SiQL

Registry: [HOWL-COMP-7-2026]

Series Path: [HOWL-COMP-1-2026] → [HOWL-COMP-2-2026] → [HOWL-COMP-3-2026] → [HOWL-COMP-4-2026] → [HOWL-COMP-5-2026] → [HOWL-COMP-6-2026] → [HOWL-COMP-7-2026]

Parent Framework: [HOWL-COMP-1-2026]

DOI: 10.5281/zenodo.18676988

Date: February 2026

Domain: Software Architecture / Systems Engineering / Real-Time Computing

Status: Architectural Blueprint for Independent Implementation

AI Usage Disclosure: Only the top metadata, figures, refs and final copyright sections were edited by the author. All paper content was LLM-generated using Anthropic's Claude 4.5 Sonnet.

SiQL

SiQL is a SQL-replacement for doing queries with Prolog. Here is a minimal conversion chart:

Details

SQL Feature Completeness Check

SQL Feature	Prolog Native	Example
SELECT	Rule head	`result(Name, Health) :-`
SELECT *	Return entity	`result(Actor) :-`
SELECT DISTINCT	distinct	`distinct`
SELECT AS (alias)	Head variable names	`result(PlayerName, HP) :-`

SQL Feature	Prolog Native	Example
FROM	DR path	`character.name(Actor, Name)`
FROM multiple tables	Multiple paths	`actor(A), faction(F)`
Table alias	Variable	`A` is the alias

SQL Feature	Prolog Native	Example

SQL Feature	Prolog Native	Example
WHERE =	Unification	`faction_id(A, enemy)`
WHERE <, >, <=, >=	Comparison	`Health < 50`
WHERE !=	=	`Faction \= player`
WHERE BETWEEN	Range	`Health >= 20, Health <= 80`
WHERE IN	Member	`member(Faction, [enemy, neutral])`
WHERE NOT IN	+ member	`\+ member(Faction, [player, ally])`
WHERE LIKE	Pattern match	`prefix(Name, "Skeleton")`
WHERE IS NULL	Var check	`var(Value)` or `Value = null`
WHERE IS NOT NULL	Nonvar	`nonvar(Value)`
WHERE AND	Comma	`A, B, C`
WHERE OR	Semicolon	`A ; B`
WHERE NOT	+	`\+ enemy(A)`

SQL Feature	Prolog Native	Example
INNER JOIN	Shared variable	`faction_id(A, F), faction.name(F, Name)`
LEFT JOIN	Optional match	`(faction_id(A, F), faction.name(F, Name) ; Name = null)`
RIGHT JOIN	Flip left join	Same pattern, different order
FULL OUTER JOIN	Union of left joins	Two clauses
CROSS JOIN	No shared var	`actor(A), faction(F)` (all pairs)
SELF JOIN	Different vars	`actor(A), actor(B), A \= B`

SQL Feature	Prolog Native	Example
ORDER BY ASC	sort	`sort(Health, asc)`
ORDER BY DESC	sort	`sort(Health, desc)`
ORDER BY multiple	sort list	`sort([Faction, Health])`
LIMIT	limit	`limit(10)`
OFFSET	offset	`offset(20)`
LIMIT + OFFSET	Both	`limit(10), offset(20)`

SQL Feature	Prolog Native	Example
COUNT(*)	count/2	`count(A, enemy(A), N)`
COUNT(DISTINCT)	count + distinct	`count(A, enemy(A), N) distinct(A)`
SUM	sum/3	`sum(H, character.health.value(A, H), Total)`
AVG	avg/3	`avg(H, character.health.value(A, H), Mean)`
MIN	min/3	`min(H, character.health.value(A, H), Lowest)`
MAX	max/3	`max(H, character.health.value(A, H), Highest)`
GROUP BY	Implicit in aggregation	Grouped by unbound vars
HAVING	Condition after agg	`count(..., N), N > 5`

SQL Feature	Prolog Native	Example
UNION	Multiple clauses	Two rules, same head
UNION ALL	Multiple + all	`all` (keep duplicates)
INTERSECT	Both conditions	`rule1(X), rule2(X)`
EXCEPT	Negation	`rule1(X), \+ rule2(X)`

SQL Feature	Prolog Native	Example

SQL Feature	Prolog Native	Example
Subquery in WHERE	Rule call	`enemy(A) :- evil_faction(A.faction)`
Subquery in FROM	Rule as source	Query a rule result
Correlated subquery	Shared variable	Variable flows into sub-rule
EXISTS	+ +	`\+ \+ enemy(A)` (succeeds if any)
NOT EXISTS	+	`\+ enemy(A)`

SQL Feature	Prolog Native	Example
CASE WHEN	Conditional clauses	Multiple rules with guards
COALESCE	Default	`(Value = X ; X = default)`
NULLIF	Conditional null	`(A = B -> X = null ; X = A)`
CAST	Type conversion	`float(X, Y)` builtin

SQL Feature	Prolog Native	Example
INSERT	assert	`assert(character.health.value(5, 100))`
UPDATE	retract + assert	`retract(old), assert(new)`
DELETE	retract	`retract(character(5, _))`
UPSERT	Conditional	`(exists(X) -> update(X) ; insert(X))`

What SQL Can't Do That Prolog Can

Feature	SQL	Prolog
Recursion	CTE (clunky)	Native
Graph traversal	Multiple self-joins	`ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y)`
Pattern matching	LIKE only	Full unification
Logic composition	Views (static)	Rules (dynamic)

Feature	SQL	Prolog
Backtracking	N/A	Native
Negation as failure	NOT EXISTS (limited)	\+ (general)
Meta-queries	N/A	Query the rules themselves

Recursive Example

SQL (CTE):

```
WITH RECURSIVE ancestors AS (
SELECT parent_id, child_id FROM family WHERE child_id = 5
UNION ALL
SELECT f.parent_id, a.child_id
FROM family f
JOIN ancestors a ON f.child_id = a.parent_id
)
SELECT * FROM ancestors;
```

Prolog Native:

```
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
?- ancestor(A, entity_5).
```

SQL: 8 lines, confusing.

Prolog: 2 lines, obvious.

Graph Query Example

“Find all entities within 3 hops of the player”

SQL: Impossible without known depth or recursive CTE nightmare

Prolog:

```
nearby(X, 0) :- X = player.

nearby(X, N) :- N > 0, N1 is N - 1, nearby(Y, N1), connected(Y, X).

?- nearby(Entity, 3).
```

Yes, Prolog Native can do everything SQL can do.

Plus:

- Recursion (natural)
- Graph traversal (trivial)
- Logic composition (rules call rules)
- Dynamic queries (rules are data)
- Type safety (DR paths validated)
- Auto-complete (schema-driven)

Minus:

- Nothing

The Trade-off That Isn't

Concern	Reality
"Prolog is slow"	Ground term matching = direct comparison
"Prolog is weird"	DR paths make it readable
"SQL is standard"	Your engine, your standard
"Tooling"	You're building the tooling

Final Verdict

SQL is a query language bolted onto relational algebra.

Prolog Native is logic that happens to query data.

SQL can only query.

Prolog queries AND reasons.

Same power for data retrieval. More power for everything else.

Zero compromises. Pure upside.

References

[**HOWL-COMP-7-2026**] SiQL. Github: https://github.com/ghowland/cks/tree/main/papers/COMP/HOWL-COMP-7-2026

[**HOWL-COMP-1-2026**] Implementing a Tall-Infra Data-Only Execution System . Github: https://github.com/ghowland/cks/tree/main/papers/COMP/HOWL-COMP-1-2026

[**HOWL-COMP-2-2026**] Tall-Infra, Data-Only Development. Github: https://github.com/ghowland/cks/tree/main/papers/COMP-2-2026

[**HOWL-COMP-3-2026**] Silo OS. Github: https://github.com/ghowland/cks/tree/main/papers/COMP/HOWL-COMP-3-2026

[**HOWL-COMP-4-2026**] Geometric Security. Github: https://github.com/ghowland/cks/tree/main/papers/COMP/HOWL-COMP-4-2026

[**HOWL-COMP-5-2026**] Partial Geometric Security. Github: https://github.com/ghowland/cks/tree/main/papers/COMP/HOWL-COMP-5-2026

[**HOWL-COMP-6-2026**] SIML - Silo Markup Language (TOML Format). Github: https://github.com/ghowland/cks/tree/main/papers/COMP/HOWL-COMP-6-2026