SQL: Structured Query Language
Additional SQL Syntax

SQL WITH Clauses and Common Table Expressions

Consider the following example of a relational table `Attendance` documenting attendance of different events (e.g., meeting of a West Coast Swing Club, or performance of the jazz band) on Cal Poly campus:

```
CREATE TABLE Attendance (
  eventID INT PRIMARY KEY,
  eventName VARCHAR(32),
  eventDate DATE,
  eventType VARCHAR(10),
  attended  INT
);
```

Consider the quesion: Find the date with the largest total event attendance. One SQL query that answers this questions is

```
SELECT a.eventDate
FROM (SELECT eventDate, SUM(attended) as Attended
      FROM Attendance
      GROUP BY eventDate) a
WHERE Attended = (SELECT MAX(Attended)
                  FROM (SELECT eventDate, SUM(attended) as Attended
                        FROM Attendance
                        GROUP BY eventDate) b
                  )
;
```

This query features a repetition of an inner (nested) query:

```
SELECT eventDate, SUM(attended) as Attended
      FROM Attendance
      GROUP BY eventDate
```

It is found once in the FROM clause, and once inside the nested SELECT MAX(Attended) ... query[1].

Note, that you **must** include the verbose descriptions of the inner query **twice**. The following variant:

```
SELECT a.eventDate
FROM (SELECT eventDate, SUM(attended) as Attended
      FROM Attendance
      GROUP BY eventDate) a
WHERE Attended = (SELECT MAX(Attended)
                  FROM a
                  )
;
```

WILL RESULT IN AN ERROR MESSAGE!

At the same time, there are valid reasons to want to not have to repeat a complete SELECT statement in nested queries. Such statements are referred to as Common Table Expressions or CPEs.

SQL's WITH clause allows one to assign a Common Table Expression to a table alias and successuflly use the table alias in a single SQL query without having to repeat the Common Table Expression multiple times.

The most simple applications of the WITH syntax are "syntactic sugar", i.e., they do not alter the expressive power of the underlying SQL queries. **However**, the WITH clauses allow for recursion, and SQL queries that include recursive Common Table Expressions actually provide functionality (transitive closures) that is missing from pure SELECT-FROM-WHERE-GROUP BY-HAVING-ORDER BY-LIMIT syntax.

WITH Clause. The syntax of the SELECT statement with the WITH clause is

```
WITH
   <alias> AS <Common Table Expression> [,
   <alias> AS <Common Table Expression>, ...,
   <alias> AS <Common Table Expression>
  ]
SELECT ....
```

Example. The Find the day with the highest attendance query can be rewritten as follows:

```
WITH
  a AS  (SELECT eventDate, SUM(attended) as Attended
         FROM Attendance
         GROUP BY eventDate)
SELECT a.eventDate
FROM  a
WHERE Attended = (SELECT MAX(Attended) FROM a )
;
```

[1]Technically, the second occurrence of this query can be simplified, but for this sake of our argument we made both queries the same

1 Conditional Expressions

There are situations when what is returned in the SELECT clause needs to be conditioned by some property/properties of the row that is being covered. For example, consider the following table describing the information about Pokemon:

```
describe Stats;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| PokedexId | int(11)     | NO   |     | NULL    |       |
| name      | varchar(20) | YES  |     | NULL    |       |
| genus     | varchar(20) | YES  |     | NULL    |       |
| type      | varchar(20) | YES  |     | NULL    |       |
| weight    | int(11)     | YES  |     | NULL    |       |
| height    | int(11)     | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

Consider the following information need: for each Pokemon with PokedexId of 10 or lower, output if it is tall or not. Consider a Pokemon tall if their height is greater than 12.

Here is a simple, but somewhat awkward SQL query that will return the correct results:

```
(SELECT Name, 'tall' AS TallOrNot
 FROM Stats
 WHERE PokedexId <= 10 and height > 12
)
UNION
(SELECT Name, 'not tall' AS TallOrNot
 FROM Stats
 WHERE PokedexId  <= 10 and height <= 12);
```

result:

```
+------------+-----------+
| Name       | TallOrNot |
+------------+-----------+
| Charizard  | tall      |
| Blastoise  | tall      |
| Venusaur   | tall      |
| Caterpie   | not tall  |
| Charmander | not tall  |
| Charmeleon | not tall  |
| Squirtle   | not tall  |
| Wartortle  | not tall  |
| Bulbasaur  | not tall  |
| Ivysaur    | not tall  |
+------------+-----------+
```

or a somewhat more involved version if you wanted to sort them by the PokedexId:

```
SELECT Name, TallOrNot
```

```
FROM (
      (SELECT PokedexId, Name, 'tall' AS TallOrNot
       FROM Stats
       WHERE PokedexId <= 10 and height > 12
      )
      UNION
       (SELECT PokedexID, Name, 'not tall' AS TallOrNot
        FROM Stats
        WHERE PokedexId  <= 10 and height <= 12
     )) x
ORDER BY x.PokedexId;
```

result:

```
+------------+-----------+
| Name       | TallOrNot |
+------------+-----------+
| Bulbasaur  | not tall  |
| Ivysaur    | not tall  |
| Venusaur   | tall      |
| Charmander | not tall  |
| Charmeleon | not tall  |
| Charizard  | tall      |
| Squirtle   | not tall  |
| Wartortle  | not tall  |
| Blastoise  | tall      |
| Caterpie   | not tall  |
+------------+-----------+
```

Such a solution becomes quite inconvenient (additional `UNION` clauses) when there are more than two options that need to be considered.

MySQL SQL has two constructs that allow for incorporation of conditional information into the SELECT clause (and elsewhere).

`IF()` built-in function. The `IF()` built-in function takes as input three arguments. The first argument is the condition that needs to be checked. The second argument is the expression that needs to be computed/returned if the condition is true. The third argument is the expression that needs to be computed/returned if the condition is false.

Using this function, we can rewrite the query above as follows:

```
SELECT Name, IF(height > 12, 'tall', 'not tall') as TallOrNot
FROM Stats
WHERE PokedexId <= 10
ORDER BY PokedexID;
```

We can use this function for somewhat more sophisticated outputs. Here is an example where we return either weight of height of the Pokemon based on a the type of the Pokemon:

```
SELECT Name,
       IF( type IN ('fire','water'), weight, height) AS MainQuality,
       IF (type IN ('fire', 'water'), 'W', 'H') as Marker
FROM Stats
WHERE PokedexId <= 10
ORDER BY PokedexId;
```

This query will produce the following output:

```
+------------+-------------+--------+
| Name       | MainQuality | Marker |
+------------+-------------+--------+
| Bulbasaur  |           7 | H      |
| Ivysaur    |          10 | H      |
| Venusaur   |          20 | H      |
| Charmander |          85 | W      |
| Charmeleon |         190 | W      |
| Charizard  |         905 | W      |
| Squirtle   |          90 | W      |
| Wartortle  |         225 | W      |
| Blastoise  |         855 | W      |
| Caterpie   |           3 | H      |
+------------+-------------+--------+
```

`IF()` is a proper built-in MySQL funciton and can be used anywhere in the body of a SQL query where a built-in function could be used. Here is an example of its use in the `WHERE` clause:

```
SELECT *
FROM Stats
WHERE IF(type in ('fire','dark','water'), weight, height) < 30
```

This SQL statement results in returning of all fire, dark and water Pokemon whose weight is less than 30, and all other pokemon whose height is less than 30.

`CASE` expression . For dealing with choices involving more than two options, MySQL offers a `CASE` statement with the following syntax:

```
CASE  WHEN <expression> THEN <expression>[,
      WHEN <expression> THEN <expression>, ...
      WHEN <expression> THEN <expression> ]
      [ELSE <expression>]
END
```

The `CASE` expression is evaluated in a straightforward way. MySQL evaluates one after another the expressions in the `WHEN` clauses of the `CASE` statement. When it evaluates the expression in the `WHEN` clause as *true*, it computes the result of the expression in the corresponding `THEN` clause. If there is an `ELSE` clause, and no expressions in the `WHEN` clauses were evaluated as *true*, the `ELSE` clause expression is evaluated and returned.

Here is a rewrite of the query about tall and not tall Pokemon using the `CASE` statement.

```
SELECT Name, CASE WHEN height >= 12 THEN 'tall'
                  ELSE 'not tall'
             END as TallOrNot
FROM Stats
WHERE PokedexId <= 10
ORDER BY PokedexId;
```

Note 1: The expressions in the `WHEN` clauses don't have to be disjoint, only the first one that is true on a given tuple will trigger a result.

Here is an example.

```
SELECT name, CASE WHEN type = 'grass' THEN 'Plant'
                  WHEN type = 'fire' THEN 'Burns'
                  WHEN height > 10 THEN 'Tall'
                  ELSE 'Short'
             END  as flightStatus,
       height, type
FROM Stats
where pokedexId <= 10;
```

returns

```
+------------+--------------+--------+-------+
| name       | flightStatus | height | type  |
+------------+--------------+--------+-------+
| Caterpie   | Short        |      3 | bug   |
| Charmander | Burns        |      6 | fire  |
| Charmeleon | Burns        |     11 | fire  |
| Charizard  | Burns        |     17 | fire  |
| Squirtle   | Short        |      5 | water |
| Wartortle  | Short        |     10 | water |
| Blastoise  | Tall         |     16 | water |
| Bulbasaur  | Plant        |      7 | grass |
| Ivysaur    | Plant        |     10 | grass |
| Venusaur   | Plant        |     20 | grass |
+------------+--------------+--------+-------+
```

while

```
SELECT name, CASE WHEN height > 10 THEN 'Tall'
                  WHEN type = 'grass' THEN 'Plant'
                  WHEN type = 'fire' THEN 'Burns'

                  ELSE 'Short'
             END  as flightStatus,
       height, type
FROM Stats
where pokedexId <= 10;
```

returns

```
+------------+--------------+--------+-------+
| name       | flightStatus | height | type  |
+------------+--------------+--------+-------+
| Caterpie   | Short        |      3 | bug   |
| Charmander | Burns        |      6 | fire  |
| Charmeleon | Tall         |     11 | fire  |
| Charizard  | Tall         |     17 | fire  |
| Squirtle   | Short        |      5 | water |
| Wartortle  | Short        |     10 | water |
| Blastoise  | Tall         |     16 | water |
| Bulbasaur  | Plant        |      7 | grass |
| Ivysaur    | Plant        |     10 | grass |
| Venusaur   | Tall         |     20 | grass |
+------------+--------------+--------+-------+
```

Note 2: If `ELSE` clause is missing and a row does not match any of the condition, a `NULL` value is returned.

For example,

```
SELECT name, CASE
                  WHEN type = 'grass' THEN 'Plant'
                  WHEN type = 'fire' THEN 'Burns'
             END  as Status,
       height, type
FROM Stats
where pokedexId <= 10;
```

returns

```
+------------+--------+--------+-------+
| name       | Status | height | type  |
+------------+--------+--------+-------+
| Caterpie   | NULL   |      3 | bug   |
| Charmander | Burns  |      6 | fire  |
| Charmeleon | Burns  |     11 | fire  |
| Charizard  | Burns  |     17 | fire  |
| Squirtle   | NULL   |      5 | water |
| Wartortle  | NULL   |     10 | water |
| Blastoise  | NULL   |     16 | water |
| Bulbasaur  | Plant  |      7 | grass |
| Ivysaur    | Plant  |     10 | grass |
| Venusaur   | Plant  |     20 | grass |
+------------+--------+--------+-------+
```

Note 3. The `CASE` expression can be used anywhere in the body of SQL SELECT statement where an expression is expected. For example:

```
SELECT *
FROM Stats
WHERE CASE WHEN type in ('dark', 'water', 'grass') THEN weight
           ELSE height
      END  < 30
```

will return the dark, water and grass Pokemon with weight less than 30 and all other pokemon with height less than 30.