

SQL fue desarrollado por IBM en su Laboratorio de Investigación de San José a principios de los años 70. Presentado en una conferencia de la ACM en 1974, el lenguaje se llamó originalmente SEQUEL (Structured English Query Language) y se pronunciaba "sequel". El nombre del lenguaje se acortó más tarde a SQL. Aunque IBM fue el autor de SQL, la primera implementación de SQL fue proporcionada por Oracle Corporation (en aquel entonces llamada Relational Software Inc.). Las primeras implementaciones comerciales se concentraron en DBMS basados en UNIX de tamaño mediano, como Oracle, Ingres e Informix. IBM siguió en 1981 con SQL/DS, el precursor de DB2, que debutó en 1983. El Instituto Nacional Estadounidense de Estándares (American National Standards Institute ANSI) publicó el primer estándar SQL (SQL-86) en 1986. Una versión internacional del estándar emitida por ISO apareció en 1987. Una importante actualización de SQL-86 se publicó en 1989 (SQL-89). Prácticamente todos los DBMS relacionales que se encuentran hoy en día son compatibles con la mayoría del estándar de 1989.

En 1992, el estándar fue revisado nuevamente (SQL-92), agregando más capacidades al lenguaje. Debido a que SQL-92 era un superconjunto de SQL-89, los programas de aplicación de base de datos más antiguos se ejecutaban bajo el nuevo estándar con modificaciones mínimas. De hecho, hasta octubre de 1996, los proveedores de DBMS podían enviar sus productos al NIST (Instituto Nacional de Estándares y Tecnología) para verificar el cumplimiento del estándar SQL. Este proceso de prueba y certificación proporcionó una motivación significativa para que los proveedores de DBMS se adhirieran al estándar SQL. Aunque descontinuar las pruebas de cumplimiento del estándar ahorra dinero a los proveedores, también hace que sea más fácil que los productos se desvíen del estándar. El estándar SQL-92 fue reemplazado por SQL:1999, que una vez más fue un superconjunto del estándar anterior. Las nuevas características principales de SQL:1999 admitían el modelo de datos relacional de objetos. El estándar SQL:1999 también añade una extensión a SQL para permitir que los métodos/funciones/procedimientos se escriban en SQL o en otro lenguaje de programación como C++ o Java y luego se invoquen desde dentro de otra sentencia SQL.

Incluso el estándar SQL:1999 completo no convierte a SQL en un lenguaje de programación completo e independiente. En particular, SQL carece de sentencias de E/S. Esto tiene mucho sentido, ya que SQL debería ser independiente de la implementación y del sistema operativo. Sin embargo, el estándar SQL:1999 completo incluye operaciones como la selección y la iteración que lo hacen computacionalmente completo. Estas características del lenguaje, que son más típicas de los lenguajes de programación de propósito general, se utilizan al escribir procedimientos almacenados y activadores. El estándar SQL se ha actualizado tres veces desde la aparición de SQL:1999 en versiones denominadas SQL:2003, SQL:2006 y SQL:2008. Además de perfeccionar las capacidades de las características relacionales básicas y ampliar el soporte relacional de objetos, estas revisiones han añadido compatibilidad con XML .

Entornos Interactivos

Un entorno interactivo de los DBMS son una interfaz o conjunto de herramientas que le permite a los usuarios interactuar directamente con la base de datos para realizar operaciones como consultas, actualizaciones, administración y análisis de datos, sin necesidad de programar extensivamente.

Existen dos formas generales en las que se puede emitir un comando SQL a una base de datos:

SQL interactivo, en el que un usuario escribe un solo comando y lo envía inmediatamente a la base de datos. El resultado de una consulta interactiva es una tabla en la memoria principal (una tabla virtual). En entornos de mainframe, cada usuario tiene una tabla de resultados a la vez, que se reemplaza cada vez que se ejecuta una nueva consulta; los entornos de PC a veces permiten varias.

SQL embebido, en el que las sentencias SQL se colocan en un programa de aplicación. La interfaz presentada al usuario puede estar basada en formularios o en líneas de comandos. El SQL embebido puede ser estático, en cuyo caso el comando completo se especifica en el momento en que se escribe el programa. Alternativamente, puede ser dinámico, en cuyo caso el programa crea la sentencia utilizando la entrada del usuario y luego la envía a la base de datos. Las sintaxis básicas del SQL interactivo y el SQL embebido estático son muy similares.

Para nuestras prácticas utilizaremos alguna interfaz de SQL interactivo como por ejemplo DataGrip o PgAdmin para comunicarnos con la base de datos.

Existen muchas opciones para crear un comando SQL pero todas están formadas por los mismos elementos:

Palabras clave: cada comando SQL comienza con una palabra clave (como SELECT, INSERT o UPDATE) que le indica al procesador de comandos el tipo de operación que se realizará. El resto de las palabras clave preceden a las tablas de las que se tomarán los datos, indican operaciones específicas que se realizarán sobre los datos, etc.

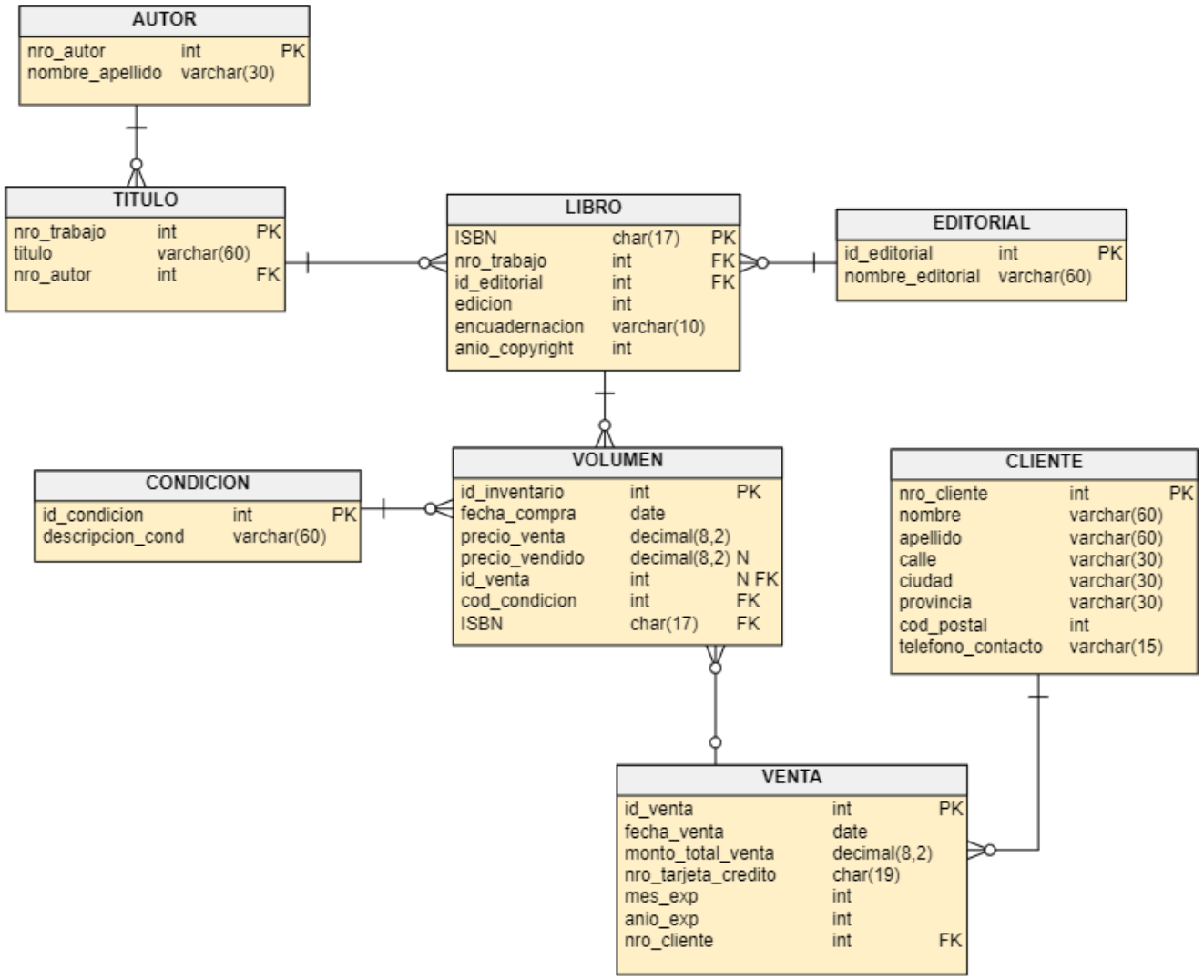
Tablas: un comando SQL incluye los nombres de las tablas sobre las que operará el comando.

Columnas: un comando SQL incluye los nombres de las columnas a las que afectará el comando.

Funciones: una función es un pequeño programa integrado en el lenguaje SQL. Por ejemplo, la función AVG calcula el promedio de valores de datos numéricos.

Se requieren palabras clave y tablas para todos los comandos SQL. Las columnas pueden ser opcionales, según el tipo de operación que se realice. Las funciones nunca son necesarias para una declaración SQL legal, pero en algunos casos pueden ser esenciales para obtener el resultado deseado.

A lo largo de nuestra práctica trabajaremos con algunas base de datos de ejemplo, una de ellas se refiere a una librería , el siguiente esquema de tablas nos representa la información de la misma:



Los datos para crear dicho esquema se pueden descargar del siguiente link, lo que se debe hacer es seleccionar todas las sentencias y pegarlas en la interfaz seleccionada para las prácticas y ejecutarlas. Estas sentencias crearán todas las tablas del esquema.

De la misma forma se debe descargar desde el siguiente link los sapos para cargar las tablas creadas anteriormente

Consultas Simples

SQL tiene un comando para recuperar datos: SELECT, posee sintaxis para elegir columnas, elegir filas, combinar tablas, agrupar datos y realizar algunos cálculos simples. De hecho, una sola instrucción SELECT puede hacer que un DBMS realice cualquiera o todas las operaciones de álgebra relacional. La sintaxis básica de la instrucción SELECT tiene la siguiente estructura general:

```
SELECT columna1, columna2 …
FROM tabla1, tabla2 …
WHERE predicado
```

La cláusula SELECT especifica las columnas que desean recuperar, la cláusula FROM, especifica las tablas utilizadas en la consulta y la cláusula WHERE puede contener una gran variedad de criterios que identifican los registros que se desean recuperar.

La mayoría de los procesadores de comandos SQL no distinguen entre mayúsculas y minúsculas cuando se trata de partes de una sentencia SQL. Las palabras clave SQL, los nombres de tablas, los nombres de columnas, etc. pueden estar tanto en mayúscula como en minúscula. Sin embargo, la mayoría de los DBMS distinguen entre mayúsculas y minúsculas cuando se trata de hacer coincidir valores de datos. Por lo tanto, siempre que coloque un valor entre comillas para que SQL coincida, debe hacer coincidir las mayúsculas y minúsculas de los datos almacenados. Es conveniente por una cuestión de mejor lectura de las sentencias SQL escribir las palabras claves en mayúsculas y los componentes de la base de datos, como los nombres de columnas y de tablas en minúsculas.

Selección de algunas columnas

Una de las características de una tabla es que se puede listar cualquiera de las columnas en el orden que se elija. Por lo tanto, SQL le permite especificar las columnas que desea listar y el orden en el que desea verlas. Para recuperar todas las columnas de una tabla, viendo las columnas en el orden en que se definieron cuando se creó la tabla, puede utilizar un asterisco (*). Por ejemplo, para ver todos los libros que maneja la librería:

```
SELECT *
FROM libro;
```

Algunos de sus datos

isbn	nro_trabajo	id_editorial	edicion	encuadernacion	anio_copyright
978-1-11111-111-1	1	1	2	Pegado	1857
978-1-11111-112-1	1	1	1	Pegado	1847
978-1-11111-113-1	2	4	1	Pegado	1842
978-1-11111-114-1	3	4	1	Pegado	1801
978-1-11111-115-1	3	4	10	Cuero	1925
978-1-11111-116-1	4	3	1	Pegado	1805
978-1-11111-117-1	5	5	1	Pegado	1808
978-1-11111-118-1	5	2	19	Cuero	1956
978-1-11111-120-1	8	4	5	Pegado	1906

En la mayoría de las consultas SQL, se querrá especificar exactamente qué columna o columnas se desea recuperar. Para especificar columnas, se deben incluir después de SELECT en el orden en el que desean listar. Por ejemplo:

```
SELECT isbn,
       encuadernacion,
       anio_copyright,
       edicion
FROM libro;
```

isbn	encuadernacion	anio_copyright	edicion
978-1-11111-111-1	Pegado	1857	2
978-1-11111-112-1	Pegado	1847	1
978-1-11111-113-1	Pegado	1842	1

Eliminando duplicados

En general todas las tablas poseen una clave primaria, es un conjunto de una o más columnas en una tabla de una base de datos relacional que se utiliza para identificar de manera única cada registro dentro de esa tabla. Es un concepto fundamental en el diseño y modelado de bases de datos. Por lo que las claves primarias garantizan que las tablas no tengan filas duplicadas. Sin embargo, cuando sólo se ve una parte de las columnas de una tabla, pueden darse registros con duplicados. Por ejemplo, al ejecutar la siguiente consulta:

```
SELECT encuadernacion
FROM libro;
```

encuadernacion
Pegado
Pegado
Pegado
Pegado
Cuero
Pegado
Pegado
Cuero
Pegado
Pegado
Cuero

Los duplicados aparecen porque hay varios libros qu poseen el mismo método de encuadernación. Para quitar los duplicados del resultado es necesario colocar la palabra clave DISTINCT delante de la lista de campos a seleccionar

```
SELECT DISTINCT encuadernacion
FROM libro;
```

encuadernacion
Pegado
Cuero

Ordenando el resultado

El orden en el que aparecen las filas en la tabla de resultados puede no ser el esperado. En algunos casos, las filas aparecerán en el orden en el que se almacenan físicamente. Sin embargo. Si se desea que el orden de las filas sea coherente y predecible, se deberá especificar cómo se desea que aparezcan. Para controlar el orden de las filas en una tabla de resultados, se debe agregar una cláusula ORDER BY a su declaración SELECT al final de toda la sentecia. Por ejemplo si se desea conocer un listado de autores ordenados por apellido y nombre tendríamos la siguiente consulta:

```
SELECT nro_autor, nombre_apellido
FROM autor
ORDER BY nombre_apellido;
```

Las palabras clave ORDER BY van seguidas de la columna o columnas en las que desea ordenar la tabla de resultados. Cuando incluye más de una columna, la primera columna representa la ordenación externa y la siguiente, una ordenación dentro de ella. Por ejemplo, supongamos que tenemos la siguiente consulta:

```
SELECT cod_postal,  apellido, nombre
FROM cliente
ORDER BY cod_postal,  apellido;
```

cod_postal	apellido	nombre
11111	Brown	Helen
11111	Doe	Jane
11111	Hayes	Edna
11111	Hayes	Franklin
11111	Jones	Janice
11111	Smith	Janice
13333	Smith	Jane
18886	Collins	Mary
18886	Collins	Peter
18886	Jerry	Helen
18888	Doe	John
18888	Johnson	Peter
18888	Johnson	Peter
18888	Jones	Jon
18888	Smith	John

El resultado primero ordena por código postal y luego ordena por apellido del cliente dentro de cada código postal.

Seleccionando registros

Además de listar algunas columnas de una tabla, también es posible listar sólo algunos registros. Es posible especificar los criterios de selección de registros en la cláusula WHERE de una sentencia SELECT. En su forma más simple, una cláusula WHERE contiene una expresión lógica con la que se evalúa cada registro de una tabla. Si un registro cumple con los criterios de la expresión, se convierte en parte de la tabla de resultados. Si el registro no cumple con los criterios, se omite. El truco para escribir criterios de selección de registros es, por lo tanto, saber cómo crear expresiones lógicas con las que se puedan evaluar los datos.

Predicados

Una expresión lógica que sigue a WHERE se conoce como predicado. Utiliza una variedad de operadores para representar los criterios de selección de registros. Si un registro cumple con los criterios de un predicado (en otras palabras, los criterios se evalúan como verdaderos), el registro se incluye en la tabla de resultados y si no cumple con los criterios (los criterios se evalúan como falsos), el registro se excluye.

- **Operadores de Comparación**

Existen seis operadores que se utilizan para expresar relaciones de datos (=, <, >, <=, >=, !=). En las consultas de bases de datos, las expresiones tienen un nombre de columna en un lado y un valor literal en el otro, por ejemplo:

precio_venta > 1.95

o nombres de columnas en ambos lados precio_venta > precio_vendido

La primera expresión pregunta "¿El precio de venta del libro es mayor que 1.95?". La segunda pregunta "¿El precio de venta del libro es mayor que el precio al cual se vendió (hubo descuento!!)?"

La forma en que ingresa valores literales en una expresión lógica depende del tipo de datos de la columna con la que está comparando el valor:

Números: Se deben escribir los números sin formato. En otras palabras, omita los signos de dólar, comas, etc. Sin embargo, debe colocar los puntos decimales en el lugar apropiado en un número con una parte fraccionaria.

Caracteres: Escriba caracteres entre comillas simples.

Fechas: Escriba las fechas en el formato utilizado para almacenarlas en la base de datos. Esto variará de un DBMS a otro, por lo tanto **siempre hay que utilizar las funciones de transformación de caracteres a fechas**.

Cuando utilice dos nombres de columna, tenga en cuenta que el predicado se aplica a cada fila de la tabla individualmente. El DBMS sustituye los valores almacenados en las columnas de la misma fila al realizar su evaluación de los criterios. Por lo tanto, puede utilizar dos nombres de columna cuando necesite examinar datos que están almacenados en el mismo registro pero en diferentes columnas.

El DBMS también basa la forma en que evalúa los datos de acuerdo a el tipo de dato, las comparaciones que involucran datos numéricos se basan en el orden numérico; las comparaciones que involucran datos de caracteres se basan en el orden alfabético; las comparaciones que involucran fechas y horas se basan en el orden cronológico.

- **Operadores Lógicos**

A veces, una expresión lógica simple no es suficiente para identificar los registros que se desean recuperar, es necesario más de un criterio. En ese caso, puede encadenar criterios con operadores lógicos. Por ejemplo, suponga que desea recuperar volúmenes que tiene en existencias que cuestan más de $75 y que están en excelentes condiciones. El predicado que necesita se compone de dos expresiones simples:

cod_condicion = 2 **precio_venta > 75**

Un registro debe cumplir con ambos criterios para ser incluído en la tabla de resultados. Por lo tanto, conecta las dos expresiones con el operador lógico AND en una sola expresión compleja:

cod_condicion = 2 AND precio_venta > 75

Puede usar los operadores AND u OR para combinar predicados, debajo se muestran las tablas de verdad para cada uno de ellos.

AND	True	False
True	True	False
False	False	False

OR	True	False
True	True	True
False	True	False

- **Negación**

El operador lógico NOT (o !) invierte el resultado de una expresión lógica. Si un registro cumple los criterios de un predicado, al colocar NOT delante de los criterios se excluye la fila del resultado. Del mismo modo, si un registro no cumple los criterios de un predicado, al colocar NOT delante de la expresión se incluye en el resultado.

Por ejemplo, NOT (precio_venta <= 50) recupera todos los registros en los que el precio de venta no es inferior o igual a 50 $ (en otras palabras, superior a 50 $). En primer lugar, el DBMS evalúa el valor de la columna precio_venta frente a la expresión precio_venta <= 50. Si la fila cumple los criterios, el DBMS no hace nada. Si

la fila no cumple los criterios, incluye la fila en el resultado. Los paréntesis del ejemplo anterior agrupan la expresión a la que se aplicará NOT.

NOT puede ser un poco complicado cuando se aplica a expresiones complejas. Como ejemplo, considere esta expresión:

NOT (precio_venta <= 50 AND precio_vendido < precio_venta)

Los registros que tienen un precio de venta menor o igual a $50 y un precio de venta menor que el precio al que se vendió el volumen de un libro cumplirán con los criterios dentro de los paréntesis. Sin embargo, el operador NOT las excluye del resultado. Aquellas filas que tienen un precio de venta mayor que $50 o un precio de venta mayor o igual que el precio vendido no cumplirán con los criterios dentro de los paréntesis, pero se incluirán en el resultado por NOT. Eso significa que la expresión es en realidad la misma que

precio_venta > 50 OR precio_vendido >= precio_venta

o

NOT (precio_venta <= 50) OR NOT (precio_vendido < precio_venta)

- **Precedencia de paréntesis**

Cuando se crea una expresión con más de una operación lógica, el DBMS debe decidir el orden en el que procesará las expresiones simples. A menos que le indique lo contrario, un DBMS utiliza un conjunto de reglas de precedencia predeterminadas. En general, un DBMS evalúa primero las expresiones simples, seguidas de la expresión lógica. Cuando hay más de un operador del mismo tipo, la evaluación se realiza de izquierda a derecha.

Por ejemplo consideremos la expresión sobre campos de la tabla VOLUMEN:

precio_venta < 50 OR cod_condicion = 2 AND precio_vendido > precio_venta

Si el precio de venta de un libro es $25, su código de condición es 3 y el precio al que fue vendido es $20, el DBMS excluirá la fila.

precio_venta < 50 OR cod_condicion = 2 AND precio_vendido > precio_venta

25 < 50	OR	3	!= 2	AND	20	>	25	
TRUE	OR	FALSE		AND	FALSE			
								⇒ FALSE
	TRUE			AND	FALSE			

La primera expresión simple es verdadera; la segunda es falsa. Un OR entre las dos primeras produce un resultado verdadero porque al menos uno de los criterios es verdadero. En el DBMS se realiza una operación AND entre el resultado verdadero de la primera parte y el resultado de la tercera expresión simple (falso). Debido a que estamos combinando un resultado verdadero y un resultado falso con AND, el resultado general es falso. Por lo tanto, la fila se excluye del resultado.

```
SELECT id_inventario,
       isbn,
       precio_venta,
       cod_condicion,
       precio_vendido
FROM volumen
WHERE precio_venta < 50 OR
      cod_condicion = 2 AND
      precio_vendido > precio_venta
ORDER BY precio_venta;
```

Podemos cambiar el orden en el que el DBMS evalúa los operadores lógicos y, coincidentemente, el resultado de la expresión, utilizando paréntesis para agrupar las expresiones que deben tener mayor precedencia:

precio_venta < 50 OR (cod_condicion = 2 AND precio_vendido > precio_venta)

25 < 50	OR (	3	!= 2	AND	20	>	25)
TRUE	OR (	FALSE		AND	FALSE		)
							⇒ TRUE
TRUE	OR			FALSE			

El DBMS otorga la mayor precedencia a las partes de la expresión dentro del paréntesis. Usando los mismos datos, la expresión dentro del paréntesis es falsa (ambas expresiones simples son falsas). Sin embargo, la operación OR con la primera expresión simple produce verdadero, porque la primera expresión simple es verdadera. Por lo tanto, la fila se incluye en el resultado.

```sql
SELECT id_inventario,
       isbn,
       precio_venta,
       cod_condicion,
       precio_vendido
FROM volumen
WHERE precio_venta < 50 OR
      (cod_condicion = 2 AND
      precio_vendido > precio_venta)
ORDER BY precio_venta;
```

- **peradores Especiales: BETWEEN, LIKE, IN y IS NULL**

Los predicados SQL pueden incluir una serie de operadores especiales que facilitan la escritura de criterios lógicos. Estos incluyen BETWEEN, LIKE, IN y IS NULL.

El operador **BETWEEN** simplifica la escritura de predicados que buscan valores que se encuentran dentro de un intervalo. Por ejemplo si buscamos todas las ventas que se realizaron entre los meses de mayo (05) y agosto (08) y las ordenamos por dicho mes inclusive:

```sql
SELECT id_venta,
       fecha_venta,
       EXTRACT(month from fecha_venta) as mes,
       monto_total_venta
FROM venta
WHERE EXTRACT(month from fecha_venta) BETWEEN 5 AND 8
ORDER BY  mes
```

En la sentencia anterior también vemos la función EXTRACT que nos permite el manejo con partes de un dato de tipo date (fecha) (Ver el manual Postgresql para más información aquí); el resultado es el siguiente:

id_venta	fecha_venta	mes	monto_total_venta
1	2013-05-29	5	510.00
4	2013-06-30	6	110.00
5	2013-06-30	6	110.00
3	2013-06-15	6	58.00
2	2013-06-05	6	125.00
7	2013-07-05	7	80.00
9	2013-07-07	7	50.00
10	2013-07-10	7	125.00
11	2013-07-10	7	200.00
12	2013-07-10	7	200.00
13	2013-07-10	7	25.95
14	2013-07-10	7	80.00
15	2013-07-12	7	75.00
16	2013-07-25	7	130.00
17	2013-07-25	7	100.00
8	2013-07-07	7	90.00
6	2013-07-05	7	505.00
18	2013-08-22	8	100.00

El operador **LIKE** proporciona una medida de coincidencia de patrones de cadenas de caracteres al permitirle utilizar marcadores de posición (comodines) para uno o más caracteres. Aunque puede encontrar que los comodines son diferentes en otro DBMS particular, en la mayoría de los casos, % (símbolo de porcentaje) representa cero, uno o más caracteres y _ (símbolo guión bajo) representa cero o un carácter. La forma en que funciona el operador LIKE se resume en:

Expression	Meaning
`LIKE 'Sm%'`	Begins with *Sm*
`LIKE '%ith'`	Ends with *ith*
`LIKE '%ith%'`	Contains *ith*
`LIKE 'Sm_'`	Begins with *Sm* and is followed by at most one character
`LIKE '_ith'`	Ends with *ith* and is preceded by at most one character
`LIKE '_ith_'`	Contains *ith* and begins and ends with at most one additional character
`LIKE '%ith_'`	Contains *ith*, begins with any number of characters, and ends with at most one additional character
`LIKE '_ith%'`	Contains *ith*, begins with at most one additional character, and ends with any number of characters

Como se puede ver, es posible combinar los dos comodines para producir una variedad de expresiones que comienza con, que termina con o que contiene un determinado string.

El operador **IN** compara el valor de una columna con un conjunto de valores. IN devuelve verdadero si el valor está dentro del conjunto. Por ejemplo, supongamos que un empleado de una tienda está comprobando el precio de venta de un libro y quiere saber si es $25, $50 o $60, con lo que se conoce hasta el momento lo podríamos utilizar con el conjunto de valores que necesitamos recuperar y la expresión se escribiría de la siguiente manera:

```
SELECT id_inventario,
       isbn,
       fecha_compra,
       precio_venta
FROM volumen
WHERE precio_venta IN (20, 50, 60)
ORDER BY precio_venta;
```

id_inventario	isbn	fecha_compra	precio_venta
61	978-1-11111-136-1	2012-10-22	50.00
10	978-1-11111-136-1	2011-12-15	50.00
36	978-1-11111-130-1	2011-12-06	50.00
2	978-1-11111-131-1	2012-01-23	50.00
50	978-1-11111-127-1	2013-01-06	50.00
51	978-1-11111-141-1	2013-01-06	50.00
52	978-1-11111-141-1	2013-01-06	50.00
8	978-1-11111-137-1	2012-06-20	50.00
60	978-1-11111-128-1	2012-12-16	50.00
55	978-1-11111-133-1	2013-02-06	60.00

Dentro de los datos podemos tener campos para los cuales no existe un valor, su contenido es **NULL** que es un indicador específico de una base de datos. Aunque las columnas que contienen valores nulos aparecen vacías cuando se las visualiza, la base de datos en realidad almacena un valor que representa un valor nulo, de modo que se puede distinguir un valor desconocido de, por ejemplo, un valor de cadena que contiene un espacio en blanco. Sin embargo, como usuario, rara vez sabrá exactamente qué utiliza un DBMS internamente para los valores nulos. Esto significa que es necesario una forma especial de identificar valores nulos en un predicado para poder recuperar registros que contengan valores nulos. Aquí es donde entra en juego el operador **IS NULL**. Por ejemplo si deseamos ver todos los volúmenes de libros que aún no se han vendido podemos consultar cuales tienen el id_venta en nulo:

```
SELECT id_inventario,
       isbn,
       precio_venta,
       cod_condicion
FROM volumen
WHERE id_venta IS NULL;
```

id_inventario	isbn	precio_venta	cod_condicion
7	978-1-11111-137-1	80.00	2
8	978-1-11111-137-1	50.00	3
9	978-1-11111-136-1	75.00	1
10	978-1-11111-136-1	50.00	2
16	978-1-11111-121-1	110.00	2
17	978-1-11111-124-1	75.00	2
27	978-1-11111-141-1	24.95	1
28	978-1-11111-141-1	24.95	1
29	978-1-11111-141-1	24.95	1
30	978-1-11111-145-1	27.95	1
31	978-1-11111-145-1	27.95	1
32	978-1-11111-145-1	27.95	1
36	978-1-11111-130-1	50.00	3

- **Usar la Clave Primaria para recuperar un registro**

Un tipo común de consulta de recuperación SQL utiliza una expresión de clave primaria en su predicado para recuperar exactamente una fila. Por ejemplo, si alguien en la librería de libros quiere ver el nombre y el número de teléfono del cliente número 6, entonces la consulta se escribe:

```
SELECT *
FROM cliente
WHERE nro_cliente = 6;
```

nro_cliente	nombre	apellido	calle	ciudad	provincia	cod_postal	telefono_contacto
6	Janice	Smith	800 Center Road	Anytown	NY	11111	518-555-6666

Si una tabla tiene una clave primaria concatenada entonces una expresión de clave primaria debe incluir un predicado complejo en el que cada columna de la clave primaria aparezca en su propia expresión lógica simple.

Aunque las consultas con expresiones de clave primaria se escriben con la intención de recuperar solo una fila, lo más común es que las consultas SQL estén diseñadas para recuperar varias filas. Cuando se desea recuperar datos en función de un valor en una sola columna, se construye un predicado que incluye solo una expresión lógica simple.

- **Predicados Complejos**

Cuando se desea ver filas que cumplen dos o más condiciones simples, se utiliza un predicado complejo en el que las condiciones simples están conectadas por AND u OR. Por ejemplo, si alguien quisiera ver los libros del pedido número 6 que se vendieron por menos del precio solicitado, la consulta se escribiría:

```
SELECT isbn,
       precio_venta,
       precio_vendido
FROM volumen
WHERE id_venta = 6 AND
      precio_venta < precio_vendido;
```

- **Campos Nulos: Lógica Trivaluada**

Los predicados vistos hasta ahora omiten una cosa importante: la presencia de valores nulos. ¿Qué debería hacer un DBMS cuando encuentra una fila que contiene un valor nulo en lugar de un valor conocido? Se requiere que el DBMS actúe de manera consistente cuando encuentra valores nulos.

Si se considera la siguiente consulta como ejemplo los libros cuyo precio de:

```
SELECT id_inventario,
       precio_vendido
FROM volumen
WHERE precio_vendido < 50;
```

id_inventario	precio_vendido
4	25.95
5	22.95
11	25.00
12	15.00
13	18.00
18	30.00
23	45.00
24	35.00
40	25.95
41	40.00
42	40.00
53	40.00
54	40.00
56	40.00
57	40.00
59	35.00
58	25.00
60	45.00

Como se ve en el resultado de la consulta cada registro en la tabla de resultados tiene un valor de precio de venta, lo que significa que las filas de libros no vendidos (aquellos con nulo en la columna de precio de venta) se omiten. El DBMS no puede determinar cuál será el precio de venta de los artículos no vendidos: tal vez sea menor a $50 o tal vez sea mayor o igual a $50. La política de la mayoría de los DBMS es excluir filas con nulos del resultado. Para filas con nulo en la columna de precio de venta, la respuesta **“desconocido” (Maybe)** a "¿El precio de venta es menor a 50?" se vuelve falsa. Esto parece bastante sencillo, pero ¿qué sucede cuando tiene una expresión lógica compleja de la cual una parte devuelve “**desconocido**”?

La operación de AND, OR y NOT debe expandirse para tener en cuenta que pueden estar operando en un “**desconocido**”. La tabla **lógica de tres valores** para AND es:

AND	True	False	Maybe
True	True	False	Maybe
False	False	False	False
Maybe	Maybe	False	Maybe

Tenga en cuenta que algo importante: la única forma de obtener un resultado verdadero es que ambas expresiones simples vinculadas por AND sean verdaderas. Dado que la mayoría de los DBMS excluyen los registros donde el predicado evalúa "**desconocido**", la presencia de valores nulos en los datos no cambiará lo que ve un usuario final.

Lo mismo es cierto cuando observa la tabla de verdad de tres valores para OR

OR	True	False	Maybe
True	True	True	True
False	True	False	Maybe
Maybe	True	Maybe	Maybe

Mientras una expresión simple sea verdadera, no importa si la segunda devuelve verdadero, falso o "**desconocido**". El resultado siempre será verdadero.

Si niega una expresión que devuelve "**deconocido**", el operador NOT no tiene efecto. En otras palabras, NOT (**desconocido**) sigue siendo **desconocido**.

Para ver las filas que devuelven **desconocido**, debe agregar una expresión a la consulta que utilice el operador **IS NULL.** Por ejemplo, la forma más fácil de ver qué volúmenes no se han vendido es escribir lo siguiente:

```
SELECT id_inventario,
       isbn
       precio_vendido
FROM volumen
WHERE precio_vendido IS NULL;
```

Recuperando Datos de más de una Tabla

Para recuperar datos de más de una tabla será necesario comprender como son los ensambles entre ellas, además de conocer que impacto tienen dichos ensambles en el rendimiento de la base de datos.

Hay dos tipos de sintaxis que se pueden utilizar para solicitar el ensamble de dos tablas. La primera, que hemos llamado sintaxis de ensamble "tradicional", es la única forma de escribir un ensamble en los estándares SQL hasta SQL-89.

SQL-92 agregó una sintaxis de ensamble más flexible y más fácil de usar. La sintaxis de ensamble SQL tradicional se basa en una combinación de operaciones de producto cartesiano y restricción. Tiene la siguiente forma general:

```
SELECT columnas
FROM tabla1, tabla2
WHERE table1.primary_key = table2.foreign_key;
```

Como se ve se colocan las tablas desde donde se desea recuperar datos después del FROM y la condición de ensamble en el predicado de la cláusula WHERE.

Por ejemplo, suponga que alguien desea ver todos los pedidos realizados por un cliente cuyo número de teléfono es 518-555-1111. El número de teléfono es parte de la tabla de cliente y la información de compra está en la tabla venta. Las dos tablas están relacionadas por la presencia del número de cliente en ambas (clave primaria de la tabla de cliente y clave extranjera de la tabla venta). La consulta para satisfacer la solicitud de información requiere, por lo tanto, un ensamble de las dos tablas sobre el número de cliente:

```
SELECT nombre,
       apellido,
       id_venta,
       fecha_venta
FROM cliente c, venta v
WHERE c.nro_cliente = v.nro_cliente
  AND telefono_contacto = '518-555-1111';
```

nombre	apellido	id_venta	fecha_venta
Janice	Jones	3	2013-06-15
Janice	Jones	17	2013-07-25
Janice	Jones	2	2013-06-05
Janice	Jones	1	2013-05-29

Hay dos cosas importantes que se deben tener en cuenta en la consulta anterior:

- El ensamble se realiza entre una clave primaria de una tabla y una clave extranjera de otra. Los ensambles que no cumplen con este patrón con frecuencia no son válidos.
- Debido a que la columna nro_cliente aparece en más de una tabla en la consulta, debe estar calificada por el nombre o alias de la tabla a la que hace referencia la columna, para nuestro ejemplo:

```
FROM cliente c, venta v
```

Para agregar un calificador, se antepone el nombre o alias al nombre de columna separando los dos con un punto.

JOIN sobre todas las columnas con el mismo nombre

El estándar SQL-92 introdujo una sintaxis de ensamble alternativa que es más simple y más flexible que la sintaxis tradicional.

Si está realizando un ensamble equitativa natural, hay tres variaciones de la sintaxis que puede usar, dependiendo de si la columna o columnas sobre las que está ensamblando tienen el mismo nombre y si desea usar todas las columnas coincidentes en la unión.

Cuando las columnas de la clave primaria y de la clave extranjera que se desea ensamblar tienen el mismo nombre y se desea utilizar todas las columnas coincidentes en la condición de ensamble se puede utilizar la siguiente sintaxis general:

SELECT column(s) FROM table1 NATURAL JOIN table2 La consulta anterior podría escribirse como:

```
SELECT nombre,
       apellido,
       id_venta,
       fecha_venta
FROM cliente c NATURAl JOIN venta v
WHERE telefono_contacto = '518-555-1111';
```

JOIN sobre las columnas seleccionadas

Si no desea utilizar todas las columnas coincidentes en una condición de ensamble pero las columnas aún tienen el mismo nombre es posible especificar los nombres de las columnas sobre las que se realizará el ensamble agregando una cláusula USING:

```
SELECT columna(s)
FROM tabla1 JOIN tabla2 USING (columna)
```

```
SELECT nombre,
       apellido,
       id_venta,
       fecha_venta
FROM cliente c JOIN venta v USING (nro_cliente)
WHERE telefono_contacto = '518-555-1111';
```

JOIN sobre columnas con diferentes nombres

Cuando las columnas sobre las que se realiza el ensamble no tienen los mismos nombres, debe utilizar una condición de ensamble similar a la utilizada en la sintaxis de ensamble SQL tradicional:

```sql
SELECT nombre,
       apellido,
       id_venta,
       fecha_venta
FROM cliente c JOIN venta v
    ON (c.nro_cliente = v.nro_cliente)
WHERE telefono_contacto = '518-555-1111';
```

Join de más de 2 tablas

Tenga en cuenta que la operación de ensamble sólo puede juntar dos tablas a la vez. Si necesita ensamblar más de dos tablas, debe juntarlas por pares. Por lo tanto, un ensamble de tres tablas requiere dos JOIN, un ensamble de cuatro tablas requiere tres JOIN, y así sucesivamente. Este último tipo de ensamble da control sobre el orden en el que se realizan los JOIN.

```sql
SELECT nombre,
       apellido,
       o.id_venta,
       fecha_venta,
       isbn
FROM cliente c JOIN venta v
    ON (c.nro_cliente = v.nro_cliente)
JOIN volumen o
    ON (v.id_venta = o.id_venta)
WHERE telefono_contacto = '518-555-1111';
```

Como se ve en el ejemplo para listar la columna id_venta es necesario prefijarla con el nombre o alias de la tabla dado que está presente tanto en la tabla venta como volumen.

OUTER JOIN

Un ensamble externo es un JOIN que incluye filas en una tabla de resultados aunque no haya una coincidencia entre las filas de las dos tablas que se están ensamblando.

Siempre que el DBMS no puede hacer coincidir las filas, coloca valores nulos en las columnas para las que no existen datos.

Para realizar un ensamble externo, se debe indicar el tipo de JOIN en la cláusula FROM. Por ejemplo, para realizar un ensamble externo a izquierda entre las tablas de cliente y venta, se podría escribir:

```
SELECT nombre, apellido, id_venta, fecha_venta
FROM cliente c
    LEFT JOIN venta v
        USING (c.nro_cliente = v.nro_cliente);
```

nombre	apellido	id_venta	fecha_venta
Janice	Jones	3	2013-06-15
Jane	Doe	4	2013-06-30
Janice	Smith	5	2013-06-30
Franklin	Hayes	6	2013-07-05
Helen	Jerry	7	2013-07-05
Jane	Smith	8	2013-07-07
Helen	Jerry	9	2013-07-07
Edna	Hayes	10	2013-07-10
Mary	Collins	11	2013-07-10
Peter	Collins	12	2013-07-10
Jon	Jones	13	2013-07-10
Janice	Smith	14	2013-07-10
Edna	Hayes	15	2013-07-12
Jon	Jones	16	2013-07-25
Janice	Jones	17	2013-07-25
Jane	Smith	18	2013-08-22
Janice	Jones	2	2013-06-05
Janice	Jones	1	2013-05-29
Janice	Smith	19	2013-09-01
Jon	Jones	20	2013-09-01
John	Smith	*NULL*	*NULL*
Peter	Johnson	*NULL*	*NULL*
John	Doe	*NULL*	*NULL*
Peter	Johnson	*NULL*	*NULL*
Helen	Brown	*NULL*	*NULL*

Observe que las últimas 5 filas las columnas id_venta y fecha_venta tiene NULL, lo que significa que los clientes no han realizado ninguna compra.

La sintaxis de OUTER JOIN para los ensambles tiene las mismas opciones que la sintaxis de los INNER JOIN. También es posible realizar un ensamble externo a derecha LEFT JOIN.