PostgreSQL vs The Standard

Vik Fearing — July 18, 2023

Abridged

What Is the SQL Standard?

Is this valid Standard SQL?

```
CREATE INDEX ON tab (col)
```

Yes!

```
<directly executable statement> ::=
    <direct SQL data statement>
  | <SQL schema statement>
  | <SQL transaction statement>
  | <SQL connection statement>
  | <SQL session statement>
  | <direct implementation-defined statement>
```

```
<direct implementation-defined statement> ::=
  !! See the Syntax Rules.
```

The Format and Syntax Rules for

`<direct implementation-defined statement>`

are implementation-defined.

The SQL Standard

How is it pronounced?

The name “SQL” is correctly pronounced “ess cue ell,” instead of the somewhat common “sequel”.

SQL, specified in the parts of the ISO/IEC 9075 series, is a database language (more precisely, a data sublanguage) used for access to pseudo-relational databases that are managed by pseudo-relational database management systems (RDBMS).

What does it mean?

Many books and articles “define” SQL by parenthetically claiming that the letters stand for “Structured Query Language”. While this was undoubtedly true for the original prototypes, it is not true of the standard.

When the letters appear in product names, they have often been assigned this meaning by the product implementors, but users are ill-served by claims that the word “structured” accurately describes the language.

In the SQL standard, the letters do not stand for anything at all.

Things the Standard gets wrong

Trigger Execution Order

The order of execution *of a set of triggers is ascending by value of their timestamp of creation in their descriptors, such that the oldest trigger executes first.*

`timestamp with time zone`

The time zone displacement is constant throughout a time zone, changing at the beginning and end of Summer Time, where applicable.

Inconsistencies

For reasons lost to history, we have

`COUNT(*)`

So why don't we have

`ROW_NUMBER(*) OVER (ORDER BY ...)`

?

Things PostgreSQL gets wrong

CREATE SCHEMA (1/5)

```
CREATE SCHEMA s
    CREATE TABLE t1 (...)
    CREATE TABLE t2 (...)
    CREATE VIEW v AS ...
```

CREATE SCHEMA (2/5)

	PostgreSQL	Standard SQL
Tables	✓	✓
Views	✓	✓
Domains		✓
Character Sets		✓
Collations		✓

CREATE SCHEMA (3/5)

	PostgreSQL	Standard SQL
Transliterations		✓
Assertions		✓
Triggers	✓	✓
Types		✓
Casts		✓

CREATE SCHEMA (4/5)

	PostgreSQL	Standard SQL
Orderings		✓
Transforms		✓
Routines		✓
Sequences	✓	✓
Grants	✓	✓

CREATE SCHEMA (5/5)

	PostgreSQL	Standard SQL
Roles		✓
Indexes	✓	

ANY aggregate

```
-- Standard SQL
SELECT …
FROM t
GROUP BY …
HAVING ANY (col < 42)
```

```
-- PostgreSQL
SELECT …
FROM t
GROUP BY …
HAVING bool_or(col < 42)
```

```
x = ANY (ARRAY[...])
```

CASE expressions

```
-- Standard and PostgreSQL
CASE expr
  WHEN x THEN ...
  WHEN y THEN ...
  WHEN z THEN ...
ELSE
    ...
END
```

```
-- PostgreSQL
CASE
  WHEN expr && x    THEN ...
  WHEN expr -|- y   THEN ...
  WHEN expr IS NULL THEN ...
ELSE
  ...
END
```

CASE expressions

```
-- Standard SQL
CASE expr
  WHEN < x      THEN ...
  WHEN < y      THEN ...
  WHEN IS NULL THEN ...
ELSE
    ...
END
```

```
DROP DOMAIN ... CASCADE
```

COMING
SOON!

JSON_TABLE

```
SELECT jt.*
FROM my_films
CROSS JOIN LATERAL JSON_TABLE (
  my_films.js,
  '$.favorites[*]'
  COLUMNS (
    id FOR ORDINALITY,
    kind text PATH '$.kind',
    NESTED PATH '$.films[*]' COLUMNS (
      title text PATH '$.title',
      director text PATH '$.director'))) AS jt
```

New Standard Features in PostgreSQL 16

Enhanced Integer Literals

```
1_000_000

0xdead_beef

0o777

0b1001_0110
```

New Standard Features in PostgreSQL 15

UNIQUE NULLS [NOT] DISTINCT

```
INSERT INTO t (unique_col) VALUES (1);
INSERT INTO t (unique_col) VALUES (1); -- ERROR


INSERT INTO t (unique_col) VALUES (NULL);
INSERT INTO t (unique_col) VALUES (NULL); -- ERROR?
```

MERGE

```
MERGE INTO target
USING source ON ...
WHEN MATCHED AND ... THEN
  UPDATE ...
WHEN NOT MATCHED AND ... THEN
  INSERT ...
```

New Standard Features in PostgreSQL 14

SEARCH and CYCLE clause

The `SEARCH` clause adds a column to recursive with list elements to enable `ORDER  BY` to sort the rows as if it were a breadth-first or depth-first search. The execution does not change, just the ability to sort.

The `CYCLE` clause adds a mechanism to automatically stop recursion when a cycle is detected in graph traversal.

Features That Should Not Be Difficult

NEXT VALUE FOR

Used to get the next value for a sequence.

```
-- PostgreSQL
SELECT nextval('seq');

-- Oracle
SELECT seq.nextval FROM dual;

-- Standard SQL
VALUES (NEXT VALUE FOR seq);
```

CAST with Formatting

```
CAST('07/18/2023' AS DATE FORMAT 'MM/DD/YYYY')
```

CORRESPONDING

```
SELECT * FROM a
UNION ALL CORRESPONDING
SELECT * FROM b
```

CORRESPONDING BY (...)

```
SELECT * FROM a
UNION ALL CORRESPONDING BY (x, y, z)
SELECT * FROM b
```

VARCHAR Limits

```
col1 CHARACTER VARYING(100)

col2 CHARACTER VARYING(100 CHARACTERS)

col3 CHARACTER VARYING(100 OCTETS)
```

Interesting Predicates

<unique predicate>

```
SELECT *
FROM orders
WHERE UNIQUE (
  SELECT suppliers.id
  FROM suppliers
  WHERE orders.supplier_id = suppliers.id
)
```

`<match predicate>`

```
WHERE (a, b, c) MATCH UNIQUE PARTIAL (subquery)
```

Pipe Dreams

Assertions

A CHECK Constraint that works across the whole database instead of just on one row of a table.

Row Pattern Recognition

```
SELECT *
FROM t MATCH_RECOGNIZE (
  PARTITION BY ...
  ORDER BY ...
  MEASURES ...
  PATTERN (x y* z)
  DEFINE x AS ..., y AS ..., z AS ...
)
```

Property Graph Queries

New in SQL:2023, this provides a simpler way of doing graph traversal.

```
SELECT * FROM GRAPH_TABLE (
  (a) -[:searched]-> (x) <-[:searched]- (b),
  (b) -[:bought]-> (y)
  WHERE a.name = 'Alice'
  COLUMNS y.name
)
```

Polymorphic Table Functions

A way to call a function when the shape of the output is unknown at query time. Useful for reading a CSV file, for example.

Periods

```
CREATE TABLE t (
  Id INTEGER GENERATED ALWAYS AS IDENTITY,
  name text NOT NULL,
  valid_from TIMESTAMP WITH TIME ZONE,
  valid_to TIMESTAMP WITH TIME ZONE,
  PERIOD validity (valid_from, valid_to)
)
```

System Versioning

Uses a special `SYSTEM_TIME` period to keep all historical rows, which can then be queried.

```
SELECT *
FROM t AS OF '2023-07-18'
```

Vik Fearing

Change “and and” to “and”.