El SQL menos conocido

Todos conocemos el lenguaje SQL como el estándar de consultas para bases de datos relacionales. Sin embargo, este estándar ha sufrido varias revisiones a lo largo de su historia, que han ido enriqueciendo el lenguaje, y que en muchos casos, son grandes desconocidas para los desarrolladores.

El objetivo de este post es dar un repaso a las funcionalidades menos conocidas de SQL.

Juan Antonio Jiménez Torres

20/03/2018

Índice

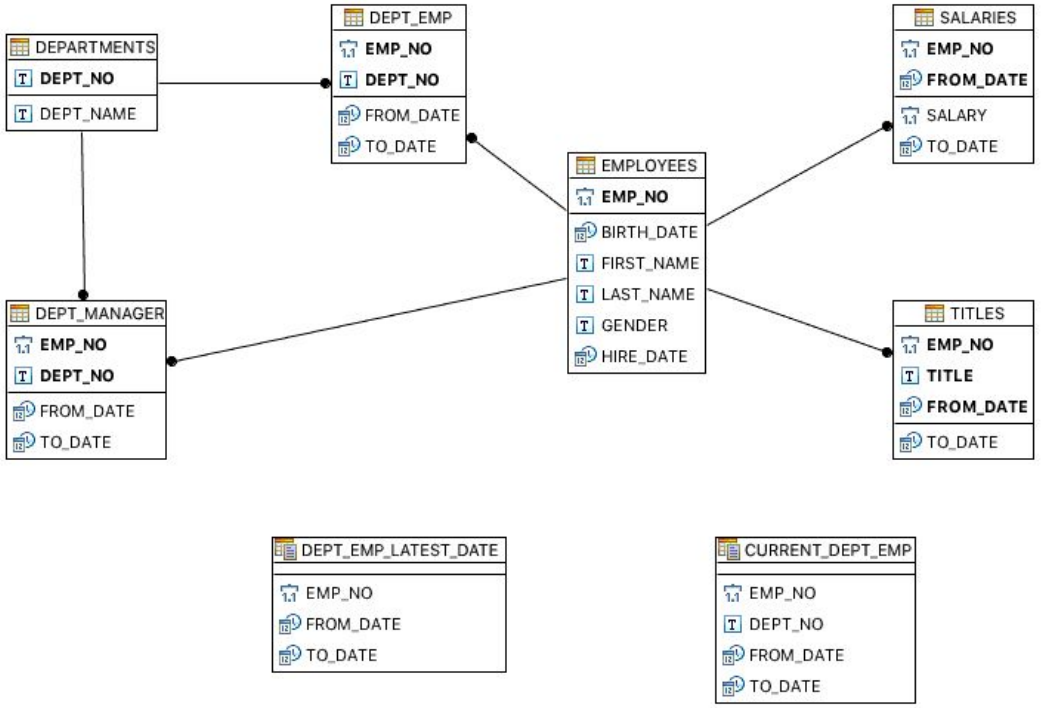
Todos conocemos el lenguaje SQL como el estándar de consultas para bases de datos relacionales. Sin embargo, este estándar ha sufrido varias revisiones a lo largo de su historia, que han ido enriqueciendo el lenguaje, y que en muchos casos, son grandes desconocidas para los desarrolladores.

El objetivo de este post es dar un repaso a las funcionalidades menos conocidas de SQL.

1. Antes de empezar

Para los ejemplos, se va a usar la BBDD de ejemplo de https://github.com/datacharmer/test_db

Que tiene el siguiente modelo:



El cliente de BBDD de este artículo es DBeaver v3.5.8

2. Evoluciones del estándar:

El lenguaje SQL ha sufrido muchas revisiones. La última en 2016.

- SQL-86
- SQL-89
- SQL-92
- SQL-1999
- SQL-2003

- SQL-2008
- SQL-2011
- SQL-2016

La mayoría de gestores de base de datos implementan el último estándar, aunque cada uno tiene sus matices. Sin embargo, el uso mayoritario que se sigue haciendo se corresponde con el de SQL-92.

3. CTE. Common Table Expressions. WITH

SQL-1999 introduce las tablas comunes y consultas recursivas a través de la construcción WITH.

A cierto modo, WITH es como si crease una vista al vuelo y luego se utilizase en una query. En la BBDD que estamos manejando de ejemplo en este artículo, los directores actuales de cada departamento se pueden obtener de la siguiente manera.

```
      SELECT EMP_NO, DEPT_NO
        FROM DEPT_MANAGER
       WHERE FROM_DATE < CURRENT_DATE
         AND TO_DATE > CURRENT_DATE
```

Si guardamos el resultado de esta query en una variable, podemos usarla a continuación y hacer cosas del estilo.

```
WITH managers AS (
      SELECT EMP_NO, DEPT_NO
        FROM DEPT_MANAGER
       WHERE FROM_DATE < CURRENT_DATE
         AND TO_DATE > CURRENT_DATE
)
SELECT
      M.EMP_NO,
      E.FIRST_NAME,
      E.LAST_NAME,
      E.GENDER,
      D.DEPT_NO,
      D.DEPT_NAME
  FROM managers M
  INNER JOIN EMPLOYEES E ON M.EMP_NO=E.EMP_NO
  INNER JOIN DEPARTMENTS D ON M.DEPT_NO=D.DEPT_NO;
```

En este caso, esa consulta se podría resolver de forma más óptima con un INNER JOIN, pero ejemplifica el uso que se puede hacer de la cláusula WITH.

4. CTE Recursivas. WITH... LEVEL

En ocasiones nos encontraremos con estructuras jerárquicas que requieren de consultas con cierta recursividad. Lo bueno de las CTE es que las podemos usar dentro de su propia definición, de forma que ahí tenemos la recursividad.

Por ejemplo, imaginemos que tenemos una tabla donde guardamos el linaje por línea materna.

```
CREATE TABLE person (
    id              INT             NOT NULL,
    name            VARCHAR(30)     NOT NULL,
    mother          INT             NOT NULL,
    PRIMARY KEY (id)
);
```

Definimos una primera mujer a la que llamaremos *Eva*, y a partir de ahí vamos guardando su descendencia haciendo referencia a quien es su madre.

```
insert into person (id, name, mother) values
(1, 'Eva', 0),
(2, 'Ana', 1),
(3, 'María', 1),
(4, 'Rocío', 2),
(5, 'Carmen', 2),
(6, 'Aurora', 3),
(7, 'Pilar', 4);
```

Y al hacer una consulta sobre esta tabla, deseamos obtener en una columna su linaje materno desde Eva hasta la madre de cada niña.

Pues podemos utilizar una CTE recursiva para hacerlo, indicando el nivel de profundidad (recursividad) que deseamos emplear, en este ejemplo 4.

```
WITH RCTE (id, name, mother, family, level) as
(
       select
              root.id,
              root.name,
              root.mother,
              CAST('' AS VARCHAR(50)) as family,
              1 as level
       from PERSON root
       where root.id=1
union all
       select
```

```
            child.id,
            child.name,
            child.mother,
            concat(concat(parent.family,' > '), parent.name),
            parent.level +1 as level
      from RCTE parent, PERSON child
      where parent.id = child.mother
        and parent.level < 4
)
select distinct id, name, mother, family from RCTE
```

Y este es el resultado de la consulta.

	ID	NAME	MOTHER	FAMILY
1	1	Eva	0	
2	2	Ana	1	> Eva
3	3	María	1	> Eva
4	5	Carmen	2	> Eva > Ana
5	4	Rocío	2	> Eva > Ana
6	6	Aurora	3	> Eva > María
7	7	Pilar	4	> Eva > Ana > Rocío

Este tipo de consultas, se suelen utilizar para obtener estructuras jerárquicas.

5. CUBE, ROLLUP y GROUPING SETS

También la especificación SQL-1999 amplía el significado de la cláusula GROUP BY con los operadores CUBE, ROLLUP y GROUPING SETS.

En el ejemplo que tenemos, vamos a obtener un listado de empleados, con su cargo actual y salario. La siguiente consulta nos podría dar este resultado:

```
select
      e.emp_no,
      first_name,
      last_name,
      gender,
      title,
      salary
from titles t
      inner join employees e on t.EMP_NO = e.EMP_NO
      inner join salaries s
            on s.EMP_NO = e.EMP_NO AND current_date > s.from_date AND
current_date < s.to_date
 where
```

```
   current_date > t.from_date
   AND current_date < t.to_date
   AND e.emp_no<10021;
```

	EMP_NO	FIRST_NAME	LAST_NAME	GENDER	TITLE	SALARY
1	10.001	Georgi	Facello	M	Senior Engineer	88.958
2	10.002	Bezalel	Simmel	F	Staff	72.527
3	10.003	Parto	Bamford	M	Senior Engineer	43.311
4	10.004	Chirstian	Koblick	M	Senior Engineer	74.057
5	10.005	Kyoichi	Maliniak	M	Senior Staff	94.692
6	10.006	Anneke	Preusig	F	Senior Engineer	59.755
7	10.007	Tzvetan	Zielinski	F	Senior Staff	88.070
8	10.009	Sumant	Peac	F	Senior Engineer	94.409
9	10.010	Duangkaew	Piveteau	F	Engineer	80.324
10	10.012	Patricio	Bridgland	M	Senior Engineer	54.423
11	10.013	Eberhardt	Terkki	M	Senior Staff	68.901
12	10.014	Berni	Genin	M	Engineer	60.598
13	10.016	Kazuhito	Cappelletti	M	Staff	77.935
14	10.017	Cristinel	Bouloucos	F	Senior Staff	99.651
15	10.018	Kazuhide	Peha	F	Senior Engineer	84.672
16	10.019	Lillian	Haddadi	M	Staff	50.032
17	10.020	Mayuko	Warwick	M	Engineer	47.017

E imaginemos que ahora queremos desglosar los sueldos medios por género y por cargo en la empresa. Podemos agrupar por género y cargo con cubo de la siguiente manera.

```
select
       gender,
       title,
       avg(salary) as salary_avg
from titles t
       inner join employees e on t.EMP_NO = e.EMP_NO
       inner join salaries s
              on s.EMP_NO = e.EMP_NO AND current_date > s.from_date AND
current_date < s.to_date
 where
   current_date > t.from_date
   AND current_date < t.to_date
   AND e.emp_no<10021
   group by gender, title with cube;
```

Fijémonos en los bloques de null.

Las cuatro primeras filas nos dan el sueldo medio por cargo, independientemente del sexo.

La fila cinco nos da el salario medio independientemente del género (null) y del cargo (null). Es decir, nos da el sueldo medio de este grupo.

	GENDER	TITLE	SALARY_AVG
1	[NULL]	Engineer	62.646
2	[NULL]	Senior Engineer	71.369
3	[NULL]	Senior Staff	87.828
4	[NULL]	Staff	66.831
5	[NULL]	[NULL]	72.901
6	F	[NULL]	82.772
7	M	[NULL]	65.992
8	F	Engineer	80.324
9	F	Senior Engineer	79.612
10	F	Senior Staff	93.860
11	F	Staff	72.527
12	M	Engineer	53.807
13	M	Senior Engineer	65.187
14	M	Senior Staff	81.796
15	M	Staff	63.983

La fila 6 y 7 nos dan el salario medio desglosado por sexos independientemente del cargo (null)

Y el resto de filas nos dan el salario medio desglosado por cargo y género.

La opción WITH ROLLUP nos da el mismo resultado, pero sin tener en cuenta las cuatro primeras filas.

```
select
      gender,
      title,
      avg(salary) as salary_avg
from titles t
      inner join employees e on t.EMP_NO = e.EMP_NO
      inner join salaries s
            on s.EMP_NO = e.EMP_NO AND current_date > s.from_date AND
current_date < s.to_date
 where
   current_date > t.from_date
   AND current_date < t.to_date
   AND e.emp_no<10021
   group by gender, title with rollup;
```

	GENDER	TITLE	SALARY_AVG
1	[NULL]	[NULL]	72.901
2	F	[NULL]	82.772
3	M	[NULL]	65.992
4	F	Engineer	80.324
5	F	Senior Engineer	79.612
6	F	Senior Staff	93.860
7	F	Staff	72.527
8	M	Engineer	53.807
9	M	Senior Engineer	65.187
10	M	Senior Staff	81.796
11	M	Staff	63.983

La primera fila nos da el salario medio de esos empleados.

La fila 2 y 3 el salario medio desglosado por género.

Y de la 5 a la 11, el salario medio, desglosado por género y cargo que ocupa.

En el caso de grouping sets, saca la información desglosada por uno y otro campo.

```
select
      gender,
      title,
      avg(salary) as salary_avg
from titles t
      inner join employees e on t.EMP_NO = e.EMP_NO
      inner join salaries s
            on s.EMP_NO = e.EMP_NO AND current_date > s.from_date AND
current_date < s.to_date
 where
   current_date > t.from_date
   AND current_date < t.to_date
   AND e.emp_no<10021
   group by grouping sets (gender, title);
```

	GENDER	TITLE	SALARY_AVG
1	[NULL]	Engineer	62.646
2	[NULL]	Senior Engineer	71.369
3	[NULL]	Senior Staff	87.828
4	[NULL]	Staff	66.831
5	F	[NULL]	82.772
6	M	[NULL]	65.992

6. Funciones de Ventana

Las funciones de ventana fueron introducidas en SQL2003 y posteriormente ampliadas en SQL2008 y nos proporcionan una forma de tratar datos calculados referidos a una ventana de nuestra query.

Por ejemplo, volviendo a la query de nuestro ejemplo que sacaba datos de los empleados, ahora queremos una columna adicional, que de esos datos nos devuelva el salario medio desglosado por género.

Esa columna sería

```
avg(salary) over (partition by gender) as avg_salary_by_gender
```

y la consulta quedaría así:

```
select
      e.emp_no,
      first_name,
```

```
      last_name,
      gender,
      title,
      salary,
      avg(salary) over (partition by gender) as avg_salary_by_gender
from titles t
      inner join employees e on t.EMP_NO = e.EMP_NO
      inner join salaries s
            on s.EMP_NO = e.EMP_NO AND current_date > s.from_date AND
current_date < s.to_date
 where
   current_date > t.from_date
   AND current_date < t.to_date
   AND e.emp_no<10021;
```

Y el resultado sería:

	EMP_NO	FIRST_NAME	LAST_NAME	GENDER	TITLE	SALARY	AVG_SALARY_BY_GENDER
1	10.002	Bezalel	Simmel	F	Staff	72.527	82.772
2	10.006	Anneke	Preusig	F	Senior Engineer	59.755	82.772
3	10.007	Tzvetan	Zielinski	F	Senior Staff	88.070	82.772
4	10.009	Sumant	Peac	F	Senior Engineer	94.409	82.772
5	10.010	Duangkaew	Piveteau	F	Engineer	80.324	82.772
6	10.017	Cristinel	Bouloucos	F	Senior Staff	99.651	82.772
7	10.018	Kazuhide	Peha	F	Senior Engineer	84.672	82.772
8	10.001	Georgi	Facello	M	Senior Engineer	88.958	65.992
9	10.003	Parto	Bamford	M	Senior Engineer	43.311	65.992
10	10.004	Chirstian	Koblick	M	Senior Engineer	74.057	65.992
11	10.005	Kyoichi	Maliniak	M	Senior Staff	94.692	65.992
12	10.012	Patricio	Bridgland	M	Senior Engineer	54.423	65.992
13	10.013	Eberhardt	Terkki	M	Senior Staff	68.901	65.992
14	10.014	Berni	Genin	M	Engineer	60.598	65.992
15	10.016	Kazuhito	Cappelletti	M	Staff	77.935	65.992
16	10.019	Lillian	Haddadi	M	Staff	50.032	65.992
17	10.020	Mayuko	Warwick	M	Engineer	47.017	65.992

Igual que se calcula la media, se puede calcular también el *max*, *min*, *sum*, etc... así como otras funciones numéricas.

Algunas estrategias de optimización suelen hacer este tipo de queries y por fuera otra query que se queda sólo con un subconjunto de estos datos. Por ejemplo, en este caso sería muy fácil obtener los empleados que cobran por encima de la media según su género.

También se utiliza con frecuencia en combinación con rownumber() para paginar resultados.

Por ejemplo, queremos obtener los empleados ordenador por apellido y nombre, la segunda página, con un tamaño de página 20

```
SELECT
      emp_no, first_name, last_name, gender, title, salary
```

```
FROM
(
      SELECT
            e.emp_no,
            first_name,
            LAST_name,
            gender,
            title,
            salary,
            rownumber() OVER (ORDER BY last_name, first_name) AS
rownumber
      FROM titles t
            INNER JOIN employees e ON t.EMP_NO = e.EMP_NO
            INNER JOIN salaries s
                  ON s.EMP_NO = e.EMP_NO
                        AND CURRENT_DATE > s.from_date
                        AND CURRENT_DATE < s.to_date
       WHERE
         CURRENT_DATE > t.from_date
         AND CURRENT_DATE < t.to_date
)
WHERE
      ROWNUMBER >= 21 AND ROWNUMBER <40
```

Y el resultado quedaría

	EMP_NO	FIRST_NAME	LAST_NAME	GENDER	TITLE	SALARY
1	252.763	Felicidad	Aamodt	F	Engineer	49.161
2	234.739	Foong	Aamodt	M	Senior Engineer	71.807
3	265.036	Fuqing	Aamodt	M	Senior Engineer	95.600
4	103.736	Garnet	Aamodt	F	Senior Engineer	67.467
5	84.904	Gladys	Aamodt	F	Senior Engineer	80.746
6	107.608	Gretta	Aamodt	M	Engineer	91.957
7	290.839	Guoxiang	Aamodt	M	Staff	84.328
8	240.075	Guther	Aamodt	M	Engineer	50.682
9	233.941	Hairong	Aamodt	M	Senior Staff	95.324
10	100.916	Heejo	Aamodt	M	Senior Engineer	78.028
11	109.542	Heeju	Aamodt	M	Senior Engineer	66.683
12	255.520	Hidefumi	Aamodt	M	Staff	62.885
13	292.102	Hidefumi	Aamodt	M	Senior Staff	116.825
14	49.007	Hiroyasu	Aamodt	M	Senior Staff	75.423
15	49.426	Huican	Aamodt	M	Senior Engineer	50.364
16	103.125	Huican	Aamodt	M	Staff	67.173
17	275.120	Jaideep	Aamodt	M	Senior Staff	93.594
18	109.883	Jeanna	Aamodt	M	Engineer	50.597
19	61.477	Jenwei	Aamodt	M	Engineer	67.798

7. Insertar o actualizar en una sentencia mediante MERGE

Una situación que nos encontramos con frecuencia es que si tenemos un registro en la BBDD debemos actualizarlo, y si no, debemos insertarlo.

La forma habitual es hacer el select del registro y si no nos devuelve resultados, lo insertamos, y si nos devuelve resultados, actualizamos sólo los campos afectados.

En SQL2003 se introdujo la sentencia MERGE que luego se ampliaría en SQL2008 y que nos permite esto mismo en una sentencia:

```
MERGE INTO tablename USING table_reference ON (condition)
   WHEN MATCHED THEN
   UPDATE SET column1 = value1, column2 = value2, ...
   WHEN NOT MATCHED THEN
   INSERT (column1, column2, ...) VALUES (value1, value2, ...);
```

8. Conclusiones

El SQL es más que las sentencias propias de un CRUD. Y cada gestor de BBDD implementa parte o la totalidad del estándar añadiendo sus particularidades. Pero antes que esas particularidades, conveniente conocer el estándar para sacar más partido al lenguaje.