Universidade da Coruña
Departamento de Computación

Ampliación de SQL

Luis A. González Ares
lgares@udc.es

Ampliación de SQL – Planteamiento

Objetivos

- Describir algunos de los elementos que más recientemente se han incorporado al lenguaje SQL.
- Analizar las características de dichos elementos.
- Conocer su implementación en un SGBD real.
- Establecer relaciones entre los nuevos conocimientos de SQL y los que ya se revisaron.

Contenidos

1. Elementos iniciales
2. Join
3. Otros elementos
4. Expresiones condicionales

Nota

El presente material es un resumen de lo impartido en las clases de la Facultad de Informática, que se entrega como documento de apoyo.

Ampliación de SQL – Elementos iniciales

Caracteres problemáticos

- ’ Cadenas de caracteres que contienen un apóstrofe. Aparición en INSERT, SELECT, etc. Solución: ’’ (su aparición dos veces lo convierte en el propio carácter, y no en lo que representa).

Predicado LIKE

Correspondencia con un patrón o modelo. Comodines: `_` `%`

```
expresión_carácter [NOT] LIKE patrón [ESCAPE carácter_protector]
```

Problemas si el patrón contiene comodines. Solución: indicar un carácter protector.

```
SELECT    *
FROM      emp
WHERE     ename LIKE '%a%%' ESCAPE 'a'
ORDER BY  nombre
```

Predicado SIMILAR

Correspondencia con un patrón (expresión regular).

Usa: `_` `%` `*` `+` `[]` `[^ ]` `[: :]` `|` `||` `()`

```
expresión_carácter [NOT] SIMILAR patrón
```

Ampliación de SQL – Elementos iniciales (ii)

Predicado EXISTS

Comprobación de si una subconsulta devuelve o no filas.

```
... [NOT]  EXISTS  subconsulta
```

```
SELECT    *
FROM      dept d
WHERE     EXISTS (SELECT *
                  FROM   emp
                  WHERE  sal > 1500
                  AND    deptno = d.deptno)
```

Predicado UNIQUE

Comprueba si en el resultado de una subconsulta aparecen filas repetidas.

```
... [NOT]  UNIQUE  subconsulta
```

```
SELECT    *
FROM      dept d
WHERE     UNIQUE (SELECT *
                  FROM   emp
                  WHERE  sal > 1500
                  AND    deptno = d.deptno)
```

Ampliación de SQL – Elementos iniciales (iii)

Operadores de cuantificación ANY o SOME, ALL

Comprobación de una expresión escalar con el resultado del operador sobre una subconsulta.

```
... expresión_escalar operador_comparación { ALL | SOME | ANY } (lista_valores)
```

Operadores de comparación: `=` `!=` `>` `<` `<=` `>=`

```
SELECT *
FROM   emp
WHERE  sal = ANY (SELECT sal
                  FROM   emp
                  WHERE  deptno = 30);
```

Operadores conjuntistas UNION, INTERSECT, EXCEPT

Aplica el operador al resultado de dos consultas.

```
consulta1
{ UNION | EXCEPT | INTERSECT }
{ ALL | DISTINCT }
consulta2
```

- `UNION`: Unión.
- `INTERSECT`: Intersección.
- `EXCEPT`: Diferencia.

Ampliación de SQL – Elementos iniciales (iv)

Operadores conjuntistas UNION, INTERSECT, EXCEPT (cont.)

Cada operador por defecto aplica `DISTINCT`.

- `UNION`: Valores que aparecen en las filas de una tabla o de la otra, sin repetir ningún valor.

- `UNION ALL`: Valores que aparecen en las filas de una tabla o de la otra, repetidos las veces que aparecen.

- `INTERSECT`: Valores que aparecen en las filas de una tabla y de la otra, sin repetir ningún valor.

- `INTERSECT ALL`: Valores que aparecen en las filas de una tabla y de la otra, repetidos n veces (si x e y son el número de veces que se repite un valor en la primera y en la segunda tabla, respectivamente, entonces $n = \text{mín}(x, y)$).

- `EXCEPT`: Valores que aparecen en las filas de la primera tabla y no en la segunda, sin repetir ningún valor.

- `EXCEPT ALL`: Valores que aparecen en las filas de la primera tabla y no en la segunda, repetidos n veces (si x e y son el número de veces que se repite un valor en la primera y en la segunda tabla, respectivamente, entonces $n = \text{máx}(x - y, 0)$).

Ampliación de SQL – Join

INNER JOIN

Permite vincular las columnas de varias tablas mediante operadores de comparación.

Sintaxis:

```
SELECT *
FROM   t1 [INNER] JOIN t2
  ON   condición_join
```

Dados:

```
emp(empno, ename, deptno)
```

```
dept(deptno, dname, loc)
```

el join se expresa mediante:

```
SELECT *
FROM   emp, dept
WHERE  emp.deptno = dept.deptno
AND    ename LIKE 'A%'
```

```
SELECT *
FROM   emp INNER JOIN dept
  ON   emp.deptno = dept.deptno
WHERE  ename LIKE 'A%'
```

Supongamos ahora:

```
emp(empno, ename, deptno, loc)
```

```
dept(deptno, dname, loc)
```

y deseamos (equijoin):

```
SELECT *
FROM   emp INNER JOIN dept
  ON   emp.deptno = dept.deptno
  AND  emp.loc = dept.loc
```

Ampliación de SQL – Join (ii)

Join natural

Si expresamos una condición de igualdad sobre todas las columnas que tienen el mismo nombre en dos tablas, se tiene el join natural.

Sintaxis:

```
SELECT *
FROM   t1 NATURAL [INNER] JOIN t2
```

Dados:

```
emp(empno, ename, deptno, loc)
```

```
dept(deptno, dname, loc)
```

el join natural sería:

```
SELECT *
FROM   emp NATURAL INNER JOIN dept
```

Una alternativa de efectos casi iguales:

```
SELECT *
FROM   emp INNER JOIN dept
  USING (deptno, loc)
```

El join normal:

```
SELECT *
FROM   emp INNER JOIN dept
  ON   emp.deptno = dept.deptno
  AND  emp.loc = dept.loc
```

Los resultados son muy parecidos: en el primer y segundo resultado, las columnas `deptno` y `loc` aparecen una única vez; no así en el tercero. El join natural y el join con `USING` son iguales del todo sólo si todas las columnas de nombres iguales están en el `USING`.

El join natural se corresponde con la operación join ($\bowtie$) de álgebra relacional.

Ampliación de SQL – Join (iii)

OUTER JOIN

Permite vincular las columnas de varias tablas mediante operadores de comparación, haciendo que aparezcan todas las columnas de una tabla, de la otra, o de ambas, cumplan o no la condición de join.

Sean dos relaciones `R` y `S`. Tenemos las operaciones:

- Left outer join de `R` y `S` ($R \ltimes S$): Todas las filas de la relación de la izquierda (`R`), rellenando con nulos en las filas que no se correspondan con las de `S`.
- Right outer join de `R` y `S` ($R \rtimes S$): Todas las filas de la relación de la derecha (`S`), rellenando con nulos en las filas que no se correspondan con las de `R`.
- Full outer join de `R` y `S` ($R \ltimes\!\rtimes S$): Todas las filas de la relación de la izquierda (`R`), y todas las filas de la relación de la derecha (`S`), rellenando con nulos en las filas que no se correspondan con las de la otra.

```
SELECT *
FROM   r {LEFT | RIGHT | FULL} [OUTER] JOIN s
  ON   condición_join
```

```
SELECT *
FROM   r LEFT OUTER JOIN s
  ON   r.c2 = s.c2
```

Ampliación de SQL – Join (iv)

CROSS JOIN

Representa el producto cartesiano de las tablas involucradas.

```
SELECT *
FROM   t1 CROSS JOIN t2
```

```
SELECT *
FROM   t1, t2
```

UNION JOIN

Dadas dos tablas, se construye una tabla que tiene cada columna y cada fila de ambas tablas.

```
SELECT *
FROM   t1 UNION JOIN t2
```

Dadas las tablas `R(a, b)` y `S(x, y, z)`, el resultado del `UNION JOIN` es:

```
 a    |   b   |  x    |  y    |  z
...     ...     NULL    NULL    NULL
...
...     ...     NULL    NULL    NULL
NULL    NULL    ...     ...     ...
...
NULL    NULL    ...     ...     ...
```

Ampliación de SQL – Join (v)

Múltiples joins

El orden de ejecución es de izquierda a derecha (asociativos a la izquierda).

Equivalencias:

```
SELECT expresion, ...
FROM  t1  JOIN t2
  ON condición_join1
  JOIN t3
  ON condición_join2
  ...
WHERE predicado
```

```
SELECT expresion, ...
FROM  (t1  JOIN t2
  ON condición_join1)
  JOIN t3
  ON condición_join2
  ...
WHERE predicado
```

Modificación del orden:

```
SELECT expresion, ...
FROM  t1  JOIN
  (t2  JOIN tb3
  ON condición_join2)
  ON condición_join1
  ...
WHERE predicado
```

Ampliación de SQL – Otros elementos

Subconsultas de fila

Se trata de subconsultas que devuelven más de una columna.

Permite realizar operaciones de comparación simultáneamente sobre cada columna que aparece en su resultado.

Sintaxis:

```
SELECT expresión_a, ...
FROM tabla_1, ...
WHERE (expresión_p1, ..., expresión_pn) operador (SELECT expresión_s1, ..., expresión_sn
                                                  FROM ...
                                                  WHERE ...)
```

Operadores válidos: comparación, `IN`, `= SOME`, `>ALL`, ...

Operadores válidos en Oracle 9.2: `IN`, `= SOME`, `= ALL` (para subconsultas de fila!)

Ejemplo:

```
SELECT *
FROM   articulo
WHERE  (cd_articulo, precio_min) IN
                                    (SELECT cd_articulo, precio
                                     FROM   ventas)
```

Ampliación de SQL – Otros elementos (ii)

Expresión de consulta

Se denomina expresión de consulta, vista en línea o tabla derivada a la utilización de una consulta en las cláusulas `SELECT` o `FROM` de otra.

Sintaxis:

```
SELECT expresión, ..., (SELECT ...)
FROM   t1, ...
 ...
```

```
SELECT expresión, ...
FROM   t1,  (SELECT ...)
...
```

- No debe confundirse con la denominación tradicional de subconsulta.
- Permite que en el resultado de una consulta aparezcan datos correspondientes a elementos diferentes.
- Si aparece en la cláusula `SELEC` sólo puede ser un `SELECT` escalar.
- Su potencialidad está en que aparezca en la cláusula `FROM`.
- Las expresiones de consulta dentro de un `FROM` deben ser autosuficientes.

Ejemplos:

```
SELECT empno, ename, sal, (SELECT MAX(sal) FROM emp) - sal
FROM   emp
```

```
SELECT empno, ename, sal, max_sal, min_sal, max_sal - sal, sal - min_sal
FROM   emp, (SELECT MAX(sal) max_sal, MIN(sal) min_sal
             FROM   emp)
WHERE  sal > max_sal / 2
```

Ampliación de SQL – Expresiones condicionales

Expresión CASE

En SQL pueden incluirse expresiones condicionales, o sea, que su valor dependa de que se cumplan unas condiciones, mediante la expresión `CASE`.

Sintaxis:

```
CASE
    WHEN condición_1 THEN expresión_1
    ...
    WHEN condición_n THEN expresión_n
    [ELSE expresión_m]
END
```

Si se produce la condición $\texttt{condición}_i$ el resultado será la expresión $\texttt{expresión}_i$, con $i = 1, \ldots, n$; en otro caso será $\texttt{expresión}_m$.

Todas las expresiones deben tener el mismo tipo de datos.

Puede aparecer donde lo pueda hacer una expresión.

```
SELECT empno, ename, sal,
       CASE
         WHEN sal > 1000 AND sal <= 1500 AND comm IS NULL
           THEN 'Salario medio'
         WHEN sal > 1500 AND sal <= 2500
           THEN 'Alto'
         ELSE  'Otros'
       END, comm
FROM   emp
```

Ampliación de SQL – Expresi. condicionales (ii)

CASE reducida

Consiste en una sintaxis alternativa, más reducida, permitida si todas las condiciones de un `CASE` son de igualdad sobre la misma expresión. Sería el caso :

```
CASE
  WHEN expresión_o = expresión_o1 THEN expresión_1
  ...
  WHEN expresión_o = expresión_on THEN expresión_n
  [ELSE expresión_m]
END
```

Podría expresarse como:

```
CASE expresión_o
  WHEN expresión_o1 THEN expresión_1
  ...
  WHEN expresión_on THEN expresión_n
  [ELSE expresión_m]
END
```

Ejemplo:

```
SELECT deptno, dname,
       CASE deptno
         WHEN 10 THEN 'CONTABILIDAD'
         WHEN 20 THEN 'INVESTIGACION'
         ELSE 'Otros'
       END
FROM   dept
```

Ampliación de SQL – Expresi. condicionales (iii)

NULLIF

Supongamos que se desea comparar la correspondencia de una expresión con otra, y si coinciden asignar un valor nulo como resultado, manteniendo el valor de la primera expresión si no se produce la correspondencia.

Esto es:

```
CASE
  WHEN expresión1 = expresión2 THEN NULL
  ELSE expresión1
END
```

Una forma alternativa de expresar lo anterior es usar la expresión `NULLIF`:

```
NULLIF( expresión1 , expresión2 )
```

Ejemplo:

```
SELECT empno, ename, sal, NULLIF(comm, -1)
FROM   emp
```

Ampliación de SQL – Expresi. condicionales (iv)

COALESCE

Dadas n expresiones, nos interesa como resultado la primera de las $n-1$ que es diferente al valor nulo; en caso de que ninguna lo sea, el resultado será la expresión n.

Esto es:

```
CASE
  WHEN expresión_1 IS NOT NULL THEN expresión_1
  WHEN expresión_2 IS NOT NULL THEN expresión_2
  ...
  WHEN expresión_{n-1} IS NOT NULL THEN expresión_{n-1}
  ELSE expresión_n
END
```

La forma alternativa de expresar lo anterior usando la expresión `COALESCE` es:

```
COALESCE(expresión_1, expresión_2, ..., expresión_{n-1}, expresión_n)
```

Ejemplo:

```
SELECT codigo, nombre, precio, precio_min, COALESCE(0.85*precio, precio_min, 12) Rebajas
FROM   articulo
```

Ampliación de SQL – Otras sentencias

TRUNCATE TABLE

La eliminación de todas las filas de una tabla se realiza con la sentencia `DELETE TABLE` usándola de la forma:

```
DELETE FROM mitabla
```

Recordemos que `DELETE` es una sentencia DML, que mantiene la estructura de la tabla intacta y que no modifica las características del espacio asignado a la tabla.

Una manera alternativa de eliminar todas las filas de una tabla es usar `TRUNCATE TABLE`

```
TRUNCATE TABLE mitabla
```

La sentencia elimina todas las filas de la tabla y libera los espacios asignados a la tabla, manteniendo la estructura de la tabla y de los índices creados.

`TRUNCATE TABLE` es en general más eficiente que `DELETE TABLE` (libera espacios, no recarga los aspectos transaccionales, etc.).

Se considera una sentencia DDL, aunque no modifica las estructuras lógicas.

Ampliación de SQL – Otras sentencias (ii)

MERGE

Permite realizar a la vez operaciones de `UPDATE` y de `INSERT` sobre una tabla destino, según que se cumpla o no una condición entre sus filas y las de otra tabla o consulta, denominada tabla origen (`<tabla origen> : {<tabla base> | <vista> | <consulta>}`) .

```
MERGE INTO <tabla destino>
USING      <tabla origen>
   ON      ( <condición> )
WHEN MATCHED THEN
           <sentencia UPDATE>
WHEN NOT MATCHED THEN
           <sentencia INSERT>
```

Las filas de la tabla origen se dividen en dos grupos:

- Las que cumplen la condición o filas UPDATE.
- Las que no cumplen la condición o filas INSERT.

Y ocurre que:

- Cada fila de la tabla destino que se corresponde, según la condición, con una de las filas UPDATE, se actualiza. Es un error que una fila de la tabla destino se corresponda con más de una de las filas UPDATE.
- Las filas INSERT se incorporan a la tabla destino.

Ampliación de SQL – Otras sentencias (iii)

MERGE (cont.)

Dadas las dos tablas `MD (C1, C2)` y `MO (C1, C2)` con las siguientes filas:

```
MD:  C1   C2              MO:  C1   C2
      a  100                    a  200
      b  150                    b  250
      c  175                    d  290
```

Ejecutando:

```
MERGE INTO md d
USING      mo o
   ON      ( d.c1 = o.c1 )
WHEN MATCHED THEN
           UPDATE  SET  c2 = o.c2
WHEN NOT MATCHED THEN
           INSERT  (c1, c2) VALUES (o.c1, o.c2)
```

Se obtiene:

```
MD:  C1   C2              MO:  C1   C2
      a  200                    a  200
      b  250                    b  250
      c  175                    d  290
      d  290
```

Ampliación de SQL – Otras sentencias (iv)

MERGE (cont.)

En resumen:

- En la sentencia `MERGE` solo se modifica la tabla destino.
- La tabla origen puede ser una tabla base, una vista o una consulta.
- Las filas de la tabla origen que se corresponden según la condición (filas UPDATE), se actualizan en la tabla destino.
- Las filas de la tabla origen que no se corresponden según la condición (filas INSERT), se incorporan a la tabla destino.
- Las filas de la tabla destino que no se corresponden según la condición, no se modifican.
- En el bloque `MATCHED` no se puede modificar una columna de la tabla destino involucrada en la condición.

La sentencia `MERGE` aparece en SQL:2003.

Tanto en las implementaciones de los fabricantes como en las revisiones del estándar (SQL:2008), se orienta a ampliar las posibilidades de los bloques `MATCHED` y `NOT MATCHED` con sentencias y condiciones adicionales.

Ampliación de SQL – Bibliografía

Referencias

[CeCa03] Celma, M.; Casamayor, J. C.; Mota, L.: **Bases de Datos Relacionales**. Prentice Hall, 2003.

[GrWe02] Groff, J.; Weinberg, P. N.: **SQL: The Complete Reference** (2^{nd} edition). McGraw-Hill, 2002. (Traducción: **SQL. Manual de referencia**. McGraw-Hill, 2003.)

[GuPe99] Gulutzan, P.; Pelzer, T.: **SQL-99 Complete, Really**. R&B Books, 1999.

[MeSi02] Melton, J.; Simon, A.: **SQL:1999 - Understanding Relational Language Components**. Morgan Kaufmann, 2002.

[RiMa02] Rivero, E; Martínez, L.; Reina, L.; Benavides, J.; Olaizola, J.: **Introducción al SQL para usuarios y programadores**. Thomson- Paraninfo, 2002.

[SiKo06] Silberschatz, A.; Korth, H.; Sudarshan, S. **Database System Concepts** (5^{th} edition). McGraw-Hill, 2006. (Traducción: **Fundamentos de Bases de Datos**. McGraw-Hill, 2006.)