Chapter 13: The Sorry Saga of SQL

When the world has once begun to use us ill, it afterwards continues the same treatment with less scruple or ceremony, as men do to a whore.

~ Jonathan Swift

SQL–and its Sequels

It might seem that we are taking an alarming digression here, by suddenly turning our attention to SQL, but we are not. It is the gorilla in the room and it needs to be discussed.

Cast your mind back, if it stretches that far back, to the days before the invention of SQL. There were a number of very capable databases. Aside from IBM's own IMS/DB, there were IDMS, ADABAS, TOTAL, Datacom/DB, and Model 204, and incidentally, all these products are still running in various necks of the woods. However the computer scientists of the day were well aware that all of these excellent products had their own data access methods. Would it not be a wonderful thing if there was but one access method defined perhaps in a query language that could be used for all databases no matter how they physically arranged the data on spinning disks.

And so it came to pass in the early 1970s that Donald D. Chamberlin and Raymond F. Boyce at IBM's San Jose Research Laboratory invented a data access language that they called SEQUEL (Structured English QUEry Language). They had learned about Ted Codd's relational model and set out to create a practical language for manipulating and retrieving data stored in System R, IBM's prototype relational database. The name SEQUEL was later shortened to SQL for trademark reasons.

Through the late 1970s and early 1980s, SQL spread beyond IBM. Oracle (then Relational Software Inc.) shipped the first commercial SQL-based

product in 1979. IBM followed with SQL/DS in 1981 and DB2 in 1983. By the mid-1980s, multiple vendors were shipping SQL databases, each with its own syntax and semantics.

SQL Standards

Well, of course, that wasn't good. And thus the gods of computing decreed that there should be a standards committee. And thus the Accredited Standards Committee X3 (now INCITS) developed the first SQL standard, which was adopted by ANSI in 1986. ISO published a technically identical standard, ISO 9075-1987, the following year. What follows documents the evolution of SQL. Read this only if you're a standards fanatic.

SQL-86

SQL-86 as it came to be known codified the core SELECT/INSERT/UPDATE/DELETE syntax and defined bindings for four host languages: COBOL, FORTRAN, Pascal, and PL/I. It was largely a formalization of the IBM dialect as it was. And that needed some improving.

SQL-89

The first enhancement, SQL-89 (ANSI X3.135-1989 / ISO/IEC 9075:1989) was a minor revision. The principal addition was the inclusion of referential integrity constraints (FOREIGN KEY, REFERENCES). The standard remained a single document.

SQL-92

Thence came SQL-92 (ANSI X3.135-1992 / ISO/IEC 9075:1992) which was a major expansion. It tripled the size of the standard and introduced three conformance levels — Entry, Intermediate, and Full — presciently acknowledging that no vendor was likely to implement the whole thing. The key additions included: JOIN syntax (INNER, LEFT/RIGHT/FULL OUTER, CROSS), CASE expressions, CAST for type conversion, string operations (SUBSTRING, TRIM, concatenation with ||), dynamic SQL, temporary tables, scrollable cursors, and the INFORMATION_SCHEMA.

In practice, SQL-92 Entry Level became the de facto baseline that most vendors eventually claimed to support. Time passed, the millenium was approaching and it was clearly necessary to get a new SQL standard into existence before all the computers in the world failed to fail on 1st January 2000.

SQL:1999

That was SQL:1999 (ISO/IEC 9075:1999). Note here, by the way the radical naming adjustment. They replaced the dash of SQL-92 with a colon as in SQL:1999. This was a substantial expansion. It introduced: regular expression matching, recursive queries (WITH RECURSIVE and Common Table Expressions), triggers, user-defined types (structured types, distinct types), roles for access control, OLAP functions (ROLLUP, CUBE, GROUPING SETS), savepoints, and the procedural language framework (SQL/PSM — Persistent Stored Modules). The three-level conformance structure was superseded by a Core SQL specification plus optional feature packages.

SQL:2003

The fourth in line revision to SQL was SQL:2003 (ISO/IEC 9075:2003). This added window functions (OVER, PARTITION BY, ROW_NUMBER, RANK, DENSE_RANK, LEAD, LAG), the MERGE statement (upsert), sequence generators, XML-related specifications (SQL/XML as Part 14), multiset types, BIGINT and BOOLEAN data types, and the TABLESAMPLE clause. Window functions were arguably the most consequential addition since SQL-92's JOIN syntax.

SQL:2006

The fifth revision was SQL:2006 (ISO/IEC 9075:2006). This was a focused revision that substantially expanded SQL/XML — defining how to import, store, query (via XQuery), and export XML data within SQL databases. Happily, there were no major changes to other parts of the standard.

SQL:2008

The sixth was SQL:2008 (ISO/IEC 9075:2008), a relatively modest revision: INSTEAD OF triggers, the TRUNCATE TABLE statement, enhanced FETCH clause (FETCH FIRST n ROWS ONLY as the standard pagination syntax), and various corrections to the existing standard.

SQL:2011

The seventh was SQL:2011 (ISO/IEC 9075:2011). In this we encounter temporal tables — system-versioned tables that automatically track row history, with temporal query syntax (FOR SYSTEM_TIME AS OF, BETWEEN, FROM...TO). Also added: pipelined DML, enhanced

window functions (CUME_DIST, PERCENT_RANK, NTH_VALUE), and improvements to the CALL statement.

SQL:2016

And then there were eight. SQL:2016 (ISO/IEC 9075:2016) introduced 44 new optional features, dominated by two areas. First, JSON support: functions to create, access, and validate JSON data (JSON_VALUE, JSON_QUERY, JSON_TABLE, JSON_OBJECT, JSON_ARRAY, IS JSON). Second, row pattern recognition (MATCH_RECOGNIZE) for matching sequences of rows against regular expression patterns. Also added: polymorphic table functions, LISTAGG for string aggregation, and various date/time enhancements.

SQL:2023

And finally, although probably not that finally, we arrived at SQL:2023 (ISO/IEC 9075:2023). The most significant addition was a new Part 16: Property Graph Queries (SQL/PGQ), which adds graph query capabilities on top of existing SQL tables — defining graph patterns, path-finding, and graph traversal within the SQL framework. Also added: enhancements to JSON support (JSON_SERIALIZE, simplified JSON syntax), the ANY_VALUE aggregate function, and various corrections throughout.

This brief history tells its own story. The SQL standard has grown from what began as a single concise document in 1986 to a multi-part specification estimated at over 4000 pages – equivalent to roughly 10 substantial novels. Each revision has added new syntax to handle data shapes and query patterns that the original design did not anticipate: recursive queries, XML, JSON, temporal data, window analytics, row pattern matching, graph traversal.

The language has grown because the underlying data model (flat tables of typed tuples) cannot accommodate new requirements without new syntax.

And despite nearly four decades of standardization, no vendor fully implements the SQL standard, and every significant vendor has added proprietary (i.e. non-standard) extensions that their users depend on.

Below is a catalog of the principal SQL dialects, grouped into eight different categories. We do not expect you to read all the entries, maybe just browse them a little. So How Many?

It is difficult to arrive at an exact count of SQLs. It depends on how you draw the boundary between "distinct dialect" and "minor fork." A reasonable tally of actively maintained, materially distinct SQL dialects gives a range somewhere between 40 and 50.

SQLFluff, the SQL linter (static analysis tool that checks SQL code for errors, style violations, and potential problems without actually executing it) currently supports about 30 dialect configurations. Apache Calcite, the SQL framework, defines dialect output for roughly 20 databases. Google Looker supports connections to over 50 SQL-based data sources.

If you include historical and discontinued dialects such as Sybase SQL Anywhere, Netezza SQL, Greenplum SQL, Vertica SQL, ParAccel, Teradata Aster, Pivotal HAWQ, and others, it's 60+. And it's a moving target. Each new cloud data warehouse, streaming engine, and embedded analytical database usually introduces its own SQL variant.

For the record, the following tables provide a fairly comprehensive picture of the current situation as of June (2026).

Traditional Enterprise RDBMS		
Vendor	**Dialect Name**	**Notable divergence**
Oracle	PL/SQL	Proprietary procedural language, CONNECT BY for hierarchies, ROWNUM pagination, NVL instead of COALESCE, (+) outer join syntax, object-relational extensions
Microsoft	T-SQL	TOP for pagination, square bracket quoting, + for string concatenation, proprietary date functions (GETDATE, DATEDIFF), IDENTITY columns, cross-apply
IBM	DB2 SQL	Closest to the standard historically, but has its own procedural extensions, OLAP syntax variations, MQT (Materialized Query Tables), XML extensions
SAP/Sybase	T-SQL variant	Shares ancestry with Microsoft T-SQL (both descend from Sybase SQL Server), but has diverged over time

Traditional Enterprise RDBMS		
Vendor	**Dialect Name**	**Notable divergence**
SAP	SAP HANA SQL	Column-oriented extensions, in-memory-specific syntax, calculation views, spatial extensions
Teradata	Teradata SQL	QUALIFY clause (filtering window function results), proprietary date arithmetic, COLLECT STATISTICS, hash distribution syntax
Informix	Informix SQL	BM acquisition; its own procedural extensions (SPL), collection types, time-series extensions
Open Source RDBMS		
Vendor	**Dialect Name**	**Notable divergence**
PostgreSQL	PL/pgSQL	Generally closest to the standard, but: folds unquoted identifiers to lowercase (standard says uppercase), SERIAL for auto-increment, rich extension ecosystem, array types, JSONB, custom operators, DO blocks
MySQL	SQL	Backtick quoting, LIMIT for pagination, AUTO_INCREMENT, non-standard GROUP BY behavior (historically), IF()/IFNULL(), ENGINE= storage engine selection
MariaDB	MariaDB SQL	MySQL fork with additional divergences: sequences, system-versioned tables, Oracle compatibility mode, window functions added earlier than MySQL
SQLite	SQLite SQL	Type affinity instead of strict types, no ALTER TABLE (beyond basic ADD COLUMN), manifest typing, no RIGHT/FULL OUTER JOIN (until 3.39.0), no stored procedures
CockroachDB	CockroachDB SQL	PostgreSQL-compatible wire protocol but distributed SQL with its own extensions for partitioning, locality, and serializable isolation

Open Source RDBMS		
Vendor	**Dialect Name**	**Notable divergence**
YugabyteDB	YSQL	PostgreSQL-compatible dialect for distributed SQL, plus its YCQL (Cassandra-compatible) interface
TiDB	TiDB SQL	MySQL-compatible dialect for distributed SQL, with divergences in transaction handling and some function behavior
Cloud Data Warehouses		
Vendor	**Dialect Name**	**Notable divergence**
Google	BigQuery SQL	STRUCT and ARRAY as first-class types, UNNEST for array flattening, backtick quoting for projects/datasets, scripting syntax, ML integration (BQML), no UPDATE/DELETE without WHERE, federated query syntax
Snowflake	SQL	VARIANT type for semi-structured data, FLATTEN for nested data, QUALIFY clause, time travel (AT/BEFORE), zero-copy cloning syntax, Snowpark extensions
Amazon	Redshift SQL	PostgreSQL 8.x derived, but: no arrays, limited JSON support, DISTKEY/SORTKEY declarations, WLM queue syntax, COPY/UNLOAD for bulk operations
Databricks	Databricks SQL	Spark SQL heritage, Delta Lake extensions (MERGE, time travel, OPTIMIZE, VACUUM), Unity Catalog syntax, ML functions
Microsoft	Azure Synapse SQL	T-SQL variant with distribution syntax (HASH, ROUND_ROBIN, REPLICATE), external table syntax, PolyBase extensions
Firebolt	Firebolt SQL	PostgreSQL-influenced but with its own aggregating index syntax, UNNEST, and engine management DDL

Streaming and Real Time		
Vendor	**Dialect Name**	**Notable divergence**
Apache Flink	Flink SQL	Streaming extensions: watermarks, event-time windowing (TUMBLE, HOP, SESSION), temporal joins, MATCH_RECOGNIZE, connector DDL
Apache Kafka	KSQL/ksqlDB	Stream/table duality syntax, EMIT CHANGES, window types, persistent queries, connector integration
Materialize	Materialize SQL	PostgreSQL-compatible with streaming extensions: CREATE MATERIALIZED SOURCE, temporal filters, tail queries
RisingWave	RisingWave SQL	PostgreSQL-compatible streaming SQL with CREATE SOURCE, CREATE SINK, watermark syntax
Embedded and Edge		
Vendor	**Dialect Name**	**Notable divergence**
SQLite	SQLite SQL	Dominant embedded SQL dialect
DuckDB	DuckDB SQL	PostgreSQL-influenced analytical dialect: list/struct types, PIVOT/UNPIVOT, friendly SQL extensions (FROM-first queries, column aliases in WHERE), in-process execution
Apache Derby	Derby SQL	Java-embedded, subset of SQL standard with Java stored procedures
New SQL and Distributed		
Vendor	**Dialect Name**	**Notable divergence**
Google Spanner	GoogleSQL	Globally-distributed, strongly-consistent; INTERLEAVE IN PARENT for table hierarchies, ARRAY/STRUCT types, stale reads syntax
VoltDB	VoltDB SQL	Subset of SQL with stored procedure focus, partitioned table syntax
SingleStore (MemSQL)	SingleStore SQL	MySQL-compatible with columnstore extensions, PIPELINE syntax for ingestion, JSON support

Time-Series and Specialized		
Vendor	**Dialect Name**	**Notable divergence**
TimescaleDB	TimescaleDB SQL	PostgreSQL extension with hypertable syntax, continuous aggregates, time_bucket functions
QuestDB	QuestDB SQL	PostgreSQL wire protocol with SAMPLE BY for time-series downsampling, LATEST ON for deduplication
ClickHouse	ClickHouse SQL	PostgreSQL-compatible with streaming extensions: CREATE MATERIALIZED SOURCE, temporal filters, tail queries
RisingWave	RisingWave SQL	Column-oriented analytical dialect: ENGINE= table engines, MATERIALIZED/ ALIAS columns, approximate aggregate functions, array functions, FINAL keyword
Apache Druid	Druid SQL	Read-only analytical SQL translated to native Druid queries; limited DML
Legacy and Niche		
Vendor	**Dialect Name**	**Notable divergence**
Fujitsu	Symfoware SQL	Primarily Japanese market; proprietary extensions
Progress	OpenEdge SQL	ABL integration, proprietary syntax
InterSystems	IRIS SQL / Caché SQL	Object-relational with ObjectScript integration
Actian	Ingres SQL / Vector SQL	Codd-era heritage; QUEL predecessor
Mimer	Mimer SQL	Claims highest SQL standard conformance
ADABAS/ Natural	SQL gateway	Mainframe-era data with SQL access layer
Firebird	Firebird SQL	InterBase fork; EXECUTE BLOCK, generators, domains
H2	H2 SQL	Java embedded; multiple compatibility modes (PostgreSQL, MySQL, Oracle, SQL Server)
HSQLDB	HyperSQL	Java embedded; multiple SQL standard conformance levels

A Modus Operandi

The proliferation of SQL dialects is not accidental. It is a natural consequence of SQL itself. The concept of a single query language was from the get-go a marketing fantasy wrapped in a pipe dream.

Because SQL is a programming language, every genuinely new capability inevitably requires new syntax. Each vendor, pursuing its desired and often innovative direction and riding on its own technical architecture, invents an appropriate syntax to satisfy the user requests and naturally does so without considering the interests of any other SQL users.

Some time later, the standards committee meets and attempts to harmonize the proliferating dialects as best it can. But the committee has no teeth. If it "rules against" a particular innovation the vendor will most likely ignore it, after all the vendor will have an installed base that is already using the new syntax. And, in truth, it is the job of database vendors to innovate and compete.