The Clifford Fact Table: A Quantum-State-Aware Schema for Multidimensional Data Analysis

R.J. Mathews
mail.rjmathews@gmail.com

Draft v14 — January 2026

Contents

1 The Clifford Fact Table: A Quantum-State-Aware Schema for Multidimensional Data Analysis 5

2 Abstract 5

3 1. Introduction 6
3.1 1.1 The Problem: Geometry-Blind Data Architecture 6
3.2 1.2 The Hypothesis: Algebraic Structure as Schema 6
3.3 1.3 The Foundational Vision 6
3.4 1.4 Related Work 6
3.4.1 1.4.1 Geometric Algebra in Spatial Data Systems 6
3.4.2 1.4.2 Clifford Algebra in Computing 7
3.4.3 1.4.3 Quantum and Lie-Algebraic Data Representations 7
3.4.4 1.4.4 Domain-Specific Precedents 7
3.4.5 1.4.5 Gap Analysis 7
3.5 1.5 Paper Organization 8

4 2. Theoretical Framework: The Multivector Fact 8
4.1 2.1 Clifford Algebra Fundamentals 8
4.2 2.2 The Grade Structure: Blades as Data Types 8
4.2.1 2.2.1 The 0-Blade (Scalar) 8
4.2.2 2.2.2 The 1-Blade (Vector) 9
4.2.3 2.2.3 The 2-Blade (Bivector) 9
4.2.4 2.2.4 Higher Grades and the Pseudoscalar 9
4.3 2.3 The Lie-Jordan Decomposition: Separating Incompatibility from Alignment . . . 9
4.3.1 2.3.1 The Fundamental Split 9
4.3.2 2.3.2 Storage Implications 9
4.3.3 2.3.3 Why This Matters for Data 10
4.4 2.4 Generators vs Rotors: The Infinitesimal-Finite Distinction 10
4.4.1 2.4.1 Generators (Lie Algebra Elements) 10
4.4.2 2.4.2 Rotors (Lie Group Elements) 10
4.4.3 2.4.3 The Relationship 11
4.5 2.5 The Spin-Spin Simplification 11

4.6 2.6 Commutator Decomposition for General Matrices . . . 11

5 3. Schema Architecture 11
5.1 3.1 The Axis Tables: Dimensions as Basis Vectors . . . 11
5.1.1 3.1.1 Standard Axis Configuration . . . 12
5.1.2 3.1.2 Axis Table Schema . . . 12
5.1.3 3.1.3 Dynamic Basis Extension . . . 12
5.1.4 3.1.4 Location Keys (Domain-Specific Coordinate Dimensions) . . . 12
5.2 3.2 The Multivector Fact Table (MFT) . . . 13
5.2.1 3.2.1 Core Design Principles . . . 13
5.2.2 3.2.2 Complete Schema . . . 13
5.3 3.3 Specialized Storage Types . . . 15
5.3.1 3.3.1 Quaternion Storage with Canonicalization . . . 15
5.3.2 3.3.2 Quaternion Distance Function . . . 16
5.3.3 3.3.3 SLERP Interpolation . . . 16
5.4 3.4 Domain-Specific Blade Dictionaries . . . 17

6 4. Quantum Computation and Population 18
6.1 4.1 Population Strategies Overview . . . 18
6.2 4.2 State Tomography Mapping (Full but Expensive) . . . 18
6.3 4.3 Classical Shadows (Scalable Low-Grade Population) . . . 18
6.4 4.4 Hadamard Test for Commutator Population . . . 19
6.5 4.5 VQE and Optimization Landscape Population . . . 19
6.6 4.6 Honest Quantum Framing . . . 20

7 5. Proposed Query Language: Geometric SQL (G-SQL) 20
7.1 5.1 Design Philosophy . . . 20
7.2 5.2 Core Operations . . . 20
7.2.1 5.2.1 Grade Projection . . . 20
7.2.2 5.2.2 Geometric Product . . . 20
7.2.3 5.2.3 Rotor Operations . . . 21
7.2.4 5.2.4 Reflection and Projection . . . 21
7.3 5.3 Metric-Aware Queries . . . 21
7.3.1 5.3.1 Quaternion Distance Queries . . . 21
7.3.2 5.3.2 Interpolation Queries . . . 21
7.4 5.4 Diagnostic Queries . . . 22
7.4.1 5.4.1 Commutator-Based Anomaly Detection . . . 22
7.4.2 5.4.2 Lie-Jordan Regime Classification . . . 22
7.4.3 5.4.3 Cross-Product Optimization . . . 22
7.5 5.5 Example Complex Queries . . . 23
7.5.1 5.5.1 Vortex Tracking in CFD . . . 23
7.5.2 5.5.2 Entanglement Detection in Quantum Data . . . 23
7.6 5.6 Query Planning and Cost Model (G-SQL Optimizer) . . . 24
7.6.1 5.6.1 The Optimization Problem . . . 24
7.6.2 5.6.2 Cost Units: Coefficient Counts as a First Approximation . . . 24
7.6.3 5.6.3 Operator Cost Classes . . . 25
7.6.4 5.6.4 Rewrite Rules: Turning Geometry into Scalars . . . 25
7.6.5 5.6.5 Indexing Geometric Predicates . . . 26

7.6.6 5.6.6 Statistics and Selectivity for Arrays . . . 26
7.6.7 5.6.7 Optimizer "Representation Switching" (Key Capability) . . . 26
7.7 5.7 Two-Stage and Approximate Geometric Querying . . . 26
7.7.1 5.7.1 Example: Nearest Orientation Frames . . . 26
7.7.2 5.7.2 Example: "Find anomalies in bivector structure" . . . 27

8 6. Applications and Use Cases 27
8.1 6.1 Power Grid Stability Monitoring (Λ_G) . . . 27
8.2 6.2 Tokamak Plasma Edge Diagnostics (Λ_Δt) . . . 28
8.3 6.3 CFD Adaptive Mesh Refinement (Λ_L) . . . 28
8.4 6.4 Quantum Error Correction Validation (getQore) . . . 28
8.5 6.5 Automotive Sensor Diagnostics (Bonus Application) . . . 29

9 7. Implementation Considerations 29
9.1 7.0 Strategic Decisions for the Final Paper Shape . . . 29
9.2 7.1 Storage Backend Options (Reference vs Portability) . . . 30
9.3 7.2 Index Strategies . . . 30
9.4 7.3 Performance Optimization . . . 30
9.4.1 7.3.1 Precompute Common Invariants . . . 30
9.4.2 7.3.2 Materialized Views for Diagnostics . . . 31
9.4.3 7.3.3 Partitioning . . . 31
9.5 7.4 Data Quality and Validation . . . 31
9.5.1 7.4.1 Constraint Checks . . . 31
9.5.2 7.4.2 Geometric Consistency Checks . . . 32
9.6 7.5 Roadblocks and Mitigation Strategies . . . 32
9.6.1 7.5.1 Scalability of Multivector Storage . . . 32
9.6.2 7.5.2 Interoperability with Existing Systems . . . 32
9.6.3 7.5.3 Quantum Hardware Limitations . . . 33
9.6.4 7.5.4 Generator-Rotor Implementation Overhead . . . 33
9.6.5 7.5.5 Lie-Jordan Decomposition Complexity . . . 34
9.7 7.6 Enhancement Opportunities . . . 34
9.7.1 7.6.1 Equivariant Learning Integration . . . 34
9.7.2 7.6.2 Bivector Ellipse Invariants . . . 34
9.7.3 7.6.3 Spatiotemporal Extensions via Cl(3,1) . . . 34
9.7.4 7.6.4 Sparse Clifford Compression . . . 35
9.8 7.7 Evaluation and Benchmarking Plan . . . 35
9.8.1 7.7.1 Evaluation Questions . . . 35
9.8.2 7.7.2 Baseline Schemas . . . 35
9.8.3 7.7.3 Workloads (Representative Queries) . . . 36
9.8.4 7.7.4 Metrics . . . 36
9.8.5 7.7.5 Analytical Baseline Estimates (Non-Empirical, but Useful) . . . 36
9.8.6 7.7.6 Benchmark Harness (Reproducible Structure) . . . 36
9.9 7.8 Reference Implementation Outline . . . 37
9.9.1 7.8.1 Deliverables . . . 37
9.9.2 7.8.2 Minimal "Day 1 Prototype" Feature Set . . . 37
9.10 7.9 Empirical Results . . . 37
9.10.1 7.9.0 Experimental Setup . . . 37
9.10.2 7.9.1 CFD Ingest and Storage Validation (N = 10,000) . . . 38

9.10.3 7.9.2 CFD Scale-Up Benchmark (N = 1,000,000) . . . 39
9.10.4 7.9.3 QEC Semantic Validation (N = 1,000) . . . 40
9.10.5 7.9.4 Algebraic Identity Validation . . . 41
9.10.6 7.9.5 Summary of Validations . . . 42

10 8. Conclusion **42**
10.1 8.1 Summary of Contributions . . . 42
10.2 8.2 Novelty and Related Work . . . 43
10.3 8.3 Acknowledged Limitations . . . 43
10.4 8.4 From Data Processing to Geometric Insight . . . 43
10.5 8.5 Future Directions . . . 44
10.6 8.6 The Bridge Built . . . 44

11 Acknowledgments **44**

12 References **44**

13 Appendix A: Clifford Algebra Quick Reference **46**
13.1 A.1 Cl(3,0) Basis and Products . . . 46
13.2 A.2 Key Identities . . . 46
13.3 A.3 Generator vs Rotor Summary . . . 46

14 Appendix B: SQL Function Library **46**
14.1 B.1 Minimal PostgreSQL Prototype Implementations . . . 47

15 Appendix C: Proof Sketches for Geometric Consistency **49**
15.1 C.1 Wedge Product as Half the Commutator (Vectors) . . . 49
15.2 C.2 Quaternion Double-Cover and Canonicalization Correctness . . . 49
15.3 C.3 Geodesic Distance Between Rotations from Quaternion Dot Product . . . 50
15.4 C.4 Spin-Spin Commutator Equals Cross Product (Hat Map) . . . 50
15.5 C.5 Rotor Sandwich Preserves Vector Norm (Unit Rotors) . . . 51

16 Appendix D: Reference Implementation Repository Skeleton **51**
16.1 D.1 Suggested Repository Layout . . . 51
16.2 D.2 Minimal Synthetic Data Models . . . 52
16.3 D.3 Validation Tests (Must-Have) . . . 52

17 Document Status **52**
17.1 Key Additions from Literature Review: . . . 53
17.2 Remaining Work . . . 53

Contents

1 The Clifford Fact Table: A Quantum-State-Aware Schema for Multidimensional Data Analysis

R.J. Mathews
Independent Researcher
Chattanooga, TN
mail.rjmathews@gmail.com

Draft v14 — January 2026

---

2 Abstract

Traditional relational and dimensional models (Star/Snowflake schemas) represent data as discrete scalars in independent columns, systematically destroying the inherent geometric and topological relationships between variables. We propose a **Quantum Multivector Schema (QMS)** where the central Fact Table stores Clifford algebra multivectors as first-class data citizens. This architecture preserves grade structure (scalars, vectors, bivectors, trivectors), enables native storage of quantum state amplitudes and correlations, and supports "Geometric Querying" through algebraic operations (rotors, projections, reflections) that would require expensive post-hoc computation in traditional schemas.

To make the proposal testable, we provide a reproducible PostgreSQL 16 reference implementation and validation harness. Empirically, we report microbenchmarks on synthetic CFD telemetry (N=10,000) and QEC syndrome data (N=1,000), and a CFD scale-up benchmark (N=1,000,000) comparing QMS to flat baselines. At N=1,000,000 rows, indexed geometric workloads show 500×–2,242× lower latency with QMS (W2/W5), with ~2.6× higher storage footprint than a mixed-precision flat baseline. The harness also validates a critical semantic claim: nave Euclidean distance on quaternion components violates rotation equivalence (q ≡ -q), while a geodesic metric respects it.

We provide rigorous mathematical foundations distinguishing **generators** (Lie algebra elements, infinitesimal) from **rotors** (Lie group elements, finite transformations), and decompose stored relationships into **Lie** (antisymmetric/bivector) and **Jordan** (symmetric/metric) components. We address practical implementation concerns including quaternion canonicalization, geodesic distance metrics, scalable quantum population via classical shadows, and explicit roadblocks (exponential storage growth, interoperability, quantum hardware limitations) with mitigation strategies.

A comprehensive literature review confirms the novelty of this unified approach: while geometric algebra appears in GIS spatial modeling, cryptographic encoding, and isolated domain applications, no prior work combines multivector storage, quantum-native population, Lie-Jordan decomposition, and cross-domain applications (power grids, fusion plasmas, CFD, quantum error correction) with explicit SQL schemas and Geometric SQL extensions.

Keywords: Clifford algebra, geometric algebra, data warehouse, quantum computing, multivector, Lie-Jordan decomposition, commutator diagnostics

---

3 1. Introduction

3.1 1.1 The Problem: Geometry-Blind Data Architecture

Current “Big Data” architectures—from traditional star schemas to modern columnar stores—treat each column as an independent scalar quantity. When we store a velocity field, we create three columns (v__x, v__y, v__z) that the database considers unrelated. When we store a rotation, we flatten it to Euler angles or a 3×3 matrix, losing the constraint that it lives on SO(3). When we store quantum correlations, we extract scalar expectation values and discard the algebraic structure that generated them.

This is **geometry blindness**: the systematic destruction of mathematical structure at the storage layer, forcing expensive reconstruction during analysis.

Concrete consequences: - Nearest-neighbor queries on rotations use Euclidean distance in $\mathbb{R}^9$, which is mathematically wrong - Correlation analyses recompute relationships that could have been stored directly - Quantum state data loses entanglement structure when flattened to probability vectors - Time-series of orientations exhibit discontinuities from gimbal lock and quaternion double-cover

3.2 1.2 The Hypothesis: Algebraic Structure as Schema

We propose that **Clifford algebra provides the natural mathematical framework** for geometry-aware data storage. By embedding Cl(3,0) (Euclidean 3D), Cl(3,1) (spacetime), or higher-dimensional structures directly into the schema, we can:

1. **Store relationships, not just values:** Bivectors encode correlations as first-class objects
2. **Preserve constraints:** Unit quaternions/rotors stay normalized; generators stay in their Lie algebra
3. **Enable algebraic queries:** Rotations, projections, and reflections become native operations
4. **Unify quantum and classical:** The same schema stores both quantum amplitudes and classical field data

3.3 1.3 The Foundational Vision

This approach traces intellectual lineage to:

- **Grassmann (1844):** The exterior algebra of “extensive magnitudes”
- **Clifford (1878):** Unification of Grassmann’s exterior algebra with Hamilton’s quaternions
- **Dirac (1928):** Spinors and the “geometric reality” of quantum mechanics
- **Hestenes (1966):** Spacetime Algebra and the reformation of physics in geometric terms

The key insight: **geometry is not decoration applied after the fact—it is the fundamental structure that data should preserve.**

3.4 1.4 Related Work

A comprehensive literature review confirms that no prior work fully replicates the integrated schema proposed here. However, several related efforts provide valuable context and validation.

3.4.1 1.4.1 Geometric Algebra in Spatial Data Systems

The closest conceptual relatives appear in geographic information systems (GIS) and spatial databases:

- **GA-based GIS frameworks** use geometric algebra to unify data structures and computational models, enabling outer products for topological relations [13]. This addresses "geometry blindness" but remains domain-specific to spatial data.
- **Conformal Geometric Algebra (CGA) for cadastral data** provides compact multivector representations of boundaries and volumes [14], demonstrating grade-structured storage in practice.
- **Multi-layer geographic modeling** employs blades for multi-temporal simulations, integrating algebraic expressions with geometric constructions [15].

These systems align with QMS's emphasis on storing relationships natively but lack quantum integration or the full Lie-Jordan decomposition.

3.4.2 1.4.2 Clifford Algebra in Computing

Multivector representations appear in specialized computing contexts:

- **Cryptographic applications** use multivector decompositions for homomorphic data concealment [16], demonstrating multivectors as general-purpose encoding tools.
- **Spatiotemporal analysis** employs Clifford algebra for geo-simulation data, treating multivectors as core elements for multi-scale representations [17].
- **Software libraries** including Clifford (R) and CliffordAlgebras.jl (Julia) support multivector operations but focus on computation rather than storage schemas [18, 19].

3.4.3 1.4.3 Quantum and Lie-Algebraic Data Representations

Emerging work connects algebraic structures to quantum data:

- **Equivariant neural networks on Lie algebras** introduce layers for processing matrix-valued data while preserving symmetries [20]—a potential enhancement for G-SQL queries.
- **Group Equivariant Non-Expansive Operators (GENEOs)** unify geometric and topological deep learning via algebraic frameworks [21], suggesting compression strategies for high-grade storage.
- **Multimodal quantum data processing** uses geometric representations for complex problems [22], aligning with QMS's quantum population strategies.

3.4.4 1.4.4 Domain-Specific Precedents

In power systems specifically, GA has been applied to fault analysis via ellipse fitting and bivectors, classifying faults in 3D voltage/current spaces [23]. This validates GA's utility for grid diagnostics (our Λ_G application) but without schema integration.

3.4.5 1.4.5 Gap Analysis

Existing Work	What It Has	What QMS Adds
GIS/CGA schemas	Grade-structured storage	Lie-Jordan split, quantum population
Cryptographic GA	Multivector encoding	Relational schema, SQL queries
Equivariant networks	Lie algebra processing	Database-native operations
Power system GA	Bivector diagnostics	Full fact table, cross-domain unification

The unification of multivector storage, quantum-native population, Lie-Jordan decomposition, and cross-domain applications with explicit SQL schemas distinguishes the Clifford Fact Table as a novel contribution.

3.5 1.5 Paper Organization

- Section 2: Mathematical foundations (Clifford algebra, Lie-Jordan decomposition, generators vs rotors)
- Section 3: Schema architecture (axis tables, multivector fact table, storage types)
- Section 4: Quantum computation and population strategies
- Section 5: Geometric SQL query language
- Section 6: Applications (grid monitoring, fusion, CFD, QEC)
- Section 7: Implementation considerations and roadblocks
- Section 8: Conclusion

---

4 2. Theoretical Framework: The Multivector Fact

4.1 2.1 Clifford Algebra Fundamentals

Definition 2.1 (Clifford Algebra). The Clifford algebra Cl(p,q) over ℝ^{p+q} is the associative algebra generated by orthonormal basis vectors {e_1, …, e_{p+q}} satisfying:

$$e_i e_j + e_j e_i = 2\eta_{ij}$$

where η = diag(+1,…,+1,-1,…,-1) has p positive and q negative entries.

Key algebras for data applications:

Algebra	Signature	Dimension	Application Domain
Cl(2,0)	(+,+)	4	2D graphics, complex analysis
Cl(3,0)	(+,+,+)	8	3D Euclidean (CFD, robotics)
Cl(3,1)	(+,+,+,-)	16	Spacetime (relativity, EM)
Cl(4,1)	(+,+,+,+,-)	32	Conformal 3D (computer vision)

4.2 2.2 The Grade Structure: Blades as Data Types

A general multivector M in Cl(n,0) decomposes by **grade**:

$$M = \underbrace{m_0}_{\text{grade 0}} + \underbrace{m_i e_i}_{\text{grade 1}} + \underbrace{m_{ij} e_i \wedge e_j}_{\text{grade 2}} + \cdots + \underbrace{m_{12\ldots n} e_1 \wedge \cdots \wedge e_n}_{\text{grade n}}$$

Each grade corresponds to a distinct geometric and physical meaning:

4.2.1 2.2.1 The 0-Blade (Scalar)

Storage: Single float
Meaning: Magnitude, probability, count, trace, energy
Examples: - Quantum probability $|\alpha|^2$ - Thermal energy kT - Database COUNT(*) aggregates

Database analog: Traditional numeric columns

4.2.2 2.2.2 The 1-Blade (Vector)

Storage: n floats for Cl(n,0)
Meaning: Directed quantity with magnitude and orientation
Examples: - Velocity field u(x,t) - Gradient ∇f - Bloch vector for qubit state

Critical distinction: Vectors transform covariantly under rotations. Storing (v__x, v__y, v__z) as independent columns loses this constraint.

4.2.3 2.2.3 The 2-Blade (Bivector)

Storage: n(n-1)/2 floats (3 for Cl(3,0))
Meaning: Oriented plane segment, rotation generator, correlation
Examples: - Angular momentum L = r ∧ p - Electromagnetic field F = E + IB (in Cl(3,1)) - **Commutator [A, B]** of operators (antisymmetric part) - Quantum entanglement correlators

The bivector equation:

$$a \wedge b = \frac{1}{2}(ab - ba) = \frac{1}{2}[a, b]$$

For vectors a, b, the wedge product equals half the commutator. The bivector represents the **"area of interaction"** between the two quantities—a geometric measure of their incompatibility.

4.2.4 2.2.4 Higher Grades and the Pseudoscalar

Trivector (grade 3): Oriented volume element
Pseudoscalar (grade n): Highest grade element, encodes handedness/chirality

Example: In Cl(3,0), the pseudoscalar I = $e_1e_2e_3$ satisfies I^2 = -1 and represents the orientation of 3D space. Data with "handedness" (chiral molecules, magnetic helicity) requires this grade.

4.3 2.3 The Lie-Jordan Decomposition: Separating Incompatibility from Alignment

4.3.1 2.3.1 The Fundamental Split

Any product AB of operators decomposes into:

$$AB = \frac{1}{2}[A, B] + \frac{1}{2}\{A, B\}$$

where: - **Commutator (Lie bracket):** [A, B] = AB - BA (antisymmetric) - **Anticommutator (Jordan product):** {A, B} = AB + BA (symmetric)

Physical interpretation:

Component	Algebraic Property	Physical Meaning
[A, B]	Antisymmetric	Incompatibility, twist, rotation
{A, B}	Symmetric	Co-alignment, shared structure

4.3.2 2.3.2 Storage Implications

Definition 2.2 (Lie Diagnostic).

$$\Lambda = \|[A, B]\|_F$$

Definition 2.3 (Jordan Diagnostic).

$$\Gamma = \|\{A, B\}\|_F$$

Definition 2.4 (Lie-Jordan Ratio).

$$\rho = \frac{\Lambda}{\Gamma + \epsilon}$$

A complete relational schema stores **both** Lie and Jordan components, not just one.

4.3.3 2.3.3 Why This Matters for Data

Traditional correlation matrices store **symmetric** information (covariance). But many physical relationships are **antisymmetric**: - Non-commuting quantum observables - Fluid vorticity (curl of velocity) - Electromagnetic field tensor

CFD example: velocity gradient decomposition. In fluid dynamics, the velocity gradient tensor $\nabla\mathbf{u}$ decomposes naturally into Lie and Jordan components:

Component	Tensor	Physical Interpretation
Lie (antisymmetric)	Vorticity tensor $\Omega = \frac{1}{2}(\nabla\mathbf{u} - \nabla\mathbf{u}^T)$	Rotation without shape change
Jordan (symmetric)	Strain rate tensor $S = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$	Deformation without rotation

The commutator diagnostic Λ captures vorticity-dominated flow (turbulent eddies), while the anticommutator diagnostic Γ captures strain-dominated flow (shear layers). A geometry-blind schema storing only $\nabla\mathbf{u}$ as 9 scalars loses this fundamental physical distinction.

A geometry-aware schema must distinguish these fundamentally different relationship types.

4.4 2.4 Generators vs Rotors: The Infinitesimal-Finite Distinction

This is perhaps the most critical practical distinction for implementation.

4.4.1 2.4.1 Generators (Lie Algebra Elements)

Definition 2.5 (Generator). An infinitesimal transformation, living in the tangent space at identity.

Examples: - Angular velocity ω (3 floats) - Vorticity $\nabla \times$ u (3 floats) - Hamiltonian H (determines infinitesimal time evolution)

Storage: For 3D rotations, **3 floats** (axial vector / bivector components)

Key property: Generators **add** under composition of infinitesimal transformations.

4.4.2 2.4.2 Rotors (Lie Group Elements)

Definition 2.6 (Rotor). A finite transformation, an element of the Lie group.

Examples: - Unit quaternion q (4 floats) - Orientation frame - Finite rotation by angle θ about axis $\hat{n}$

Storage: For 3D rotations, **4 floats** (unit quaternion) or **3 floats** (axis-angle, but with singularities)

Key property: Rotors **multiply** under composition; they live on a curved manifold (S^3 for quaternions).

4.4.3 2.4.3 The Relationship

The exponential map connects generators to rotors:

$$R = e^{-\theta B/2} = \cos(\theta/2) - B\sin(\theta/2)$$

where B is a unit bivector (rotation plane) and θ is the angle.

For quaternions and angular velocity:

$$\dot{q} = \frac{1}{2} q \otimes (0, \omega)$$

Critical implementation rule: Store ω as 3 floats (generator), store q as 4 floats (rotor). These are different data types with different operations.

4.5 2.5 The Spin-Spin Simplification

A key computational result for CFD and turbulence applications:

Theorem 2.7. For skew-symmetric matrices $W_1 = \hat{\omega}$ and $W_2 = \hat{\nu}$ (hat map from axial vectors):

$$[W_1, W_2] = \widehat{\omega \times \nu}$$

Corollary 2.8.

$$\|[W_1, W_2]\|_F = \sqrt{2}\,|\omega \times \nu|$$

Implication: The commutator of spin tensors reduces to a **cross product**. This transforms O(n^3) matrix operations into O(n) vector operations—a massive computational simplification.

4.6 2.6 Commutator Decomposition for General Matrices

For matrices A = S + Ω (symmetric + skew):

$$[A, B] = \underbrace{[S, T] + [\Omega, \Phi]}_{\text{skew-symmetric result}} + \underbrace{[S, \Phi] + [\Omega, T]}_{\text{symmetric result}}$$

The full commutator has both symmetric and skew parts. Not everything is a bivector.

Term	Result Symmetry	Computation	Physical Meaning
$[S, T]$	Skew	Matrix commutator	Strain-strain coupling
$[\Omega, \Phi]$	Skew	**Cross product**	Spin-spin coupling
$[S, \Phi]$	Symmetric	Matrix commutator	Strain-spin coupling
$[\Omega, T]$	Symmetric	Matrix commutator	Spin-strain coupling

---

5 3. Schema Architecture

5.1 3.1 The Axis Tables: Dimensions as Basis Vectors

Unlike standard dimensional tables that provide lookup keys, **Axis Tables define the basis vectors** of the Clifford algebra.

5.1.1 3.1.1 Standard Axis Configuration

Axis Table	Basis	Signature	Physical Meaning
Time	e_0	- (in Cl(3,1))	Temporal coordinate
Space	e_1, e_2, e_3	+,+,+	Spatial coordinates
Feature	$e_4, \ldots, e_n$	+	Abstract feature dimensions

5.1.2 3.1.2 Axis Table Schema

```
CREATE TABLE axis_space (
    axis_id INT PRIMARY KEY,
    basis_index INT NOT NULL, -- Which e_i this represents
    label VARCHAR(64), -- Human-readable name
    units VARCHAR(32), -- Physical units
    signature INT DEFAULT 1, -- +1 or -1 for metric
    range_min FLOAT,
    range_max FLOAT,
    UNIQUE(basis_index)
);

-- Example population
INSERT INTO axis_space VALUES
    (1, 1, 'x_position', 'meters', 1, -1000, 1000),
    (2, 2, 'y_position', 'meters', 1, -1000, 1000),
    (3, 3, 'z_position', 'meters', 1, -1000, 1000);
```

5.1.3 3.1.3 Dynamic Basis Extension

The schema supports dynamic addition of basis vectors:

```
-- Add a new feature dimension
INSERT INTO axis_space (axis_id, basis_index, label, units)
VALUES (4, 4, 'temperature', 'kelvin');

-- This extends the algebra from Cl(3,0) to Cl(4,0)
-- Multivector storage must accommodate new grades
```

5.1.4 3.1.4 Location Keys (Domain-Specific Coordinate Dimensions)

The Fact Table uses `location_key` to bind multivectors to a spatial (or abstract) coordinate. In many domains this is the true "grain" of the fact table:

- **CFD:** a mesh cell index `(i,j,k)` plus optional physical coordinates
- **Tokamak:** a flux-surface coordinate (e.g., ψ, θ, ϕ) or a diagnostic channel ID
- **Power grid:** `bus_id`, `line_id`, or substation region keys
- **QEC:** qubit index, patch ID, code distance, or syndrome class

A generic, minimal location dimension can be implemented as:

```
CREATE TABLE axis_space_locations (
    location_id BIGSERIAL PRIMARY KEY,
    domain VARCHAR(32) NOT NULL,
    -- Optional physical coordinates (nullable when location is abstract)
    x FLOAT NULL,
    y FLOAT NULL,
    z FLOAT NULL,
    -- Optional discrete coordinates (mesh indices, channel numbers, etc.)
    i INT NULL,
    j INT NULL,
    k INT NULL,
    label VARCHAR(128) NULL,
    UNIQUE(domain, x, y, z, i, j, k)
);
```

This table intentionally supports both **continuous coordinates** and **discrete indices**, because QMS must unify physical fields and abstract "feature spaces" under one schema.

5.2 3.2 The Multivector Fact Table (MFT)

5.2.1 3.2.1 Core Design Principles

1. **Grade-separated storage:** Columns organized by blade grade
2. **Lie-Jordan separation:** Distinct columns for antisymmetric and symmetric relationships
3. **Generator-rotor distinction:** Different types for infinitesimal vs finite transformations
4. **Sparse high grades:** Nullable columns for rarely-populated higher grades
5. **Precomputed invariants:** Generated columns for common queries

5.2.2 3.2.2 Complete Schema

```
CREATE TABLE multivector_fact (
    -- Identity
    fact_id BIGINT PRIMARY KEY,
    timestamp TIMESTAMPTZ NOT NULL,
    location_key BIGINT REFERENCES axis_space_locations(location_id),

    -- Algebra specification
    algebra_id VARCHAR(32) NOT NULL, -- e.g., 'Cl(3,0)', 'Cl(3,1)'
    basis_version INT NOT NULL, -- schema version for blade ordering
    domain VARCHAR(32), -- 'grid', 'tokamak', 'cfd', 'qec'

    -- NOTE (portability):
    -- Array types like REAL[] and DOUBLE PRECISION[] are PostgreSQL-specific.
    -- Other backends can map these to native vector types, JSON arrays, or wide tables.

    -- PRECISION POLICY:
    -- DOUBLE PRECISION (float8): invariants, ratios, rotors, accumulated diagnostics
    -- REAL (float4): bulk telemetry coefficients where bandwidth dominates

    -- GRADE 0: SCALARS (raw measurements often tolerate REAL)
    scalar_magnitude REAL, -- primary scalar value (pressure, load, etc.)
    scalar_probability REAL, -- quantum probability (if applicable)
    scalar_energy REAL, -- energy/trace-like scalar
    scalar_trace REAL, -- matrix trace
    scalar_determinant REAL, -- matrix determinant
```

```
-- GRADE 1: VECTORS (bulk coefficients typically REAL)
vector_coeffs REAL[], -- full vector (dimension varies)
vector_gradient REAL[3], -- gradient direction (3D)
vector_bloch DOUBLE PRECISION[3], -- Bloch vector (often used in ratios/metrics)

-- Precomputed vector invariants (stored as float8)
vector_norm DOUBLE PRECISION GENERATED ALWAYS AS (
    ARRAY_L2_NORM(vector_coeffs)
) STORED,

-- GRADE 2: BIVECTORS (Antisymmetric / Lie)
bivector_coeffs REAL[], -- full bivector (n(n-1)/2 components)
bivector_xy REAL, -- $e_{1}$$\wedge$$e_{2}$ component
bivector_yz REAL, -- $e_{2}$$\wedge$$e_{3}$ component
bivector_zx REAL, -- $e_{3}$$\wedge$$e_{1}$ component

-- Commutator diagnostics ($\Lambda$) (stored as float8)
lambda_total DOUBLE PRECISION, -- ||[A, ]||_F
lambda_skew DOUBLE PRECISION, -- skew part of commutator
lambda_sym DOUBLE PRECISION, -- symmetric part of commutator

-- Precomputed bivector invariant (stored as float8)
bivector_norm DOUBLE PRECISION GENERATED ALWAYS AS (
    sqrt((bivector_xy::double precision)^2 +
         (bivector_yz::double precision)^2 +
         (bivector_zx::double precision)^2)
) STORED,

-- GRADE 2: SYMMETRIC TENSORS (Jordan)
symmetric_coeffs REAL[], -- symmetric 2-tensor (n(n+1)/2 components)
strain_eigenvalues REAL[3], -- principal values (sorted)
covariance_matrix REAL[6], -- upper triangle of covariance

-- Anticommutator diagnostic ($\Gamma$) (stored as float8)
gamma_total DOUBLE PRECISION, -- ||{A, }||_F

-- Lie--Jordan ratio (stored as float8)
lie_jordan_ratio DOUBLE PRECISION GENERATED ALWAYS AS (
    lambda_total / (gamma_total + 1e-10)
) STORED,

-- GENERATORS (Lie Algebra Elements) --- 3 floats for SO(3)
angular_velocity REAL[3], -- $\omega$ (spin/vorticity)
angular_velocity_dot REAL[3], -- $\omega$

-- Cross-product diagnostic (optimized spin--spin commutator)
-- Stored as float8 even when $\omega$ is float4.
spin_cross_norm DOUBLE PRECISION GENERATED ALWAYS AS (
    SPIN_COMMUTATOR_NORM(angular_velocity, angular_velocity_dot)
) STORED,

-- ROTORS (Lie Group Elements) --- unit quaternions (float8)
orientation_quat DOUBLE PRECISION[4], -- unit quaternion (w, x, y, z)
```

```
    strain_frame_quat DOUBLE PRECISION[4], -- principal strain axes orientation

    -- Quaternion canonicalization check (boolean)
    quat_canonical BOOLEAN GENERATED ALWAYS AS (
       orientation_quat[1] >= 0 OR
       (orientation_quat[1] = 0 AND orientation_quat[2] >= 0)
    ) STORED,

    -- GRADE 3+: TRIVECTORS AND PSEUDOSCALAR (Sparse)
    trivector_coeffs REAL[] NULL, -- oriented volumes
    pseudoscalar REAL NULL, -- handedness/chirality

    -- FULL MULTIVECTOR (optional; heavy)
    full_multivector REAL[] NULL, -- all 2^n coefficients (if needed)

    -- METADATA
    alarm_status VARCHAR(16), -- 'normal', 'warning', 'critical'
    quality_score REAL, -- data quality metric (raw score)
    source_system VARCHAR(64) -- origin of data
);

-- Recommended indices (PostgreSQL syntax shown; adapt as needed):
-- CREATE INDEX ON multivector_fact (timestamp);
-- CREATE INDEX ON multivector_fact (domain);
-- CREATE INDEX ON multivector_fact (lambda_total);
-- CREATE INDEX ON multivector_fact (bivector_norm);
-- CREATE INDEX ON multivector_fact (lie_jordan_ratio);
-- CREATE INDEX ON multivector_fact (spin_cross_norm); -- accelerates W5
-- CREATE INDEX ON multivector_fact (alarm_status);

-- Optional (PostgreSQL expression index): distance-from-identity thresholds for rotors (
    W6)
-- For q0 = identity, d(q,I)=2*acos(|w|), so |w| is sufficient as an indexable surrogate.
-- CREATE INDEX ON multivector_fact ((ABS(orientation_quat[1])));
```

5.3 3.3 Specialized Storage Types

5.3.1 3.3.1 Quaternion Storage with Canonicalization

Problem: q and -q represent the same rotation, breaking nearest-neighbor search and time-series continuity.

Solution: Canonicalization constraint

```
-- Trigger to enforce canonical quaternion form
CREATE FUNCTION canonicalize_quaternion() RETURNS TRIGGER AS $$
BEGIN
    -- Ensure w >= 0 (or w=0 and x>=0, etc.)
    IF NEW.orientation_quat[1] < 0 OR
      (NEW.orientation_quat[1] = 0 AND NEW.orientation_quat[2] < 0) THEN
       NEW.orientation_quat := ARRAY[
           -NEW.orientation_quat[1],
           -NEW.orientation_quat[2],
           -NEW.orientation_quat[3],
           -NEW.orientation_quat[4]
```

```
        ];
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_quat_canonical
    BEFORE INSERT OR UPDATE ON multivector_fact
    FOR EACH ROW EXECUTE FUNCTION canonicalize_quaternion();
```

5.3.2 3.3.2 Quaternion Distance Function

Problem: Euclidean distance in $\mathbb{R}^4 \neq$ rotation distance
Solution: Geodesic distance function

```
CREATE FUNCTION quat_geodesic_distance(q1 DOUBLE PRECISION[4], q2 DOUBLE PRECISION[4])
RETURNS DOUBLE PRECISION AS $$
DECLARE
    dot_product DOUBLE PRECISION;
BEGIN
    dot_product := q1[1]*q2[1] + q1[2]*q2[2] + q1[3]*q2[3] + q1[4]*q2[4];
    -- Clamp to valid range for acos
    dot_product := LEAST(1.0, GREATEST(-1.0, ABS(dot_product)));
    RETURN 2.0 * ACOS(dot_product);
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

5.3.3 3.3.3 SLERP Interpolation

```
CREATE FUNCTION quat_slerp(q1 DOUBLE PRECISION[4], q2 DOUBLE PRECISION[4], t DOUBLE
    PRECISION)
RETURNS DOUBLE PRECISION[4] AS $$
DECLARE
    dot_product DOUBLE PRECISION;
    theta DOUBLE PRECISION;
    sin_theta DOUBLE PRECISION;
    s1 DOUBLE PRECISION;
    s2 DOUBLE PRECISION;
    q2_use DOUBLE PRECISION[4];
BEGIN
    dot_product := q1[1]*q2[1] + q1[2]*q2[2] + q1[3]*q2[3] + q1[4]*q2[4];

    -- If dot < 0, negate q2 to take shorter path
    IF dot_product < 0 THEN
        q2_use := ARRAY[-q2[1], -q2[2], -q2[3], -q2[4]];
        dot_product := -dot_product;
    ELSE
        q2_use := q2;
    END IF;

    -- If very close, use linear interpolation
    IF dot_product > 0.9995 THEN
```

```
        RETURN ARRAY[
            q1[1] + t*(q2_use[1] - q1[1]),
            q1[2] + t*(q2_use[2] - q1[2]),
            q1[3] + t*(q2_use[3] - q1[3]),
            q1[4] + t*(q2_use[4] - q1[4])
        ];
    END IF;

    theta := ACOS(dot_product);
    sin_theta := SIN(theta);
    s1 := SIN((1-t)*theta) / sin_theta;
    s2 := SIN(t*theta) / sin_theta;

    RETURN ARRAY[
        s1*q1[1] + s2*q2_use[1],
        s1*q1[2] + s2*q2_use[2],
        s1*q1[3] + s2*q2_use[3],
        s1*q1[4] + s2*q2_use[4]
    ];
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

5.4 3.4 Domain-Specific Blade Dictionaries

Different domains interpret blades differently:

```
CREATE TABLE blade_dictionary (
    algebra_id VARCHAR(32),
    domain VARCHAR(32),
    blade_index INT,
    grade INT,
    symbol VARCHAR(64),
    physical_meaning TEXT,
    units VARCHAR(32),
    PRIMARY KEY (algebra_id, domain, blade_index)
);

-- CFD domain
INSERT INTO blade_dictionary VALUES
    ('Cl(3,0)', 'cfd', 0, 0, '1', 'Scalar (pressure, density)', 'Pa, kg/m$^{3}$'),
    ('Cl(3,0)', 'cfd', 1, 1, '$e_{1}$', 'x-velocity component', 'm/s'),
    ('Cl(3,0)', 'cfd', 2, 1, '$e_{2}$', 'y-velocity component', 'm/s'),
    ('Cl(3,0)', 'cfd', 3, 1, '$e_{3}$', 'z-velocity component', 'm/s'),
    ('Cl(3,0)', 'cfd', 4, 2, '$e_{12}$', 'xy-vorticity ($\omega$z)', '1/s'),
    ('Cl(3,0)', 'cfd', 5, 2, '$e_{23}$', 'yz-vorticity ($\omega$x)', '1/s'),
    ('Cl(3,0)', 'cfd', 6, 2, '$e_{31}$', 'zx-vorticity ($\omega$y)', '1/s'),
    ('Cl(3,0)', 'cfd', 7, 3, '$e_{123}$', 'Helicity density', 'm/s$^{2}$');

-- Quantum domain
INSERT INTO blade_dictionary VALUES
    ('Cl(3,0)', 'quantum', 0, 0, '1', 'Probability |$\psi$|$^{2}$', ''),
    ('Cl(3,0)', 'quantum', 1, 1, '$e_{1}$', 'Bloch x ($\sigma$)', ''),
    ('Cl(3,0)', 'quantum', 2, 1, '$e_{2}$', 'Bloch y ($\sigma$)', ''),
    ('Cl(3,0)', 'quantum', 3, 1, '$e_{3}$', 'Bloch z ($\sigma$_z)', ''),
    ('Cl(3,0)', 'quantum', 4, 2, '$e_{12}$', 'XY correlation', ''),
    ('Cl(3,0)', 'quantum', 5, 2, '$e_{23}$', 'YZ correlation', ''),
    ('Cl(3,0)', 'quantum', 6, 2, '$e_{31}$', 'ZX correlation', ''),
    ('Cl(3,0)', 'quantum', 7, 3, '$e_{123}$', 'Purity/chirality', '');

-- Power grid domain
INSERT INTO blade_dictionary VALUES
    ('Cl(n,0)', 'grid', 0, 0, '1', 'Total power', 'MW'),
    ('Cl(n,0)', 'grid', -1, 1, 'e', 'Bus voltage deviation', 'pu'),
    ('Cl(n,0)', 'grid', -2, 2, 'e', 'Line flow sensitivity', 'MW/pu');
```

---

6 4. Quantum Computation and Population

6.1 4.1 Population Strategies Overview

Source	Method	Grades Populated	Scalability
Classical simulation	Direct computation	All	O(N)
Quantum tomography	Full state reconstruction	All	O(4) — **impractical**
Classical shadows	Randomized measurement	Low grades	O(log n)
Hadamard test	Commutator estimation	Grade 2 (bivector)	O(1) after encoding
VQE iteration	Gradient + curvature	Grades 1, 2	O(parameters)

6.2 4.2 State Tomography Mapping (Full but Expensive)

For a quantum state ρ, the Pauli decomposition provides a natural mapping to Cl(3,0) for a single qubit:

$$\rho = \frac{1}{2}(I + r_x\sigma_x + r_y\sigma_y + r_z\sigma_z)$$

Mapping to multivector: - Scalar (grade 0): Tr(ρ) = 1 (normalization) - Vector (grade 1): Bloch vector r = (r__x, r__y, r__z) - Bivector (grade 2): Off-diagonal coherences (for mixed states)

Multi-qubit extension: For n qubits, use weight-k Pauli strings: - Weight-0: Identity $\rightarrow$ scalar - Weight-1: Single-qubit Paulis $\rightarrow$ vectors - Weight-2: Two-qubit correlators $\rightarrow$ bivectors - Weight-k: k-body correlators $\rightarrow$ grade-k blades

Scalability problem: Full tomography requires O(4) measurements. For n > 10 qubits, this is impractical.

6.3 4.3 Classical Shadows (Scalable Low-Grade Population)

Key insight from peer review: Full tomography doesn't scale. Use classical shadows instead.

Protocol: 1. Apply random Clifford unitaries 2. Measure in computational basis 3. Reconstruct low-weight observables efficiently

Schema implication: Populate low-grade blades densely, higher grades sparsely or on-demand.

```
-- Schema annotation for population method
ALTER TABLE multivector_fact ADD COLUMN population_method VARCHAR(32);
-- Values: 'tomography', 'shadows', 'hadamard', 'classical', 'vqe'

ALTER TABLE multivector_fact ADD COLUMN shadow_samples INT;
-- Number of shadow samples used (for confidence estimation)
```

6.4 4.4 Hadamard Test for Commutator Population

The Hadamard test circuit directly estimates commutator-related quantities:

```
|0 H H [Measure]

|$\psi$ [e^{iAt}][e^{-iBt}]
```

What it measures: i[A, B] (the Hermitian combination)

Important correction from peer review: - [A, B] for Hermitian A, B is **anti-Hermitian** - i[A, B] is **Hermitian** and thus measurable - This is the generator in Lie algebra terms—exactly what we want for bivector population

Schema population:

```
-- Hadamard test directly populates bivector coefficients
UPDATE multivector_fact
SET bivector_coeffs[k] = hadamard_test_result,
    population_method = 'hadamard'
WHERE fact_id = ?;
```

6.5 4.5 VQE and Optimization Landscape Population

Variational Quantum Eigensolver iterations produce: - **Gradient** E/θ $\rightarrow$ grade-1 (vector) - **Berry curvature** (antisymmetric part of QGT) $\rightarrow$ grade-2 (bivector) - **Quantum Fisher metric** (symmetric part of QGT) $\rightarrow$ symmetric tensor (not bivector!)

Correction from peer review: The Hessian ^{2}E/$\theta\theta$ is **symmetric**, not a bivector. The Berry curvature is the correct antisymmetric/bivector object.

```
-- VQE iteration populates both vector and bivector
INSERT INTO multivector_fact (
    domain,
    vector_gradient,
    bivector_coeffs, -- Berry curvature
    symmetric_coeffs, -- Quantum Fisher metric
    population_method
) VALUES (
    'vqe',
    ?, -- gradient components
    ?, -- Berry curvature components
    ?, -- metric components
    'vqe'
);
```

6.6 4.6 Honest Quantum Framing

What to claim: - Representational alignment: Quantum measurements naturally produce multivector components - Hadamard tests output bivector coefficients directly - Schema is compatible with quantum data sources

What NOT to claim: - "Quantum computers populate the database faster" — data loading is the bottleneck - "Grover search on the database" — requires QRAM which doesn't exist at scale - Automatic speedup for classical data processing

The realistic value proposition: 1. **Near-term:** Schema structure improves classical analytics (better invariants, geometric operations) 2. **Future:** When quantum data sources become common, no impedance mismatch

---

7 5. Proposed Query Language: Geometric SQL (G-SQL)

7.1 5.1 Design Philosophy

G-SQL extends SQL with operations native to Clifford algebra: - **Geometric product:** The fundamental product of GA - **Grade projection:** Extract specific grades from multivectors - **Rotor operations:** Rotate, reflect, project - **Metric queries:** Use geodesic distance for rotors

7.2 5.2 Core Operations

7.2.1 5.2.1 Grade Projection

```sql
-- Extract scalar part
SELECT GRADE(full_multivector, 0) AS scalar_part
FROM multivector_fact;

-- Extract bivector part
SELECT GRADE(full_multivector, 2) AS bivector_part
FROM multivector_fact;

-- Shorthand for common projections
SELECT SCALAR(M), VECTOR(M), BIVECTOR(M), PSEUDOSCALAR(M)
FROM multivector_fact;
```

7.2.2 5.2.2 Geometric Product

```sql
-- Multiply two multivectors
SELECT GEOM_PRODUCT(M1.full_multivector, M2.full_multivector)
FROM multivector_fact M1, multivector_fact M2
WHERE M1.timestamp = M2.timestamp;

-- Inner product (grade-lowering)
SELECT INNER_PRODUCT(M.vector_coeffs, reference_vector)
FROM multivector_fact M;

-- Outer/wedge product (grade-raising)
SELECT WEDGE(M.vector_coeffs, another_vector)
```

```
FROM multivector_fact M;
```

7.2.3 5.2.3 Rotor Operations

```
-- Rotate a vector by a rotor
SELECT ROTATE(vector_coeffs, rotor_quat) AS rotated_vector
FROM multivector_fact;

-- Apply rotor to full multivector: R M R$\dagger$
SELECT SANDWICH(full_multivector, rotor_quat) AS transformed
FROM multivector_fact;

-- Compose rotors
SELECT ROTOR_COMPOSE(R1.orientation_quat, R2.orientation_quat)
FROM multivector_fact R1, multivector_fact R2;
```

7.2.4 5.2.4 Reflection and Projection

```
-- Reflect vector across plane defined by bivector
SELECT REFLECT(vector_coeffs, plane_bivector)
FROM multivector_fact;

-- Project onto subspace
SELECT PROJECT(full_multivector, subspace_blade)
FROM multivector_fact;

-- Rejection (component orthogonal to subspace)
SELECT REJECT(full_multivector, subspace_blade)
FROM multivector_fact;
```

7.3 5.3 Metric-Aware Queries

7.3.1 5.3.1 Quaternion Distance Queries

```
-- Find nearest rotations (using geodesic distance, not Euclidean!)
SELECT fact_id,
      QUAT_DISTANCE(orientation_quat, target_quat) AS rotation_distance
FROM multivector_fact
ORDER BY rotation_distance
LIMIT 10;

-- Clustering with correct metric
SELECT QUAT_KMEANS(orientation_quat, 5) AS cluster_id
FROM multivector_fact;
```

7.3.2 5.3.2 Interpolation Queries

```
-- SLERP between two timestamps
SELECT QUAT_SLERP(
    (SELECT orientation_quat FROM multivector_fact WHERE timestamp = T1),
    (SELECT orientation_quat FROM multivector_fact WHERE timestamp = T2),
    0.5 -- interpolation parameter
) AS midpoint_orientation;

-- Time-series interpolation
SELECT timestamp,
       QUAT_SLERP_SERIES(orientation_quat, timestamp, '1 second')
           OVER (ORDER BY timestamp) AS interpolated
FROM multivector_fact;
```

7.4 5.4 Diagnostic Queries

7.4.1 5.4.1 Commutator-Based Anomaly Detection

```
-- Find high-$\Lambda$ events (potential instabilities)
SELECT fact_id, timestamp, location_key, lambda_total
FROM multivector_fact
WHERE lambda_total > (
    SELECT PERCENTILE_CONT(0.95) WITHIN GROUP (ORDER BY lambda_total)
    FROM multivector_fact
    WHERE timestamp > NOW() - INTERVAL '1 hour'
)
ORDER BY lambda_total DESC;
```

7.4.2 5.4.2 Lie-Jordan Regime Classification

```
-- Classify dynamics regime
SELECT
    fact_id,
    timestamp,
    CASE
        WHEN lie_jordan_ratio > 2.0 THEN 'rotation_dominated'
        WHEN lie_jordan_ratio < 0.5 THEN 'alignment_dominated'
        ELSE 'transitional'
    END AS dynamics_regime,
    lambda_total,
    gamma_total
FROM multivector_fact
WHERE domain = 'cfd';
```

7.4.3 5.4.3 Cross-Product Optimization

```
-- For CFD: use cross-product formula for spin-spin commutator
SELECT
    fact_id,
    spin_cross_norm AS lambda_spin_spin, -- 2|$\omega$ $\times$ $\omega$| (precomputed)
    lambda_total,
    spin_cross_norm / lambda_total AS spin_contribution_fraction
FROM multivector_fact
WHERE domain = 'cfd'
  AND lambda_total > 0;
```

7.5 5.5 Example Complex Queries

7.5.1 5.5.1 Vortex Tracking in CFD

```
-- Find coherent vortex structures
-- (high bivector norm, stable orientation)
WITH vortex_candidates AS (
    SELECT
        fact_id,
        timestamp,
        location_key,
        bivector_norm,
        orientation_quat,
        LAG(orientation_quat) OVER (
            PARTITION BY location_key ORDER BY timestamp
        ) AS prev_quat
    FROM multivector_fact
    WHERE domain = 'cfd'
      AND bivector_norm > 0.1
)
SELECT
    fact_id,
    timestamp,
    location_key,
    bivector_norm,
    QUAT_DISTANCE(orientation_quat, prev_quat) AS orientation_change
FROM vortex_candidates
WHERE QUAT_DISTANCE(orientation_quat, prev_quat) < 0.1 -- Stable orientation
ORDER BY bivector_norm DESC;
```

7.5.2 5.5.2 Entanglement Detection in Quantum Data

```
-- Find highly entangled states
-- (large bivector components indicate 2-body correlations)
SELECT
    fact_id,
    timestamp,
    SQRT(bivector_xy^2 + bivector_yz^2 + bivector_zx^2) AS correlation_strength,
    scalar_probability,
    -- Concurrence estimate from bivector
    2.0 * ABS(bivector_xy) AS concurrence_estimate
FROM multivector_fact
WHERE domain = 'quantum'
  AND bivector_norm > 0.5
ORDER BY bivector_norm DESC;
```

7.6 5.6 Query Planning and Cost Model (G-SQL Optimizer)

This section fills the remaining gap in Section 5: how a database optimizer can plan and cost geometric queries so that GSQL remains practical at scale.

7.6.1 5.6.1 The Optimization Problem

A conventional SQL optimizer is designed around: - scalar predicates (cheap) - joins (expensive but well-modeled) - aggregates (streamable)

GSQL adds new operator families whose computational cost depends on **algebra size**, **grade support**, and **representation choice** (dense multivectors vs sparse vs rotor/generator special cases).

A minimal optimizer for GSQL must therefore answer:

1. *Which representation should be used for a given query?*
   (e.g., rotor/quaternion vs bivector generator vs full multivector)
2. *Which invariants can be exploited to avoid expensive algebraic operations?*
   (e.g., precomputed norms, ratios, and cross-product commutators)
3. *Which indices (if any) can accelerate geometric predicates and similarity search?*

7.6.2 5.6.2 Cost Units: Coefficient Counts as a First Approximation

Let: - (n) be the algebra dimension (Cl(n,0) has (2^n) coefficients), - (k) be the maximum grade populated densely, - (c_k) be the number of coefficients stored for grade (k) (e.g., (c_2=n(n-1)/2)).

For a multivector stored in dense coefficient arrays, a first-order CPU cost model is:

$$\mathrm{Cost}(\mathrm{op}) \approx \alpha_{\mathrm{op}} + \beta_{\mathrm{op}} \cdot N_{\mathrm{coeff}}$$

where (N_{coeff}) depends on the representation actually used by the operator.

Representative coefficient counts:

Representation	Stored Object	Coefficients
Scalar	grade 0	1
3D vector	grade 1 in Cl(3,0)	3
3D bivector (generator)	grade 2 in Cl(3,0)	3
Quaternion / rotor	even subalgebra Cl(3,0)	4
Symmetric 3×3 tensor	(D)	6
Full Cl(3,0) multivector	grades 0–3	8
Full Cl(3,1) multivector	grades 0–4	16

This model is intentionally “rough”: it allows the optimizer to distinguish constant-time rotor operations from coefficient-heavy full multivector products.

Optimization burden shift. In this framework, the performance advantage of generator-vs-rotor representation is realized at *schema design time*, not *query optimization time*. The schema designer chooses to store angular velocity as a 3-float generator (enabling the cross-product identity) rather than as a 3×3 skew-symmetric matrix. PostgreSQL’s planner then sees `spin_cross_norm` as a precomputed scalar column—it need not understand the underlying algebraic simplification (that `SPIN_COMMUTATOR_NORM(`ω`, `$\dot{\omega}$`)` reduces from $O(54)$ operations to $O(6)$ via the cross-product identity).

This is a deliberate trade-off: we shift optimization intelligence from the query engine (which would require a geometric algebra-aware cost model) to the schema (where domain expertise can select optimal representations at ingestion). Future native GA engines could internalize these rewrites, but the approach is immediately deployable on standard SQL systems.

7.6.3 5.6.3 Operator Cost Classes

GSQL operators fall into three practical cost classes.

Class A — Cheap predicates (index-friendly): - `lambda_total > tau` - `bivector_norm BETWEEN a AND b` - `lie_jordan_ratio > tau` - generated/derived scalar columns

Class B — Medium-cost algebraic predicates (often KNN-friendly): - `QUAT_DISTANCE(q, q0)< tau` - `DOT(q, q0)> tau` (see 5.6.4) - `SPIN_COMMUTATOR_NORM(omega, omega)> tau` (precomputed column)

Class C — Expensive algebraic transforms (avoid on many rows): - `SANDWICH(M , R)` for full multivector (M) - `GEOM_PRODUCT(M1, M2)` on high-dimensional algebras - on-the-fly `COMMUTATOR(A,B)` if (A,B) are full matrices or dense multivectors

Optimizer rule of thumb: push Class A filters as early as possible, use Class B for candidate pruning, and apply Class C only on a small candidate set.

7.6.4 5.6.4 Rewrite Rules: Turning Geometry into Scalars

A key performance trick is to rewrite expensive geometric predicates into cheaper scalar predicates using identities that preserve correctness.

Example 1 — Quaternion distance threshold as dot-product threshold.
For unit quaternions (q, q_0), the SO(3) geodesic distance used in this paper is:

$$d(q, q_0) = 2 \arccos(|q \cdot q_0|)$$

Then:

$$d(q, q_0) < \tau \quad \iff \quad |q \cdot q_0| > \cos(\tau/2)$$

So an optimizer can rewrite:

```
WHERE QUAT_DISTANCE(orientation_quat, :q0) < :tau
```

into:

```
WHERE ABS(QUAT_DOT(orientation_quat, :q0)) > COS(:tau/2)
```

This rewrite is important because dot products are far cheaper than `ACOS`, and they are more indexable (e.g., via approximate nearest-neighbor structures).

Example 2 — Spin-spin commutator norm.
For 3D skew tensors ($W_1 = widehatomega$) and ($W_2 = widehatnu$):

$$\|[W_1, W_2]\|_F \;=\; \sqrt{2}\,\|\omega \times \nu\|$$

Thus a commutator-norm predicate can be rewritten in terms of cross products (or in-schema `spin_cross_norm`), avoiding any matrix commutator evaluation at query time.

7.6.5 5.6.5 Indexing Geometric Predicates

Scalar columns: ordinary Btree indices.

Quaternion similarity: - *Exact*: requires custom operator classes; most systems lack native RP^3 indexing. - *Practical today*: two-stage approach: 1. Candidate retrieval via a fast approximate index on ((w,x,y,z)) (or on a 3vector log map), then 2. Exact re-ranking using `QUAT_DISTANCE()`.

Bivector similarity (3D generators): - treat as a 3vector in $\mathbb{R}^3$ with an ordinary vector index if using approximate nearest-neighbor, - or rely on Btree over `bivector_norm` for threshold queries.

Sparse high-grade storage: `multivector_sparse(fact_id, grade, blade_index)` can be indexed by `(grade, blade_index)` for fast "find all rows with blade coefficient in range" queries.

7.6.6 5.6.6 Statistics and Selectivity for Arrays

When arrays store coefficient vectors, selectivity estimation becomes difficult. A prototype optimizer can improve estimates by:

1. Maintaining generated scalar features (norms, ratios, top-k coefficient magnitudes).
2. Using extended statistics on generated columns rather than raw arrays.
3. Sampling-based estimation for expensive operators (e.g., geodesic thresholds).

This aligns with the general DB principle: **make expensive predicates decomposable into cheap, estimable predicates.**

7.6.7 5.6.7 Optimizer "Representation Switching" (Key Capability)

A central design goal for QMS is that the same phenomenon can be stored and queried in multiple representations:

- rotor/quaternion (finite orientation)
- generator/bivector (infinitesimal spin)
- full multivector (general-purpose, heavy)

A minimal optimizer can choose a representation by rules such as:

- If a query touches only rotations (SO(3)), prefer quaternions.
- If a query touches only commutator magnitudes of skew parts, prefer generator/cross-product form.
- If the query uses reflections/projections requiring mixed grades, fall back to full multivector.

This is the schema analogue of "vectorize where possible" in numerical computing.

7.7 5.7 Two-Stage and Approximate Geometric Querying

Because full geometric operations can be expensive, many practical GSQL queries should be executed in two stages:

1. **Coarse filter:** use cheap invariants and/or approximate indices to produce a candidate set.
2. **Exact refinement:** apply correct geometric operations only to candidates.

7.7.1 5.7.1 Example: Nearest Orientation Frames

```
-- Stage 1: approximate candidate set (fast)
WITH candidates AS (
  SELECT fact_id, orientation_quat
  FROM multivector_fact
  WHERE domain = 'cfd'
  ORDER BY L2_APPROX_QUAT(orientation_quat, :q0)
  LIMIT 200
)
-- Stage 2: exact re-ranking (correct)
SELECT fact_id,
       QUAT_DISTANCE(orientation_quat, :q0) AS d
FROM candidates
ORDER BY d
LIMIT 10;
```

7.7.2 5.7.2 Example: "Find anomalies in bivector structure"

```
-- Stage 1: cheap threshold on generated norm
WITH candidates AS (
  SELECT fact_id, bivector_coeffs
  FROM multivector_fact
  WHERE bivector_norm > :tau
  LIMIT 100000
)
-- Stage 2: exact geometric predicate on coefficients
SELECT fact_id
FROM candidates
WHERE BIVECTOR_PATTERN_MATCH(bivector_coeffs, :pattern);
```

This design supports high throughput while preserving mathematical correctness in the refinement stage.

8 6. Applications and Use Cases

8.1 6.1 Power Grid Stability Monitoring (Λ_G)

The problem: Monitor sensitivity conflicts in power flow Jacobian before cascade failure.

Multivector storage: - Scalar: Total power, loading percentage - Vector: Voltage gradient across system - Bivector: **Commutator [J, J]** — sensitivity conflict metric - Symmetric: Jacobian eigenvalue distribution

Schema usage:

```
INSERT INTO multivector_fact (
    domain, timestamp, location_key,
    scalar_magnitude, -- Total load (MW)
    lambda_total, -- ||[J, J]||_F
    lambda_skew, -- Skew component
    alarm_status
) VALUES (
    'grid', NOW(), bus_id,
    total_load,
    jacobian_commutator_norm,
    skew_component,
    CASE WHEN jacobian_commutator_norm > 27.85 THEN 'warning' ELSE 'normal' END
);
```

Validated results (IEEE 118-bus): - 151s lead time for cascade detection - +13.8s advantage via shock channel - >6σ sensitivity for localized stress

8.2 6.2 Tokamak Plasma Edge Diagnostics (Λ_Δt)

The problem: Predict pedestal collapse (ELMs) from zonal flow breakdown.

Multivector storage: - Scalar: Pressure, temperature - Vector: E×B flow velocity - Bivector: **Commutator [A(t), A(t-Δt)]** — zonal flow health - Generator: Flow shear rate (3 floats)

Multi-scale query:

```
-- Multi-scale commutator ladder
SELECT
    timestamp,
    location_key,
    lambda_dt_1, -- $\Delta$t = 1$\mu$s
    lambda_dt_2, -- $\Delta$t = 2$\mu$s
    lambda_dt_4, -- $\Delta$t = 4$\mu$s
    lambda_dt_8, -- $\Delta$t = 8$\mu$s
    -- Scaling exponent
    REGR_SLOPE(LOG(lambda_dt), LOG(dt)) OVER (
        PARTITION BY location_key
        ORDER BY timestamp
        ROWS BETWEEN 10 PRECEDING AND CURRENT ROW
    ) AS beta_lambda
FROM tokamak_diagnostics
WHERE beta_lambda > critical_threshold;
```

8.3 6.3 CFD Adaptive Mesh Refinement (Λ_L)

The problem: Predict turbulent structure formation for pre-emptive mesh adaptation.

Multivector storage: - Scalar: Pressure, density - Vector: Velocity u - Bivector: Vorticity ω (stored as generator, 3 floats) - **Commutator split:** - λ_skew: Spin-spin → **cross product** $\omega \times \omega$ - λ_sym: Strain-spin coupling

Optimized computation:

```
-- Exploit cross-product simplification
UPDATE multivector_fact
SET
    -- Spin-spin part: O(6 mults) instead of O(54)
    lambda_spin_spin = sqrt(2) * NORM(CROSS(angular_velocity, angular_velocity_dot)),
    -- Full commutator (includes strain terms)
    lambda_total = compute_full_lambda(velocity_gradient, velocity_gradient_dot)
WHERE domain = 'cfd';
```

Predictive capability: 10-30 timesteps before structure formation.

8.4 6.4 Quantum Error Correction Validation (getQore)

The problem: Validate QEC decoder performance without full state tomography.

Multivector storage: - Scalar: Error rate, fidelity - Vector: Syndrome pattern - Bivector/Rotor: Error rotation in Bloch space - **Validated against Google Willow:** $R^2 > 0.999$

Rotor-native queries:

```
-- Find error syndromes with anomalous rotation patterns
SELECT
    syndrome_id,
    QUAT_DISTANCE(error_rotor, identity_quat) AS rotation_magnitude,
    error_rate
FROM qec_diagnostics
WHERE QUAT_DISTANCE(error_rotor, identity_quat) > threshold
  AND error_rate > expected_rate * 1.5;
```

8.5 6.5 Automotive Sensor Diagnostics (Bonus Application)

The problem: Detect physical misalignment in SRS/IMU sensors via geometric analysis.

Multivector storage: - Vector: Accelerometer readings (3-axis) - Rotor: Sensor orientation quaternion - Bivector: Angular rate (from gyroscope)

Misalignment detection:

```
-- Detect sensor misalignment via orientation drift
WITH orientation_history AS (
    SELECT
        sensor_id,
        timestamp,
        orientation_quat,
        QUAT_DISTANCE(
            orientation_quat,
            LAG(orientation_quat, 100) OVER (PARTITION BY sensor_id ORDER BY timestamp)
        ) AS drift_100
    FROM automotive_sensors
)
SELECT sensor_id, AVG(drift_100) AS avg_drift
FROM orientation_history
GROUP BY sensor_id
HAVING AVG(drift_100) > 0.05 -- Threshold in radians
ORDER BY avg_drift DESC;
```

---

9 7. Implementation Considerations

9.1 7.0 Strategic Decisions for the Final Paper Shape

This paper is intentionally framed as a **schema + query semantics** contribution, not as a claim of a new storage engine. Three choices determine the final narrative and evaluation design.

Flagship evaluation domain (scale): CFD. Computational fluid dynamics telemetry is the most punishing “bigN” stress test ($10^{6-10}9$ rows) and forces the schema to confront real issues: partitioning, indexing, memory bandwidth, and perrow compute overhead. It also showcases the most concrete algebraic optimization in this paper: the **spin–spin commutator norm collapsing to a cross product** (Theorem 2.7), which can be implemented as a cheap generated column.

Secondary evaluation domain (semantic validation): QEC. Quantum error correction provides the cleanest demonstration that the schema is truly *quantumnative*: Bloch vectors, lowweight correlators, and rotor operations are natural objects here, even if the dataset is smaller. In other words: CFD proves the system can scale; QEC proves the semantics are not cosmetic.

Reference platform and portability: The schema is **backendagnostic in design**, but this paper provides a reproducible **PostgreSQL 15+ reference implementation** (arrays + UDFs). This keeps the work concrete while leaving room for future implementations on columnar stores, timeseries databases, or purposebuilt GA engines.

Mixed-precision policy: We adopt a mixed-precision strategy: `float8`/double precision for quantities involving accumulation, ratios, or manifold constraints (quaternions/rotors, Λ/Γ diagnostics, Lie–Jordan ratio), and `float4`/real for bulk telemetry coefficients where memory bandwidth dominates and ~7 digits of precision is sufficient.

9.2 7.1 Storage Backend Options (Reference vs Portability)

The Clifford Fact Table is **backend-agnostic in design**: the algebraic semantics do not depend on any particular database. However, to keep the work concrete and reproducible, we provide a reference implementation targeting **PostgreSQL 15+** using array types, generated columns, and user-defined functions (UDFs). When time-series ingestion dominates (e.g., CFD telemetry), TimescaleDB is a drop-in extension worth noting but not required.

Porting guidelines are straightforward:

- **Coefficient vectors** → array/vector types (preferred), or JSON arrays if necessary
- **Generated invariants** → computed columns or view/materialization strategies
- **Geometric operators** → UDFs in the backend's extension language (SQL, PL/pgSQL, C/C++, Rust, WASM)

Backend family	Pros	Cons	Typical role
PostgreSQL 15+ (reference)	Reproducible; mature SQL + UDF ecosystem	No native GA ops	Existence proof + correctness
TimescaleDB (optional)	Excellent time partitioning + compression	Postgres extension	CFD / sensor telemetry
Columnar analytics (ClickHouse, DuckDB)	Very fast scans/aggregations	UDF surface varies	Large-scale offline analytics
Purpose-built GA engine	Native multivector ops, typed blades	High engineering cost	Research systems / long-term path

9.3 7.2 Index Strategies

For scalar queries (lambda_total, bivector_norm): Standard B-tree

For quaternion similarity: - Custom distance function + GiST index - Or: Store quaternion as 4D vector, use pgvector with caution (wrong metric!)

For multivector similarity: - Grade-specific indices - Or: Dimensionality reduction (PCA on coefficients) + vector index

9.4 7.3 Performance Optimization

9.4.1 7.3.1 Precompute Common Invariants

```
-- Generated columns for common queries
bivector_norm FLOAT GENERATED ALWAYS AS (...) STORED,
lie_jordan_ratio FLOAT GENERATED ALWAYS AS (...) STORED,
spin_cross_norm FLOAT GENERATED ALWAYS AS (...) STORED
```

9.4.2 7.3.2 Materialized Views for Diagnostics

```
CREATE MATERIALIZED VIEW hourly_diagnostics AS
SELECT
    date_trunc('hour', timestamp) AS hour,
    domain,
    AVG(lambda_total) AS avg_lambda,
    MAX(lambda_total) AS max_lambda,
    PERCENTILE_CONT(0.95) WITHIN GROUP (ORDER BY lambda_total) AS p95_lambda,
    COUNT(*) FILTER (WHERE alarm_status = 'warning') AS warning_count
FROM multivector_fact
GROUP BY 1, 2;

-- Refresh strategy
REFRESH MATERIALIZED VIEW CONCURRENTLY hourly_diagnostics;
```

9.4.3 7.3.3 Partitioning

```
-- Partition by domain and time
CREATE TABLE multivector_fact (
    ...
) PARTITION BY LIST (domain);

CREATE TABLE multivector_fact_cfd PARTITION OF multivector_fact
    FOR VALUES IN ('cfd')
    PARTITION BY RANGE (timestamp);

CREATE TABLE multivector_fact_cfd_2026_01 PARTITION OF multivector_fact_cfd
    FOR VALUES FROM ('2026-01-01') TO ('2026-02-01');
```

9.5 7.4 Data Quality and Validation

9.5.1 7.4.1 Constraint Checks

```
-- Quaternion normalization check
ALTER TABLE multivector_fact ADD CONSTRAINT chk_quat_unit
    CHECK (
        ABS(orientation_quat[1]^2 + orientation_quat[2]^2 +
            orientation_quat[3]^2 + orientation_quat[4]^2 - 1.0) < 0.001
        OR orientation_quat IS NULL
    );

-- Non-negative diagnostics
ALTER TABLE multivector_fact ADD CONSTRAINT chk_lambda_positive
    CHECK (lambda_total >= 0 OR lambda_total IS NULL);
```

9.5.2 7.4.2 Geometric Consistency Checks

```
-- Validate: M M should yield consistent scalar (for appropriate M)
-- This is domain-specific; example for rotors:
CREATE FUNCTION validate_rotor_consistency(q FLOAT[4]) RETURNS BOOLEAN AS $$
BEGIN
    -- q q* = 1 for unit quaternions
    RETURN ABS(q[1]^2 + q[2]^2 + q[3]^2 + q[4]^2 - 1.0) < 0.001;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

9.6 7.5 Roadblocks and Mitigation Strategies

Literature review and practical experience identify several barriers to QMS implementation:

9.6.1 7.5.1 Scalability of Multivector Storage

Problem: High-dimensional multivectors grow exponentially—Cl(n,0) has 2 coefficients. For n=10, this is 1024 floats per row; for n=20, over 1 million.

Compounding factor: Full quantum state tomography scales as O(4), rendering direct population impractical for $n > 10$ qubits.

Algebra	Coefficients	Storage (float64)
Cl(3,0)	8	64 bytes
Cl(4,1)	32	256 bytes
Cl(10,0)	1024	8 KB
Cl(20,0)	1,048,576	8 MB

Mitigation strategies: 1. **Sparse storage:** Store only non-zero coefficients with grade/index pairs 2. **Grade-limited population:** Dense storage for grades 0-2, sparse for grades 3+ 3. **Classical shadows:** O(log n) measurements for low-weight observables [7] 4. **Compression via GENEOs:** Group equivariant operators can reduce dimensionality while preserving algebraic structure [21]

```
-- Sparse multivector storage alternative
CREATE TABLE multivector_sparse (
    fact_id BIGINT,
    grade INT,
    blade_index INT,
    coefficient FLOAT,
    PRIMARY KEY (fact_id, grade, blade_index)
);
-- Only stores non-zero entries
```

9.6.2 7.5.2 Interoperability with Existing Systems

Problem: Traditional relational databases lack native GA operations. G-SQL functions require custom implementations that may complicate integration with existing data pipelines.

Specific challenges: - Query optimizers don't understand geometric constraints - ORMs (Django, SQLAlchemy) don't map multivector types - BI tools (Tableau, PowerBI) can't visualize grade structure - ETL tools lack multivector transformations

Mitigation strategies: 1. **PostgreSQL as prototype platform:** Arrays + PL/pgSQL provide sufficient expressiveness 2. **Materialized scalar views:** Export traditional metrics for BI compatibility 3. **REST API layer:** Abstract G-SQL behind standard interfaces 4. **Formal verification:** Reference algebraic database theories where schemas are categories/functors [24] for theoretical grounding

```
-- Compatibility view for traditional BI tools
CREATE VIEW traditional_metrics AS
SELECT
    fact_id,
    timestamp,
    location_key,
    scalar_magnitude,
    lambda_total,
    bivector_norm,
    CASE WHEN lie_jordan_ratio > 2 THEN 'rotation'
         WHEN lie_jordan_ratio < 0.5 THEN 'alignment'
         ELSE 'mixed' END AS regime
FROM multivector_fact;
```

9.6.3 7.5.3 Quantum Hardware Limitations

Problem: Hadamard tests and VQE population assume fault-tolerant quantum hardware. Current NISQ devices suffer from: - Gate errors (1-2% for 2-qubit gates) - Limited qubit counts (< 100 usable) - Short coherence times - Data loading bottlenecks that negate speedups

Reality check from peer review: > "If the quaternions are classical data, you still need to load them into quantum states. Data loading is usually the bottleneck."

Mitigation strategies: 1. **"Future-compatible" framing:** Emphasize near-term classical benefits 2. **Error mitigation:** Zero-noise extrapolation, probabilistic error cancellation 3. **Hybrid workflows:** Quantum for specific subroutines (commutator estimation), classical for bulk storage 4. **NISQ-appropriate applications:** QAOA for sensor placement is viable today; full quantum population is not

Application	Quantum Readiness	Honest Assessment
Schema storage	N/A (classical)	Ready now
Classical analytics	N/A	Ready now
QAOA sensor placement	NISQ	2024-2026
Hadamard commutator	Early fault-tolerant	2028+
Full quantum population	Fault-tolerant	2030+

9.6.4 7.5.4 Generator-Rotor Implementation Overhead

Problem: Maintaining the generator/rotor distinction adds complexity: - Canonicalization triggers for quaternions - Geodesic distance functions instead of built-in Euclidean - SLERP interpolation instead of linear - Double-cover awareness in all quaternion operations

Performance impact: Canonicalization trigger adds ~5-10% overhead on INSERT operations.

Mitigation strategies: 1. **Batch canonicalization:** Process in bulk rather than per-row triggers 2. **Application-layer enforcement:** Move validation to client for high-throughput scenarios 3. **Pre-computed geodesic indices:** Spatial index structures aware of quaternion geometry

9.6.5 7.5.5 Lie-Jordan Decomposition Complexity

Problem: For non-Hermitian operators (common in power systems), computing Lie-Jordan decomposition requires additional care: - [A, B] for non-Hermitian A, B has both real and imaginary parts - Physical interpretation is less direct than for Hermitian cases

Mitigation: Focus on the Frobenius norm [A, B]_F which is always real and well-defined, regardless of Hermiticity.

9.7 7.6 Enhancement Opportunities

Based on literature review, several extensions could strengthen the framework:

9.7.1 7.6.1 Equivariant Learning Integration

Incorporate Lie-algebra equivariant networks [20] for querying multivector data:

```
-- Conceptual: Equivariant aggregation that respects rotational symmetry
SELECT EQUIVARIANT_AGGREGATE(
    bivector_coeffs,
    GROUP_SYMMETRY('SO(3)')
) AS rotation_invariant_summary
FROM multivector_fact
GROUP BY location_key;
```

This would enhance anomaly detection in domains with physical symmetries (e.g., rotational invariance in CFD).

9.7.2 7.6.2 Bivector Ellipse Invariants

Build on GA power system fault analysis [23] to add ellipse-based precomputed columns:

```
-- Ellipse invariants for rapid fault classification
ALTER TABLE multivector_fact ADD COLUMN ellipse_eccentricity FLOAT
    GENERATED ALWAYS AS (compute_ellipse_invariant(bivector_coeffs)) STORED;

ALTER TABLE multivector_fact ADD COLUMN ellipse_orientation FLOAT
    GENERATED ALWAYS AS (compute_ellipse_angle(bivector_coeffs)) STORED;
```

These provide O(1) access to fault signatures currently requiring O(n^2) computation.

9.7.3 7.6.3 Spatiotemporal Extensions via Cl(3,1)

Adapt GIS multivector models [15] for infrastructure with explicit temporal blades:

```
-- Spacetime algebra storage
ALTER TABLE multivector_fact ADD COLUMN spacetime_bivector FLOAT[6];
-- Components: $e_{01}$, $e_{02}$, $e_{03}$ (boosts), $e_{12}$, $e_{23}$, $e_{31}$ (
    rotations)
```

This enables relativistic corrections for: - Propagation delays in continental-scale grids - Relativistic plasma effects in tokamaks - Compressible flow corrections in CFD

9.7.4 7.6.4 Sparse Clifford Compression

Draw from sparse Clifford algebra implementations [18] for high-grade optimization:

```
-- Hybrid dense/sparse storage
CREATE TABLE multivector_hybrid (
    fact_id BIGINT PRIMARY KEY,
    -- Dense storage for low grades
    grade_0_1_2 FLOAT[10], -- Fixed size for grades 0-2
    -- Sparse storage for high grades
    high_grade_indices INT[],
    high_grade_values FLOAT[],
    -- Compression metadata
    sparsity_ratio FLOAT GENERATED ALWAYS AS (
        array_length(high_grade_values, 1)::FLOAT / expected_high_grade_count
    ) STORED
);
```

9.8 7.7 Evaluation and Benchmarking Plan

This paper prioritizes **CFD as the flagship benchmark domain** (scale and performance) and uses **QEC as a secondary semantic validation domain** (quantum-native correctness). Accordingly, the benchmark harness should include a large CFD workload (≥10ˆ6 rows in the reference runs, scaling upward as hardware permits) and a smaller QEC workload focused on rotor/geodesic distance correctness and low-weight correlator queries.

This section provides a concrete benchmarking methodology so the Clifford Fact Table can be evaluated against traditional schemas without relying on claims that cannot be reproduced.

9.8.1 7.7.1 Evaluation Questions

1. **Storage efficiency:** What is the byte cost per row for common domain representations (CFD, grid, QEC)?
2. **Ingest throughput:** How many rows/sec can be inserted with canonicalization + generated invariants enabled?
3. **Query latency and scalability:** How do representative diagnostic queries scale with table size and partitioning?
4. **Correctness of similarity search:** Do quaternion nearest-neighbor results match true SO(3) geodesic distance rankings?
5. **Operator overhead:** How costly are geometric UDFs relative to scalar predicates?

9.8.2 7.7.2 Baseline Schemas

Benchmark against three baselines:

Baseline A — Flat relational columns (traditional): - separate scalar columns for each component - no constraints, no manifold-aware distance

Baseline B — Matrix-heavy "physics table": - store 3×3 tensors as 9 columns (or arrays) - compute commutators and norms at query time

Baseline C — JSONB/document storage: - store multivector coefficients in a document field - compute everything in application layer (worst-case integration test)

9.8.3 7.7.3 Workloads (Representative Queries)

Workload	Query Type	Example
W1	Scalar threshold	"Find λ_total > p95 over last hour"
W2	Bivector norm scan	"Find bivector_norm > τ"
W3	Quaternion KNN	"Find nearest orientation frames (geodesic)"
W4	Regime classification	"Lie/Jordan ratio buckets by location"
W5	Cross-product commutator	"Rank by 2
W6	Rotation magnitude threshold	"Find rotors with d(q, I) > τ"
W7	Full multivector transform	"Apply rotor to selected rows"

9.8.4 7.7.4 Metrics

- **Storage:** bytes/row, index size, compression ratio (if enabled)
- **Latency:** p50/p95/p99 response time
- **Throughput:** rows/sec ingestion and refresh for materialized views
- **CPU:** total CPU time per query
- **Correctness:** KNN recall@K vs exact geodesic ranking (for quaternion workloads)

9.8.5 7.7.5 Analytical Baseline Estimates (Non-Empirical, but Useful)

Even before running empirical tests, several comparisons can be stated without speculation:

1. **Full Cl(3,0) multivector storage is fixed-width:** 8 coefficients (32 bytes as float32/`real`, or 64 bytes as float64/`double precision`) for the full algebra.
2. **Quaternions reduce rotation storage vs 3×3 matrices:** 4 vs 9 floats, and normalization is O(1).
3. **Spin-spin commutator reduces from matrix commutator to cross product:** replacing O(9×9) style operations with O(3) operations (Theorem 2.7).

These are algebraic/representational facts; measured performance will additionally depend on memory bandwidth, indexing, and implementation details.

9.8.6 7.7.6 Benchmark Harness (Reproducible Structure)

A minimal harness should include:

- synthetic data generators for each domain (`cfd`, `qec`, `grid`)
- schema deployment scripts
- workload query suite with parameter sweeps
- results capture (CSV/JSON) + plotting notebooks

See Appendix D for a suggested repository layout.

9.9 7.8 Reference Implementation Outline

To enable reproducible evaluation, the project should ship a minimal reference implementation, even if only as a prototype.

9.9.1 7.8.1 Deliverables

1. **Schema DDL** (`schema.sql`)
2. **Function library** (`gsql_functions.sql`)
3. **Data generators** (Python scripts for synthetic CFD/QEC/grid data)
4. **Benchmark suite** (SQL workloads + runner)
5. **Validation tests** (unit quaternion checks, invariants, regression tests)

9.9.2 7.8.2 Minimal "Day 1 Prototype" Feature Set

- Insert rows into `multivector_fact` with:
  - canonicalized quaternions
  - generated norms (bivector_norm, spin_cross_norm)
- Run:
  - W1 scalar anomaly query
  - W3 quaternion KNN query with exact re-ranking
  - W5 cross-product commutator ranking query

This set is sufficient to validate the core claim: that geometry-aware storage enables geometry-aware querying without reconstructing structure post-hoc.

9.10 7.9 Empirical Results

This section reports empirical validation results produced by the reference PostgreSQL harness described in 7.7.6 and Appendix D. The suite includes:

- **Algebraic identity tests** (Theorem 2.7 / Corollary 2.8)
- **Quaternion metric correctness tests** (geodesic distance, double-cover invariance)
- **System microbenchmarks** (ingest throughput and representative query latency)

Reproducibility note. The harness records the exact database version, key configuration settings, dataset random seeds, and the SQL workload suite used for each run. Readers should treat the specific throughput numbers below as *implementation-dependent* (hardware, WAL settings, indexing strategy), but the qualitative findings (e.g., Euclidean distance breaking double-cover invariance) are invariant.

9.10.1 7.9.0 Experimental Setup

Hardware. - CPU: AMD Ryzen 9 5900X 12-Core Processor - RAM: 64 GB - OS: Windows 11 Pro (Build 26100) - Storage: Local disk (implementation-dependent)

Software. - PostgreSQL 16.10 (compiled with Visual C++ build 1944, 64-bit) - Python 3.11.9 - psycopg2-binary 2.9.11 - numpy 1.20.0+

PostgreSQL configuration (key settings). - `shared_buffers`: 128 MB - `fsync`: on - `synchronous_commit`: on - `wal_level`: replica - `work_mem`: 4 MB - `maintenance_work_mem`: 64 MB - `max_wal_size`: 1 GB - `checkpoint_completion_target`: 0.9

Ingestion method. - Batched `INSERT` using `psycopg2.extras.execute_values` - Batch size: 10,000 rows (CFD), 1,000 rows (QEC) - Generated columns (`spin_cross_norm`, `bivector_norm`, `lie_jordan_ratio`) computed at insert time - Quaternion canonicalization applied in Python before insert (no triggers) - Indexes present during ingest (created in schema deployment)

Measurement protocol. - Cache state: Cold (database restarted before each run for micro-scale validation) - Repetitions: Single run per workload (sufficient for micro-scale correctness validation) - Query timing: planning + execution time reported by `EXPLAIN ANALYZE` (single run per workload) - Ingest timing: Wall-clock time from first batch to last commit

Baseline schemas (for comparison). - **Baseline A (flat):** Traditional schema with scalar columns (`omega_x`, `omega_y`, `omega_z`, `quat_w`, `quat_x`, `quat_y`, `quat_z`, etc.) - **Baseline B (all-double):** Same structure as Baseline A but all numeric columns use `DOUBLE PRECISION` - **Baseline C:** Not measured in micro-scale run (intended for larger N comparisons)

9.10.2 7.9.1 CFD Ingest and Storage Validation (N = 10,000)

Dataset. 10,000 synthetic CFD telemetry rows with angular velocity generators (ω), orientation quaternions (q), and computed Lie/Jordan diagnostics (Λ, Γ, ρ). This "micro-scale" run is intended to validate correctness and system overheads; 7.9.2 reports a scale-up run at N = 1,000,000.

Ingest throughput.

Configuration	Rows/sec	Notes
QMS (CFD row payload)	9,339	Includes generated columns (`spin_cross_norm`, `bivector_norm`, `lie_jordan_ratio`)

Storage footprint (mixed precision). Table 7.9.1 summarizes the *logical* per-element sizes implied by the mixed-precision policy. Note that PostgreSQL array values include additional headers/alignment; physical on-disk row size should be measured with `pg_column_size(...)` and `pg_total_relation_size(...)` for a given deployment.

Column Type	Precision	Element Size	Example Columns
Bulk telemetry coefficients	`REAL` (float4)	4 bytes	`angular_velocity[3]`, `bivector_xy/yz/zx`
Rotors / invariants	`DOUBLE PRECISION` (float8)	8 bytes	`orientation_quat[4]`, `lambda_total`, `spin_cross_norm`
Generated invariants	`DOUBLE PRECISION` (float8)	8 bytes	`bivector_norm`, `lie_jordan_ratio`

Physical storage measurement (10,000 rows). Measured via `pg_total_relation_size()` and `pg_relation_size()`:

Schema	Total Size	Table Size	Index Size	Table Bytes/Row
QMS (multi-vector_fact)	4.7 MB	2.9 MB	1.8 MB	277.0

Schema	Total Size	Table Size	Index Size	Table Bytes/Row
Baseline A (flat, mixed precision)	1.6 MB	1.4 MB	0.2 MB	141.7
Baseline B (flat, all float8)	2.3 MB	2.1 MB	0.2 MB	209.7

Table bytes/row computed as `pg_relation_size(table)/ row_count` (index storage excluded).

Interpretation. Mixed precision reduces bandwidth and storage for bulk coefficients while preserving accuracy for manifold-constrained values (unit quaternions) and accumulated diagnostics (Λ, Γ, ratios). QMS uses ~1.95× the storage of Baseline A due to: (1) generated columns (precomputed invariants) stored in indices (~1.8 MB), (2) PostgreSQL array overhead (~24 bytes per array column), and (3) additional metadata columns. The trade-off is query-time computation savings: Baseline A would need to compute `spin_cross_norm`, `bivector_norm`, and `lie_jordan_ratio` on-the-fly for equivalent queries.

9.10.3 7.9.2 CFD Scale-Up Benchmark (N = 1,000,000)

Dataset. 1,000,000 synthetic CFD telemetry rows with angular velocity generators (ω), orientation quaternions (q), and computed Lie/Jordan diagnostics (Λ, Γ, ρ). This run uses the same reference harness and schema definitions as 7.9.1, but at a size where index selectivity and precomputed invariants dominate query performance.

End-to-end runtime (single run). ~4.5 minutes total, including data generation, ingest into three schemas (QMS, Baseline A, Baseline B), index creation, query execution (W1/W2/W5/W6), and storage measurement.

Ingest throughput (1,000,000 rows).

Schema	Ingest Time	Rows/sec
QMS (multivector_fact)	107 s	9,346
Baseline A (flat, mixed precision)	72 s	13,889
Baseline B (flat, all float8)	80 s	12,500

Query latency (1,000,000 rows). Table 7.9.2 reports representative `EXPLAIN ANALYZE` execution times for the core geometric workloads. Baseline A computes geometric invariants on-the-fly; QMS uses stored/generated invariants (`bivector_norm`, `spin_cross_norm`, `lie_jordan_ratio`) and indexed predicates where applicable.

Workload	Description	QMS	Baseline A	Speedup
W5	Cross-product commutator ranking ($2\omega\times\omega$)	0.153 ms	343 ms	2,242×
W2	Bivector norm threshold ($M > \tau$)	0.785 ms	397 ms	506×

Workload	Description	QMS	Baseline A	Speedup
W6	Rotation magnitude threshold (d(q, I) > τ)	20 ms	200 ms	10.0×
W1	Scalar threshold / anomaly query (λ_total > τ)	540 ms	1,039 ms	1.92×

Storage at scale (1,000,000 rows). Total size measured via `pg_total_relation_size()`:

Schema	Total Size	Relative to Baseline A
QMS (multivector_fact)	423 MB	2.58×
Baseline A (flat, mixed precision)	164 MB	1.00×

Interpretation (why speedups vary). - **Indexed, precomputed invariants (W2, W5)** show the largest gains: QMS turns expensive per-row computation into index-friendly predicates over stored scalars. In contrast, Baseline A must evaluate norms/cross-products during query execution, leading to linear scaling in N. - **Threshold queries dominated by aggregation/scan cost (W1)** show smaller but still meaningful speedups. This is consistent with a workload where I/O and aggregation dominate over the per-row arithmetic saved by precomputation. - **Rotor magnitude (W6)** benefits from the optimizer rewrite in 5.6.4 and from storing rotors as a single semantic type with canonicalization; the resulting query is simpler and less error-prone than reconstructing metrics over flattened columns.

Scale-up takeaway. At N = 1,000,000 rows, the QMS design trade-off becomes more favorable: QMS uses ~2.6× more storage than a flat baseline, but delivers 500×–2,242× speedups for geometric workloads where invariants can be precomputed and indexed.

9.10.4 7.9.3 QEC Semantic Validation (N = 1,000)

Dataset. 1,000 synthetic QEC syndrome rows with Bloch vectors, error rotors, and bivector correlators.

Ingest throughput.

Configuration	Rows/sec	Notes
QMS (QEC row payload)	12,193	Smaller row payload than CFD

Representative query latency.

Workload	Description	Execution Time	Notes
W6	Rotation magnitude threshold	1.325 ms	146 matches from 1,000 QEC rows

This query leverages an indexed `domain` predicate and computes geodesic distance at query time. For larger datasets, a precomputed "rotation-from-identity" generated column (or the optimizer rewrite in 5.6.4) can reduce per-row trigonometric overhead.

Critical semantic validation: geodesic vs. Euclidean distance. The most important semantic claim in this paper is that Euclidean distance in $\mathbb{R}^4$ is the *wrong metric* for quaternion/rotation similarity unless one explicitly enforces the SU(2)→SO(3) double-cover invariance.

Empirical test. For a unit quaternion q representing a rotation, the antipodal quaternion -q represents the *same* rotation. A correct distance should return 0; a nave Euclidean metric returns a maximum value.

Quaternion	Geodesic Distance	Euclidean Distance
q	0.000000	0.000000
-q (same rotation)	**0.000000 ✓**	**2.000000 ✗**

Interpretation. - Geodesic distance correctly identifies q and -q as the same rotation (distance = 0). - Euclidean distance incorrectly treats them as maximally different (distance = 2 for unit quaternions).

This validates the claim that schemas storing quaternions as four independent scalars and relying on Euclidean KNN can produce incorrect nearest-neighbor rankings for rotation similarity.

Semantic query example (rotor anomaly).

```sql
-- Find syndromes with anomalous rotation magnitude (error rotor far from identity)
SELECT
    fact_id,
    quat_geodesic_distance(orientation_quat, ARRAY[1,0,0,0]) AS rotation_from_identity,
    scalar_magnitude AS error_rate,
    alarm_status
FROM multivector_fact
WHERE domain = 'qec'
  AND quat_geodesic_distance(orientation_quat, ARRAY[1,0,0,0]) > 0.1
ORDER BY rotation_from_identity DESC
LIMIT 50;
```

In the QMS schema, the geodesic distance function operates directly on the stored rotor. In a traditional schema with flattened columns (`quat_w`, `quat_x`, `quat_y`, `quat_z`), the query typically requires reconstructing the quaternion, implementing the correct metric manually (including the absolute dot product), and ensuring sign-invariance.

9.10.5 7.9.4 Algebraic Identity Validation

Theorem 2.7 / Corollary 2.8 (spin-spin commutator). The identity
$lVert[$
$widehatomega,$
$widehatnu]$
$rVert_F =$
$sqrt2$
$,$
$lVert$
$omega$
$times$

nu
rVert was validated numerically using 1,000 random vector pairs.

- **Max absolute error:** < 1e-10 (float64)

Quaternion canonicalization tests.

Test Case	Result
Positive w unchanged	PASS
Negative w flipped to positive	PASS
q and -q canonicalize to same	PASS

9.10.6 7.9.5 Summary of Validations

Claim	Validation Method	Result
Cross-product commutator identity (Theorem 2.7)	1,000 random vector pairs	max error < 1e-10
Geodesic distance correctness	Known rotation angles	Matches expected values
Double-cover invariance	d(q, -q) = 0	**Validated**
Euclidean distance is sign-variant	d_euc(q, -q) $\neq$ 0	**Confirmed: returns 2.0**
Canonicalization correctness	Sign flip tests	All PASS
Ingest throughput	Synthetic CFD/QEC payloads	9,339–12,193 rows/sec

Remaining empirical work. To strengthen the evaluation for larger-scale submissions: 1. **Repeatability:** Run the N = 1,000,000 benchmark $\geq$3 times (cold and warm cache variants) and report median and variability for ingest and query latency. 2. **Baseline completeness:** Report Baseline B query latencies and the full table/index size breakdown at N = 1,000,000 (QMS vs Baseline A already measured), and optionally fill Baseline A/B ingest throughput at N = 10,000 for micro-scale comparability. 3. **Workload coverage:** Add W3 (quaternion KNN with recall@K against exact geodesic ranking) and W7 (full multivector transform) at N = 1,000,000 to cover both metric and transform-style workloads.

Conclusion. The empirical suite validates both the *algebraic correctness* of the core identities and the *semantic correctness* of rotor similarity queries. The most practically important finding is that a nave Euclidean metric on quaternion components violates rotation equivalence (q $\equiv$ -q), while the geodesic metric respects it.

10 8. Conclusion

10.1 8.1 Summary of Contributions

We have presented the **Clifford Fact Table**, a quantum-state-aware schema that moves data architecture from "geometry-blind" scalar storage to "geometry-native" algebraic structure. Key contributions:

1. **Grade-structured storage:** Scalars, vectors, bivectors, and higher grades as distinct semantic types with appropriate operations
2. **Lie-Jordan decomposition:** Separation of antisymmetric (incompatibility) and symmetric (co-alignment) relationships—both are physically meaningful but fundamentally different

3. **Generator-rotor distinction:** Clear separation of infinitesimal transformations (3 floats for SO(3)) from finite transformations (4 floats for unit quaternions)
4. **Computational optimizations:** Spin-spin commutator reduces to cross product; geodesic distance for quaternion queries; SLERP for interpolation
5. **Practical implementation:** SQL schema, constraint checking, indexing strategies, and Geometric SQL query language
6. **Cross-domain unification:** Same schema structure applies to power grids, fusion plasmas, CFD, and quantum computing—demonstrating that geometric data architecture is fundamental, not domain-specific
7. **Empirical validation harness:** A reproducible PostgreSQL harness validates the core algebraic identities (e.g., the spin-spin commutator cross-product simplification), quaternion canonicalization, and rotor-distance semantics. It reports microbenchmarks (CFD N=10,000; QEC N=1,000) including physical storage overhead vs flat baselines, and a CFD scale-up benchmark (N=1,000,000) showing that indexed geometric workloads can achieve 500×–2,242× lower latency with QMS at the cost of ~2.6× higher storage footprint than a mixed-precision flat baseline (implementation-dependent).

10.2 8.2 Novelty and Related Work

A comprehensive literature review confirms that while geometric algebra appears in: - GIS and spatial database systems (grade-structured storage) - Cryptographic applications (multivector encoding) - Power system fault analysis (bivector diagnostics) - Equivariant neural networks (Lie algebra processing)

No prior work combines multivector storage, quantum-native population, Lie-Jordan decomposition, and explicit SQL schemas in a unified architecture. The Clifford Fact Table fills this gap.

10.3 8.3 Acknowledged Limitations

We have identified and addressed key roadblocks:

Challenge	Mitigation
Exponential storage (2 coefficients)	Sparse high-grade, dense low-grade
Tomography scaling (O(4))	Classical shadows for population
Interoperability (non-standard SQL)	Compatibility views, REST API layer
Quantum hardware (NISQ limitations)	“Future-compatible” framing
Generator-rotor overhead	Batch canonicalization, application-layer validation

These limitations are real but manageable with the strategies presented.

10.4 8.4 From Data Processing to Geometric Insight

The Clifford Fact Table represents a paradigm shift:

Traditional Approach	Geometric Approach
Store scalars	Store multivectors

Traditional Approach	Geometric Approach
Compute relationships	Store relationships
Euclidean distance	Geodesic/algebraic distance
Post-hoc rotation	Native rotor operations
Domain-specific schema	Unified algebraic schema

The core insight: Geometry is not decoration applied after analysis—it is the fundamental structure that data should preserve from collection through storage to query.

10.5 8.5 Future Directions

1. **Native database engine:** Purpose-built storage engine with GA operations as primitives
2. **Equivariant query optimization:** Cost-based optimizer aware of geometric symmetries [20]
3. **Quantum interface:** Direct population from quantum hardware via classical shadows [7]
4. **Ellipse invariants:** Precomputed bivector invariants for rapid fault classification [23]
5. **Spatiotemporal extensions:** Cl(3,1) for relativistic corrections in infrastructure monitoring
6. **Sparse compression:** GENEOs for high-grade dimensionality reduction [21]

10.6 8.6 The Bridge Built

This work bridges: - **Abstract algebra** (Clifford, Lie, Jordan) **Practical engineering** (SQL, indices, constraints) - **Quantum information** (density matrices, Pauli strings) **Classical dynamics** (velocity gradients, Jacobians) - **Theoretical physics** (Dirac, spacetime algebra) **Data engineering** (schemas, queries, optimization) - **Academic precedent** (GIS, cryptography, equivariant networks) **Novel unification** (this work)

The Clifford Fact Table is not merely a database schema—it is a statement that **mathematical structure deserves first-class citizenship in data infrastructure**.

---

11 Acknowledgments

The author thanks the anonymous peer reviewers whose rigorous feedback on generator vs rotor distinctions, Lie-Jordan decomposition, cross-product simplifications, and quaternion database gotchas substantially improved the mathematical rigor of this work. Additional thanks for the comprehensive literature review that confirmed novelty while identifying valuable related work in GIS, cryptography, and equivariant learning that strengthened the positioning of this contribution.

---

12 References

[1] Grassmann, H. *Die Lineale Ausdehnungslehre* (1844).

[2] Clifford, W.K. "Applications of Grassmann's Extensive Algebra." American Journal of Mathematics 1(4), 350-358 (1878).

[3] Dirac, P.A.M. "The Quantum Theory of the Electron." Proc. Roy. Soc. A 117, 610 (1928).

[4] Hestenes, D. *Space-Time Algebra.* Gordon and Breach (1966).

[5] Hestenes, D. and Sobczyk, G. *Clifford Algebra to Geometric Calculus.* Reidel (1984).

[6] Doran, C. and Lasenby, A. *Geometric Algebra for Physicists.* Cambridge University Press (2003).

[7] Huang, H.-Y. et al. “Predicting Many Properties of a Quantum System from Very Few Measurements.” Nature Physics 16, 1050-1057 (2020). [Classical shadows]

[8] Kimball, R. and Ross, M. *The Data Warehouse Toolkit.* Wiley (2013). [Traditional schema design]

[9] Shoemake, K. “Animating Rotation with Quaternion Curves.” SIGGRAPH ’85 Proceedings, 245-254 (1985). [SLERP]

[10] Diamond, P.H. et al. “Zonal flows in plasma—a review.” Plasma Phys. Control. Fusion 47, R35 (2005).

[11] Kundur, P. *Power System Stability and Control.* McGraw-Hill (1994).

[12] Dudson, B.D. et al. “BOUT++: A framework for parallel plasma fluid simulations.” Comput. Phys. Commun. 180, 1467 (2009).

[13] Yuan, L. et al. “A Geometric Algebra-Based Framework for Next Generation GIS.” Geomatics and Information Science of Wuhan University (2016). [GA-GIS unification]

[14] Li, L. et al. “A 3D Cadastral Data Model Based on Conformal Geometric Algebra.” ISPRS International Journal of Geo-Information 7(10), 382 (2018). [CGA cadastral]

[15] Yuan, L. et al. “Geometric Algebra for Multidimension-Unified Geographical Information System.” Advances in Applied Clifford Algebras 23(2), 497-518 (2013). [Multi-temporal GIS]

[16] Hitzer, E. and Sangwine, S.J. “Multivector and Multivector Matrix Inverses in Real Clifford Algebras.” Applied Mathematics and Computation 311, 375-389 (2017). [Clifford computation]

[17] Yu, Z. et al. “Clifford Algebra-Based Spatiotemporal Analysis for Regional Soil Organic Matter Change Detection.” Mathematical Geosciences 50(8), 889-911 (2018). [Spatiotemporal GA]

[18] Breuils, S. et al. “New Applications of Clifford’s Geometric Algebra.” Advances in Applied Clifford Algebras 32, 17 (2022).

$Softwareimplementations$

[19] De Keninck, S. “ganja.js: Geometric Algebra for JavaScript.” GitHub repository (2020). [GA software]

[20] Finzi, M. et al. “A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups.” ICML (2021). [Lie-equivariant networks]

[21] Conti, F. et al. “Construction of Group Equivariant Non-Expansive Operators.” arXiv:2104.03443 (2021). [GENEOs]

[22] Larocca, M. et al. “Group-Invariant Quantum Machine Learning.” PRX Quantum 3, 030341 (2022). [Quantum geometric ML]

[23] Stankovic, A. et al. “Geometric Algebra for Power System Fault Analysis.” IEEE Transactions on Power Delivery (2019). [GA power systems]

[24] Spivak, D. “Category Theory for the Sciences.” MIT Press (2014). [Categorical database theory]

[25] Harrow, A.W., Hassidim, A., and Lloyd, S. “Quantum algorithm for linear systems of equations.” Phys. Rev. Lett. 103, 150502 (2009).

$HHLalgorithm$

[26] Farhi, E., Goldstone, J., and Gutmann, S. “A quantum approximate optimization algorithm.” arXiv:1411.4028 (2014). [QAOA]

[27] Nature Scientific Reports. “A universal variational quantum eigensolver for non-Hermitian systems.” (2023). [Quantum eigensolvers]

[28] arXiv:2402.08136. “Early Exploration of Quantum Linear Solvers in Power Systems.” (2024). [Quantum power systems]

---

13 Appendix A: Clifford Algebra Quick Reference

13.1 A.1 Cl(3,0) Basis and Products

Basis	Grade	Square	Geometric Meaning
1	0	1	Scalar
e_1, e_2, e_3	1	+1	Vectors (directions)
e_{12}, e_{23}, e_{31}	2	-1	Bivectors (planes)
e_{123} = I	3	-1	Pseudoscalar (volume)

13.2 A.2 Key Identities

Identity	Formula
Geometric product	ab = a·b + a∧b
Commutator	[a,b] = 2(a∧b) for vectors
Rotor action	v’ = RvR†
Quaternion-bivector map	i e_{23}, j e_{31}, k e_{12}

13.3 A.3 Generator vs Rotor Summary

Property	Generator (ω)	Rotor (q)
Mathematical role	Lie algebra	Lie group
Storage	3 floats	4 floats
Composition	Addition	Multiplication
Constraint	None	Unit norm
Interpolation	Linear	SLERP

---

14 Appendix B: SQL Function Library

```
-- Complete G-SQL function signatures (PostgreSQL)

-- Grade extraction
CREATE FUNCTION GRADE(mv FLOAT[], g INT) RETURNS FLOAT[];
CREATE FUNCTION SCALAR(mv FLOAT[]) RETURNS FLOAT;
CREATE FUNCTION VECTOR(mv FLOAT[]) RETURNS FLOAT[];
CREATE FUNCTION BIVECTOR(mv FLOAT[]) RETURNS FLOAT[];

-- Products
CREATE FUNCTION GEOM_PRODUCT(a FLOAT[], b FLOAT[]) RETURNS FLOAT[];
CREATE FUNCTION INNER_PRODUCT(a FLOAT[], b FLOAT[]) RETURNS FLOAT[];
CREATE FUNCTION WEDGE(a FLOAT[], b FLOAT[]) RETURNS FLOAT[];
CREATE FUNCTION COMMUTATOR(a FLOAT[], b FLOAT[]) RETURNS FLOAT[];
CREATE FUNCTION ANTICOMMUTATOR(a FLOAT[], b FLOAT[]) RETURNS FLOAT[];

-- Rotor operations
CREATE FUNCTION ROTATE(v FLOAT[], q FLOAT[4]) RETURNS FLOAT[];
CREATE FUNCTION SANDWICH(mv FLOAT[], q FLOAT[4]) RETURNS FLOAT[];
CREATE FUNCTION ROTOR_COMPOSE(q1 FLOAT[4], q2 FLOAT[4]) RETURNS FLOAT[4];

-- Quaternion utilities
CREATE FUNCTION QUAT_NORMALIZE(q FLOAT[4]) RETURNS FLOAT[4];
CREATE FUNCTION QUAT_CONJUGATE(q FLOAT[4]) RETURNS FLOAT[4];
CREATE FUNCTION QUAT_DOT(q1 FLOAT[4], q2 FLOAT[4]) RETURNS FLOAT;
CREATE FUNCTION L2_APPROX_QUAT(q1 FLOAT[4], q2 FLOAT[4]) RETURNS FLOAT;
CREATE FUNCTION QUAT_DISTANCE(q1 FLOAT[4], q2 FLOAT[4]) RETURNS FLOAT;
CREATE FUNCTION QUAT_SLERP(q1 FLOAT[4], q2 FLOAT[4], t FLOAT) RETURNS FLOAT[4];

-- Diagnostics
CREATE FUNCTION ARRAY_L2_NORM(v FLOAT[]) RETURNS FLOAT;
CREATE FUNCTION FROBENIUS_NORM(m FLOAT[]) RETURNS FLOAT;
CREATE FUNCTION CROSS_PRODUCT(a FLOAT[3], b FLOAT[3]) RETURNS FLOAT[3];
CREATE FUNCTION SPIN_COMMUTATOR_NORM(omega FLOAT[3], omega_dot FLOAT[3]) RETURNS FLOAT;
```

14.1 B.1 Minimal PostgreSQL Prototype Implementations

The paper's main text focuses on schema design and query semantics. For reproducibility, below is a minimal function set that is sufficient to run most examples in Sections 3 and 5 on PostgreSQL. The functions return `float8`/double precision invariants even when the stored coefficient arrays use `real`/float4, matching the mixed-precision policy in 7.0.

```
-- Utility: L2 norm of a float array
CREATE OR REPLACE FUNCTION array_l2_norm(v FLOAT8[])
RETURNS FLOAT8
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
  SELECT sqrt(SUM(x*x)) FROM unnest(v) AS x;
$$;

-- Overload: accept REAL[] and upcast to float8 for stable invariants
CREATE OR REPLACE FUNCTION array_l2_norm(v REAL[])
RETURNS FLOAT8
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
```

```
  SELECT sqrt(SUM((x::float8)*(x::float8))) FROM unnest(v) AS x;
$$;


-- Quaternion dot product (assumes 1-based array indexing)
CREATE OR REPLACE FUNCTION quat_dot(q1 FLOAT8[4], q2 FLOAT8[4])
RETURNS FLOAT8
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
  SELECT q1[1]*q2[1] + q1[2]*q2[2] + q1[3]*q2[3] + q1[4]*q2[4];
$$;

-- Fast chord distance on RP^3 (double-cover aware).
-- For unit quaternions, this is monotone with the SO(3) geodesic angle.
CREATE OR REPLACE FUNCTION l2_approx_quat(q1 FLOAT8[4], q2 FLOAT8[4])
RETURNS FLOAT8
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
  SELECT sqrt(2.0 - 2.0*abs(quat_dot(q1,q2)));
$$;

-- 3D cross product
CREATE OR REPLACE FUNCTION cross_product(a FLOAT8[3], b FLOAT8[3])
RETURNS FLOAT8[3]
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
  SELECT ARRAY[
    a[2]*b[3] - a[3]*b[2],
    a[3]*b[1] - a[1]*b[3],
    a[1]*b[2] - a[2]*b[1]
  ];
$$;

-- Overload: accept REAL[3] generators and upcast
CREATE OR REPLACE FUNCTION cross_product(a REAL[3], b REAL[3])
RETURNS FLOAT8[3]
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
  SELECT cross_product(a::float8[], b::float8[]);
$$;


-- Spin-spin commutator norm: ||[$\hat{\omega}$, $\hat{\omega}$]||_F = 2 |$\omega$ $\
    times$ $\omega$|
CREATE OR REPLACE FUNCTION spin_commutator_norm(omega FLOAT8[3], omega_dot FLOAT8[3])
RETURNS FLOAT8
```

```
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
  SELECT sqrt(2.0) * array_l2_norm(cross_product(omega, omega_dot));
$$;

-- Overload: accept REAL[3] and return float8
CREATE OR REPLACE FUNCTION spin_commutator_norm(omega REAL[3], omega_dot REAL[3])
RETURNS FLOAT8
LANGUAGE SQL
IMMUTABLE
STRICT
AS $$
  SELECT spin_commutator_norm(omega::float8[], omega_dot::float8[]);
$$;
```

These implementations intentionally assume unit quaternions when used for rotation distances. In production, enforce unit-norm constraints (as in 7.4) and normalize at ingestion time.

---

15 Appendix C: Proof Sketches for Geometric Consistency

This appendix provides compact proofs (or proof sketches) for the most important "geometric correctness" claims used by the schema and GSQL functions.

15.1 C.1 Wedge Product as Half the Commutator (Vectors)

Let (a,b) be vectors in a Clifford algebra. By definition, the geometric product decomposes into symmetric and antisymmetric parts:

$$ab = a \cdot b + a \wedge b, \qquad ba = b \cdot a + b \wedge a$$

The inner product is symmetric ($a \cdot b = b \cdot a$) and the wedge product is antisymmetric ($b \wedge a = -a \wedge b$). Subtract:

$$ab - ba = (a \cdot b - b \cdot a) + (a \wedge b - b \wedge a) = 0 + 2(a \wedge b)$$

Therefore:

$$a \wedge b = \tfrac{1}{2}(ab - ba) = \tfrac{1}{2}[a, b]$$

This justifies interpreting the vector commutator as a bivector quantity.

15.2 C.2 Quaternion Double-Cover and Canonicalization Correctness

Unit quaternions represent rotations via the sandwich action on pure quaternions (vectors):

$$v' = q, v, q\{-1\}$$

For any unit quaternion (q), note that ((-q)^{-1} = -q^{-1}). Then:

$$(-q), v, (-q)\{-1\} = (-q), v, (-q\{-1\}) = q, v, q\{-1\}$$

So (q) and (-q) represent the same rotation in SO(3). Canonicalization rules (e.g., enforce ($w \geq 0$), and tie-break on subsequent components) do not change the represented rotation; they only choose a unique representative for indexing and time-series continuity.

15.3 C.3 Geodesic Distance Between Rotations from Quaternion Dot Product

For unit quaternions (q_1,q_2), define the relative rotation:

$$q_\Delta = q_1\{-1\}q_2$$

Any unit quaternion can be written as $q_\Delta = (\cos(\theta/2), \hat{n}\sin(\theta/2))$ for some axis $\hat{n}$ and angle $\theta \in [0, \pi]$ in SO(3). The scalar part of q_Δ is $\cos(\theta/2)$.

For unit quaternions, the dot product satisfies:

$$q_1 \cdot q_2 = \cos(\theta/2)$$

up to the sign ambiguity ($q \sim -q$). Taking absolute value resolves the double cover, giving:

$$\theta = 2\arccos(|q_1 \cdot q_2|)$$

Thus the distance function in 3.3.2 is the correct SO(3) geodesic angle.

15.4 C.4 Spin-Spin Commutator Equals Cross Product (Hat Map)

Let $\widehat{\omega}$ denote the 3×3 skew-symmetric matrix representing cross product with ω: $\widehat{\omega}x = \omega \times x$. Then:

$$[\widehat{\omega}, \widehat{\nu}]x = \widehat{\omega}(\widehat{\nu}x) - \widehat{\nu}(\widehat{\omega}x) = \omega \times (\nu \times x) - \nu \times (\omega \times x)$$

Using the vector triple product identity:

$$a \times (b \times x) - b \times (a \times x) = (a \times b) \times x$$

we obtain:

$$[\widehat{\omega}, \widehat{\nu}]x = (\omega \times \nu) \times x = \widehat{\omega \times \nu}\, x$$

So:

$$[\widehat{\omega}, \widehat{\nu}] = \widehat{\omega \times \nu}$$

This proves Theorem 2.7. The Frobenius norm relation in Corollary 2.8 follows from the fact that $\|\widehat{\omega}\|_F^2 = 2\|\omega\|^2$, which can be verified by expanding the entries of the skew matrix.

15.5 C.5 Rotor Sandwich Preserves Vector Norm (Unit Rotors)

Let R be a unit rotor ($R\widetilde{R} = 1$), where $\widetilde{R}$ denotes reversion. Define $v' = Rv\widetilde{R}$. Then:

$$(v')^2 = (Rv\widetilde{R})(Rv\widetilde{R}) = Rv(\widetilde{R}R)v\widetilde{R} = Rv(1)v\widetilde{R} = R(v^2)\widetilde{R} = v^2.$$

since (v^2) is a scalar and commutes with all multivectors. Thus rotor actions preserve the quadratic form (Euclidean norm in Cl(3,0)), validating constraint checks and rotor-based querying.

---

16 Appendix D: Reference Implementation Repository Skeleton

This appendix provides a concrete repository layout and minimal components that can serve as a "reference implementation" for the Clifford Fact Table.

16.1 D.1 Suggested Repository Layout

```
clifford-fact-table/
  README.md
  LICENSE
  paper_snippets/
    SECTION_79_FINAL.md
  sql/
    00_schema.sql
    01_functions_quat.sql
    02_functions_gsql.sql
    03_views_compat.sql
  python/
    requirements.txt
    generate_synthetic_cfd.py
    generate_synthetic_qec.py
    generate_synthetic_grid.py
    ingest.py
    validate.py
    gather_experimental_setup.py
  benchmarks/
    workloads.sql
    run_benchmarks.py
    run_baseline_comparison.py
    configs/
      small.yaml
      medium.yaml
      large.yaml
  notebooks/
    analyze_results.ipynb
  tests/
    test_quat_distance.py
    test_spin_commutator.py
    test_grade_projection.py
```

16.2 D.2 Minimal Synthetic Data Models

CFD synthetic row (diagnostics-focused): - `angular_velocity` (3 floats) - `angular_velocity_dot` (3 floats) - `bivector_norm`, `lambda_total` (scalars) - optional `orientation_quat` (4 floats)

QEC synthetic row (qubit/rotor-focused): - `vector_bloch` (3 floats) - `orientation_quat` (4 floats) - `bivector_coeffs` (low-weight correlators)

Grid synthetic row (sensitivity-focused): - `scalar_magnitude` (load) - `lambda_total` (commutator norm proxy) - minimal system identifiers

16.3 D.3 Validation Tests (Must-Have)

1. **Quaternion unit norm** within tolerance
2. **Distance symmetry:** `QUAT_DISTANCE(q1,q2)=QUAT_DISTANCE(q2,q1)`
3. **Double-cover invariance:** `distance(q1,q2)=distance(q1,-q2)`
4. **Spin commutator identity:** compare matrix commutator vs cross-product result
5. **Grade projection sanity:** projections sum back to original multivector (when stored densely)

These tests directly support the "geometric consistency" claims of the paper.

END OF PAPER

---

17 Document Status

Section	Completeness	Notes
Abstract	**95%**	Literature review findings integrated
1. Introduction	**95%**	Related work section added
2. Theoretical Framework	**95%**	Complete with peer review corrections
3. Schema Architecture	**95%**	Full SQL with constraints
4. Quantum Population	**90%**	Honest framing from peer review
5. Geometric SQL	**95%**	Query examples + optimizer cost model
6. Applications	**90%**	Four domains documented
7. Implementation	**98%**	Roadblocks + enhancements + initial empirical results
8. Conclusion	**95%**	Literature + limitations acknowledged
References	**95%**	28 citations from literature review

Section	Completeness	Notes
Appendices	**95%**	Quick reference + SQL library + proofs + repo outline

Overall completion: ~98%

17.1 Key Additions from Literature Review:

1. Section 1.4: Comprehensive related work (GIS, crypto, equivariant networks, power GA)
2. Section 7.5: Roadblocks with concrete mitigation strategies
3. Section 7.6: Enhancement opportunities (ellipse invariants, Cl(3,1), sparse compression)
4. References expanded from 12 to 28 citations
5. Abstract updated to claim novelty with evidence
6. Conclusion acknowledges limitations and positions contribution

17.2 Remaining Work

1. **Baseline A/B ingest + latency comparisons (micro-scale):** At N = 10,000, measure ingest throughput and workloads W1/W2/W5/W6 for Baseline A/B to quantify the overhead/benefit of precomputed invariants. (Physical storage overhead is already reported in 7.9.1.)
2. **Scale-up run (CFD flagship):** Execute the full benchmark suite at $N \geq 10\hat{}6$ CFD rows under realistic partitioning and indexing to report throughput/latency curves and validate bandwidth-limited behavior.
3. **Publish/reference the implementation repository:** Release the reference harness (Appendix D) including workload SQL, synthetic generators, and the setup/measurement scripts used in 7.9.
4. **(Optional) Rotor-similarity indexing:** Implement an indexable approximation (dot-product threshold rewrite + candidate generation + exact geodesic re-ranking) and report recall@K and latency.
5. **(Optional) Full formalization:** Extend Appendix C proof sketches to machine-checked proofs (proof assistant) for geometric consistency claims.