More Than a Query Language: SQL in the 21st Century

Markus Winand, winand.at
SQL Renaissance Ambassador

The only PHP conference in Russia

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

SEQUEL: A STRUCTURED
by
D. Chamberlin
F. Boyce
Laboratory
California
1974
1992

SQL-92 — Tied to the Relational Idea

SQL-92 — Tied to the Relational Idea

Relational Data Model

- “Atomic” types (domain)

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)

Atom image: https://commons.wikimedia.org/wiki/File:Stylised_atom_with_three_Bohr_model_orbits_and_stylised_nucleus.png

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)

A	B	C

Atom image: https://commons.wikimedia.org/wiki/File:Stylised_atom_with_three_Bohr_model_orbits_and_stylised_nucleus.png

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)

A	B	C

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)
- Schema independent of processing purposes
  - "Normalization"

(S Informal Review Draft) ISO/IEC 9075:1992, Database
Language SQL- July 30, 1992

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)
- Schema independent of processing purposes
  - "Normalization"

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)
- Schema independent of processing purposes
  - "Normalization"

Relational Operations

- Transform data for each particular processing purposes
  - `JOIN`, `UNION`, nesting, ...

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)
- Schema independent of processing purposes
  - "Normalization"

Relational Operations

- Transform data for each particular processing purposes
  - `JOIN`, `UNION`, nesting, ...

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)
- Schema independent of processing purposes
  - "Normalization"

Relational Operations

- Transform data for each particular processing purposes
  - `JOIN`, `UNION`, nesting, ...

SQL-92 — Tied to the Relational Idea

Relational Data Model

- "Atomic" types (domain)
- Schema independent of processing purposes
  - "Normalization"

Relational Operations

- Transform data for each particular processing purposes
  - `JOIN`, `UNION`, nesting, ...

SEQUEL: A STRUCTURED
by
D. Chamberlin
F. Boyce
Laboratory
California
1992
1999

Whitemarsh
Information Systems Corporation

The Relational Data Model is Dead!
Great News,

https://www.wiscorp.com/DBMS_-_GreatNews-TheRelationalModelIsDead_-_paper_-_sam.pdf

SQL:1999 — Escaping the Relational Cage

To say that these SQL:1999 extensions are mere "extended interpretations" of the relational data model is like saying that an intercontinental ballistic missile is merely an "extended interpretation" of a spear.

https://www.wiscorp.com/DBMS_-_GreatNews-TheRelationalModelIsDead_-_paper_-_sam.pdf

SQL:1999 — Escaping the Relational Cage

To say that these SQL:1999 extensions are mere "extended interpretations" of the relational data model is like saying that an intercontinental ballistic missile is merely an "extended interpretation" of a spear.

With SQL/99 you can get the best of both worlds and of course, you can get the worst of both worlds. It's up to the database practitioners to do the right thing.

https://www.wiscorp.com/DBMS_-_GreatNews-TheRelationalModelIsDead_-_paper_-_sam.pdf

SQL:1999 — Escaping the Relational Cage

SQL:1999 — Escaping the Relational Cage

Relational Model?

SQL:1999 — Escaping the Relational Cage

Relational Model?

I was as confused as anyone else

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

?

I was as confused as anyone else

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

I was as confused as anyone else ⚛?
By the early 1990s, however,
I'd seen the light

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

I was as confused as anyone else ⚛?
By the early 1990s, however,
I'd seen the light

Domains Can Contain Anything!

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

- Introduced rich types

I was as confused as anyone else ⚛?
By the early 1990s, however,
I'd seen the light

Domains Can Contain Anything!

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

- Introduced rich types

I was as confused as anyone else ⚛?
By the early 1990s, however,
I'd seen the light

Domains Can Contain Anything!

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

- Introduced rich types
  - arrays

A	B
⚛	[⚛, ⚛]
⚛	[⚛]
⚛	[]

I was as confused as anyone else ⚛?
By the early 1990s, however,
I'd seen the light

Domains Can Contain Anything!

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

- Introduced rich types
  - arrays
  - Nested tables (multiset)

I was as confused as anyone else ?
By the early 1990s, however,
I'd seen the light

Domains Can Contain Anything!

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

- Introduced rich types
  - arrays
  - Nested tables (multiset)
  - composite types (objects)

A	B	C	D
	[,]	C D	{x: , y: }
	[ ]	C D	{x: , y: }
	[]	C D	{x: , y: }

I was as confused as anyone else ?
By the early 1990s, however,
I'd seen the light

Domains Can Contain Anything!

Chris Date

Date on Database: Writings 2000-2006

SQL:1999 — Escaping the Relational Cage

Relational Model?

- Introduced rich types
  - arrays
  - Nested tables (multiset)
  - composite types (objects)

Non-Relational Operations

- Introduced recursive queries that process their own output
  - Transitive closure

I was as confused as anyone else ⚛?
By the early 1990s, however,
I'd seen the light

Domains Can Contain Anything!

Date on Database: Writings 2000-2006

Chris Date

SQL:1999 — Recursion

```
CREATE TABLE t (
    id     INTEGER,
    parent INTEGER,
)
```

SQL:1999 — Recursion

```
CREATE TABLE t (
    id     INTEGER,
    parent INTEGER,
)
```

SQL:1999 — Recursion

```
CREATE TABLE t (
    id     INTEGER,
    parent INTEGER,
)
```

SQL:1999 — Recursion

SQL:1999 — Recursion

SQL:1999 — Recursion

```
SELECT t.id, t.parent
  FROM t
 WHERE t.id = ?
```

SQL:1999 — Recursion

```
  SELECT t.id, t.parent
    FROM t
   WHERE t.id = ?
UNION ALL
  SELECT t.id, t.parent
    FROM t
   WHERE t.parent = ?
```

SQL:1999 — Recursion

```
  SELECT t.id, t.parent
    FROM t
   WHERE t.id = ?
UNION ALL
  SELECT t.id, t.parent
    FROM t
   WHERE t.parent = ?
```

SQL:1999 — Recursion

```
  SELECT t.id, t.parent
    FROM t
   WHERE t.id = ?
UNION ALL
  SELECT t.id, t.parent
    FROM t
   WHERE t.parent = ?
```

SQL:1999 — Recursion

```
WITH RECURSIVE prev (id, parent) AS (
  SELECT t.id, t.parent
    FROM t
   WHERE t.id = ?
UNION ALL
  SELECT t.id, t.parent
    FROM t
    JOIN prev ON t.parent = prev.id
)
SELECT * FROM prev
```

SQL:1999 — Recursion

```
WITH RECURSIVE prev (id, parent) AS (
  SELECT t.id, t.parent
    FROM t
   WHERE t.id = ?
UNION ALL
  SELECT t.id, t.parent
    FROM t
    JOIN prev ON t.parent = prev.id
)
SELECT * FROM prev
```

SQL:1999 — Recursion

```
WITH RECURSIVE prev (id, parent) AS (
  SELECT t.id, t.parent
    FROM t
   WHERE t.id = ?
UNION ALL
  SELECT t.id, t.parent
    FROM t
    JOIN prev ON t.parent = prev.id
)
SELECT * FROM prev
```

SQL:1999 — Recursion

Whitemarsh
Information Systems Corporation

The Relational Data Model is Dead!

Great News,

Whitemarsh
Information Systems Corporation
Great News,
The Relational Data Model is Dead!

LATERAL

SQL:1999 — LATERAL

Select-list sub-queries must be ***scalar***[0]:

(return no more than one domain value)

```
SELECT …
     , (SELECT column_1
          FROM t1
         WHERE t1.x = t2.y
       ) AS c
  FROM t2
    …
```

SQL:1999 — LATERAL

Select-list sub-queries must be ***scalar***[0]:
(return no more than one domain value)

```
SELECT …
     , (SELECT column_1, column_2
          FROM t1
         WHERE t1.x = t2.y
       ) AS c
  FROM t2
    …
```

More than one column?
⇒ Syntax error

SQL:1999 — LATERAL

Select-list sub-queries must be ***scalar***[0]:
(return no more than one domain value)

```
SELECT …
     , (SELECT column_1, column_2
          FROM t1
         WHERE t1.x = t2.y
       ) AS c
  FROM t2
    …
```

More than one row?
⇒ Runtime error!

More than one column?
⇒ Syntax error

SQL:1999 — LATERAL

Lateral derived tables lift both limitations <u>and</u> can be correlated:

```
SELECT …
     , ldt.*
  FROM t2
 CROSS JOIN LATERAL (SELECT column_1, column_2
                        FROM t1
                       WHERE t1.x = t2.y
                     ) AS ldt
   …
```

SQL:1999 — LATERAL

Lateral derived tables lift both limitations and can be correlated:

```
SELECT …
     , ldt.*
  FROM t2
 CROSS JOIN LATERAL (SELECT column_1, column_2
                       FROM t1
                      WHERE t1.x = t2.y
                    ) AS ldt
   …
```

"Derived table" means it's in the FROM/JOIN clause

SQL:1999 — LATERAL

Lateral derived tables lift both limitations <u>and</u> can be correlated:

```
SELECT …
     , ldt.*
  FROM t2
 CROSS JOIN LATERAL (SELECT column_1, column_2
                       FROM t1
                      WHERE t1.x = t2.y
                    ) AS ldt
…
```

"Derived table" means it's in the FROM/JOIN clause

Still "correlated"

SQL:1999 — LATERAL

Lateral derived tables lift both limitations and can be correlated:

```
SELECT …
     , ldt.*
  FROM t2
 CROSS JOIN LATERAL (SELECT column_1, column_2
                       FROM t1
                      WHERE t1.x = t2.y
                    ) AS ldt
…
```

"Derived table" means it's in the FROM/JOIN clause

Regular join semantics

Still "correlated"

SQL:1999 — LATERAL

SQL:1999 — LATERAL

- <u>Top-N per group</u>

  inside a lateral derived table
  `FETCH FIRST` (or `LIMIT`, `TOP`)
  applies per row from left tables.

SQL:1999 — LATERAL

- <u>Top-N per group</u>

  inside a lateral derived table
  `FETCH FIRST` (or `LIMIT`, `TOP`)
  applies per row from left tables.

```
FROM t
JOIN LATERAL (SELECT …
                FROM …
               WHERE t.c=…
               ORDER BY …
               LIMIT 10
             ) derived_table
```

SQL:1999 — LATERAL

- <u>Top-N per group</u>

  inside a lateral derived table
  `FETCH FIRST` (or `LIMIT`, `TOP`)
  applies per row from left tables.

Add proper index for Top-N query

```
FROM t
JOIN LATERAL (SELECT …
                FROM …
               WHERE t.c=…
               ORDER BY …
               LIMIT 10
             ) derived_table
```

https://use-the-index-luke.com/sql/partial-results/top-n-queries

SQL:1999 — LATERAL

- <u>Top-N per group</u>

  inside a lateral derived table `FETCH FIRST` (or `LIMIT`, `TOP`) applies per row from left tables.

  - Also useful to find most recent news from several subscribed topics (“multi-source top-N”).

Add proper index for Top-N query

```
FROM t
JOIN LATERAL (SELECT …
                FROM …
               WHERE t.c=…
               ORDER BY …
               LIMIT 10
             ) derived_table
```

https://use-the-index-luke.com/sql/partial-results/top-n-queries

SQL:1999 — LATERAL

- Top-N per group

  inside a lateral derived table `FETCH FIRST` (or `LIMIT`, `TOP`) applies per row from left tables.

  - Also useful to find most recent news from several subscribed topics (“multi-source top-N”).

```
FROM t
JOIN LATERAL (SELECT …
                FROM …
               WHERE t.c=…
               ORDER BY …
               LIMIT 10
             ) derived_table
```

Add proper index for Top-N query

https://use-the-index-luke.com/sql/partial-results/top-n-queries

- Table function arguments

  (`TABLE` often implies `LATERAL`)

```
FROM t
JOIN TABLE (your_func(t.c))
```

SQL:1999 — LATERAL

[0]Undocumented. Requires setting trace event 22829.
[1]`LATERAL` is not supported as of SQL Server 2016 but `[CROSS|OUTER] APPLY` can be used for the same effect.

Whitemarsh
Information Systems Corporation
1999
2003

SQL/XML is Making Good Progress

Andrew Eisenberg
IBM, Westford, MA 01886
andrew.eisenberg@us.ibm.com

http://www.acm.org:80/sigmod/record/issues/0206/standard.pdf (via Wayback machine)

SQL:2003 — Schemaless & Analytical

Schemaless

- Introduced XML
  - Non-uniform documents in a single column

SQL:2003 — Schemaless & Analytical

Schemaless

- Introduced XML
  - Non-uniform documents in a single column

Later:

- JSON added with SQL:2016
- Proprietary JSON support:
  - 2012: PostgreSQL
  - 2014: Oracle
  - 2015: MySQL
  - 2016: SQL Server

SQL:2003 — Schemaless & Analytical

Schemaless

- Introduced XML
  - Non-uniform documents in a single column

Analytical

- Introduced window functions
  - Accessing other rows of the current result

Later:

- JSON added with SQL:2016
- Proprietary JSON support:
  - 2012: PostgreSQL
  - 2014: Oracle
  - 2015: MySQL
  - 2016: SQL Server

SQL:2003 — Schemaless & Analytical

Schemaless

- Introduced XML
  - Non-uniform documents in a single column

Later:

- JSON added with SQL:2016
- Proprietary JSON support:
  - 2012: PostgreSQL
  - 2014: Oracle
  - 2015: MySQL
  - 2016: SQL Server

Analytical

- Introduced window functions
  - Accessing other rows of the current result

Later:

- Extended in SQL:2011
- Popular among “New SQLs”
  - 2013: BigQuery, Hive
  - 2014: Impala
  - 2015: Spark SQL
  - 2016: NuoDB, MemSQL, Cockroach DB, VoltDB

SQL:2003 — Analytical

```
SELECT id, value

  FROM t
```

id	value
1	+10
2	+20
3	-10
4	+50
5	-30
6	-20

SQL:2003 — Analytical

```
SELECT id, value

  FROM t
```

id	value	bal
1	+10	
2	+20	
3	-10	
4	+50	
5	-30	
6	-20	

SQL:2003 — Analytical

```
SELECT id, value
  FROM t
```

id	value	**bal**
1	+10	**+10**
2	+20	
3	-10	
4	+50	
5	-30	
6	-20	

SQL:2003 — Analytical

```
SELECT id, value



  FROM t
```

id	value	**bal**
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (



       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id



       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id



       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
             UNBOUNDED PRECEDING

       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING

       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	+10
2	+20	+30
3	-10	+20
4	+50	+70
5	-30	+40
6	-20	+20

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	**+10**
2	+20	**+30**
3	-10	**+20**
4	+50	**+70**
5	-30	**+40**
6	-20	**+20**

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	+10
2	+20	+30
3	-10	+20
4	+50	+70
5	-30	+40
6	-20	+20

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	+10
2	+20	+30
3	-10	+20
4	+50	+70
5	-30	+40
6	-20	+20

SQL:2003 — Analytical

```
SELECT id, value
     , SUM(value)
       OVER (
       ORDER BY id
        ROWS BETWEEN
            UNBOUNDED PRECEDING
          AND CURRENT ROW
       ) bal
  FROM t
```

id	value	bal
1	+10	+10
2	+20	+30
3	-10	+20
4	+50	+70
5	-30	+40
6	-20	+20

SQL:2003 — Analytical

[0]Without framing

SQL:2003 — Analytical

[0]Without framing

SQL/XML is Making Good Progress
Andrew Eis
IBM, Westford
andrew.eisenberg@us.ibm.c
the creation of a
ifications
2003
2016

SQL:2016 — JSON

http://standards.iso.org/ittf/PubliclyAvailableStandards/c067367_ISO_IEC_TR_19075-6_2017.zip

SQL:2016 — JSON

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

http://standards.iso.org/ittf/PubliclyAvailableStandards/c067367_ISO_IEC_TR_19075-6_2017.zip

SQL:2016 — JSON

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

http://standards.iso.org/ittf/PubliclyAvailableStandards/c067367_ISO_IEC_TR_19075-6_2017.zip

SQL:2016 — JSON_TABLE

```
SELECT *
  FROM JSON_TABLE
       ( ?
       , '$[*]'
         COLUMNS
         ( id INT          PATH '$.id'
         , a1 VARCHAR(…) PATH '$.a1'
         )
       ) r
```

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

SQL:2016 — JSON_TABLE

```
SELECT *
  FROM JSON_TABLE
       ( ?
         '$[*]'
         COLUMNS
         ( id INT          PATH '$.id'
         , a1 VARCHAR(…) PATH '$.a1'
         )
       ) r
```

Bind Parameter

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

SQL:2016 — JSON_TABLE

```
SELECT *
  FROM JSON_TABLE
       ( ?
         '$[*]'
         COLUMNS
         ( id INT            PATH '$.id'
         , a1 VARCHAR(…) PATH '$.a1'
         )
       ) r
```

Bind Parameter

SQL/JSON Path

- Query language to select elements from a JSON document
- Defined in the SQL standard

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

SQL:2016 — JSON_TABLE

```
SELECT *
  FROM JSON_TABLE
       ( ?
         '$[*]'
         COLUMNS
         ( id INT            PATH '$.id'
         , a1 VARCHAR(…) PATH '$.a1'
         )
       ) r
```

Bind Parameter

SQL/JSON Path

- Query language to select elements from a JSON document
- Defined in the SQL standard

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

SQL:2016 — JSON_TABLE

```
SELECT *
  FROM JSON_TABLE
       ( ?
         '$[*]'
         COLUMNS
         ( id INT              PATH '$.id'
         , a1 VARCHAR(…)       PATH '$.a1'
         )
       ) r
```

Bind Parameter

SQL/JSON Path

- Query language to select elements from a JSON document
- Defined in the SQL standard

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

SQL:2016 — JSON_TABLE — Use Case

```
SELECT *
  FROM JSON_TABLE
       ( ?
       , '$[*]'
         COLUMNS
         ( id INT          PATH '$.id'
         , a1 VARCHAR(…) PATH '$.a1'
         )
       ) r
```

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

SQL:2016 — JSON_TABLE — Use Case

```
INSERT INTO target_table
SELECT *
  FROM JSON_TABLE
       ( ?
       , '$[*]'
         COLUMNS
         ( id INT          PATH '$.id'
         , a1 VARCHAR(…) PATH '$.a1'
         )
       ) r
```

```
[
  {
    "id": 42,
    "a1": "foo"
  },
  {
    "id": 43,
    "a1": "bar"
  }
]
```

id	a1
42	foo
43	bar

SQL:2016 — JSON_TABLE

[0]Ridicoulus limitations: only 'strict $' as row expression
[1]OPENJSON provides similar functionality

SQL:2016 — JSON_TABLE

MDEV-17399
Tagged with
fix/Version: 10.6

Ready for
Committer

[0]Ridicoulus limitations: only 'strict $' as row expression
[1]OPENJSON provides similar functionality

2011
2016
TECHNICAL REPORT
Part 6:
SQL support
Notation (JSON)
Langages de base de donn
de SQL pour JavaScript Object No

SQL:2011 — Time Travelling

http://cs.ulb.ac.be/public/_media/teaching/infoh415/tempfeaturessql2011.pdf

SQL:2011 — Time Travelling

Application Versioning

- Dedicated syntax added
  - When did something happen in the real world?

http://cs.ulb.ac.be/public/_media/teaching/infoh415/tempfeaturessql2011.pdf

SQL:2011 — Time Travelling

Application Versioning

- Dedicated syntax added
  - When did something happen in the real world?

New syntax (excerpt)

- `FOR PORTION OF` in `UPDATE` and `DELETE`
- `WITHOUT OVERLAPS` in `UNIQUE` constraints & `PRIMARY KEYS`
- `[IMMEDIATELY] PRECEDES`, `OVERLAPS` in `WHERE`,`HAVING`,…

SQL:2011 — Time Travelling

Application Versioning

- Dedicated syntax added
  - When did something happen in the real world?

System Versioning

- Fully automatic and (almost) transparent
  - When did we learn about something

New syntax (excerpt)

- `FOR PORTION OF` in `UPDATE` and `DELETE`
- `WITHOUT OVERLAPS` in `UNIQUE` constraints & `PRIMARY KEYS`
- `[IMMEDIATELY] PRECEDES`, `OVERLAPS` in `WHERE`,`HAVING`,…

SQL:2011 — Time Travelling

Application Versioning

- Dedicated syntax added
  - When did something happen in the real world?

New syntax (excerpt)

- `FOR PORTION OF` in `UPDATE` and `DELETE`
- `WITHOUT OVERLAPS` in `UNIQUE` constraints & `PRIMARY KEYS`
- `[IMMEDIATELY] PRECEDES`, `OVERLAPS` in `WHERE`,`HAVING`,…

System Versioning

- Fully automatic and (almost) transparent
  - When did we learn about something

Transparent changes, new syntax for queries

- `INSERT`, `UPDATE` & `DELETE` use the system time automatically
- `SELECT` can use `FOR SYSTEM_TIME AS OF`

SQL:2011 — System Versioning

```
CREATE TABLE t (
   ...
 , from TIMESTAMP(9) GENERATED ALWAYS
                     AS ROW START
 , till TIMESTAMP(9) GENERATED ALWAYS
                     AS ROW END

 , PERIOD FOR SYSTEM_TIME (from, till)
) WITH SYSTEM VERSIONING
```

SQL:2011 — System Versioning

SQL:2011 — System Versioning

```
INSERT INTO t (id, data)
       VALUES (1 , 'X' )
```

id	data	from	till
1	X	10:00	

SQL:2011 — System Versioning

```
INSERT INTO t (id, data)
       VALUES (1 , 'X' )
```

id	data	from	till
1	X	10:00	

```
UPDATE t
   SET data = 'Y'
 WHERE id = 1
```

id	data	from	till
1	X	10:00	11:00
1	Y	11:00	

SQL:2011 — System Versioning

```
INSERT INTO t (id, data)
       VALUES (1 , 'X' )
```

id	data	from	till
1	X	10:00	

```
UPDATE t
   SET data = 'Y'
 WHERE id = 1
```

id	data	from	till
1	X	10:00	11:00
1	Y	11:00	

```
DELETE FROM t
 WHERE id = 1
```

id	data	from	till
1	X	10:00	11:00
1	Y	11:00	12:00

SQL:2011 — System Versioning

id	data	from	till
1	X	10:00	11:00
1	Y	11:00	12:00

SQL:2011 — System Versioning

id	data	from	till
1	X	10:00	11:00
1	Y	11:00	12:00

```
SELECT *
  FROM t
```

id	data	from	till

SQL:2011 — System Versioning

id	data	from	till
1	X	10:00	11:00
1	Y	11:00	12:00

```
SELECT *
  FROM t
```

id	data	from	till

```
SELECT *
  FROM t
   FOR SYSTEM_TIME AS OF
   TIMESTAMP'…10:30:00'
```

id	data	from	till
1	X	10:00	11:00

SQL:2011 — System Versioning

[0]Short term using Flashback.
[1]Flashback Archive. Proprietery syntax.

https://webstore.iec.ch/publication/59685

A **lot** has happened since SQL-92

https://webstore.iec.ch/publication/59685

A **lot** has
happened
since SQL-92

SQL has evolved
beyond
the relational idea

https://webstore.iec.ch/publication/59685

A **lot** has
happened
since SQL-92

SQL has evolved
beyond
the relational idea

If you use SQL for
CRUD operations only,
you are doing it wrong

A **lot** has
happened
since SQL-92

SQL has evolved
beyond
the relational idea

If you use SQL for
CRUD operations only,
you are doing it wrong

https://modern-sql.com
@ModernSQL by @MarkusWinand

Training:
https://winand.at/