MySQL Assertion Store for SWI-Prolog

Persistent Storage System for Logic Programming

Version 1.0

October 2025

Abstract

This document describes the MySQL Assertion Store, a comprehensive persistent storage system for SWI-Prolog that combines the declarative power of logic programming with the reliability and scalability of relational databases. The system provides ACID transactions, automatic caching, and secure query execution while maintaining the natural Prolog programming paradigm.

Contents

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Key Features . . . 3

2 Architecture **3**
2.1 System Overview . . . 3
2.2 Core Components . . . 4
2.2.1 mysql_store.pl . . . 4
2.2.2 mysql_store_advanced.pl . . . 4
2.3 Database Schema . . . 4

3 Installation **5**
3.1 Prerequisites . . . 5
3.2 Installing Dependencies . . . 5
3.3 Database Configuration . . . 5
3.3.1 Create Database and User . . . 5
3.3.2 Create Schema Tables . . . 5
3.4 ODBC Configuration . . . 5

4 Usage Guide **6**
4.1 Basic Operations . . . 6
4.1.1 Connecting to Database . . . 6
4.1.2 Managing Contexts . . . 6
4.1.3 Asserting Facts . . . 6
4.1.4 Querying Facts . . . 7
4.1.5 Retracting Facts . . . 7
4.2 Advanced Features . . . 7
4.2.1 Transactions . . . 7
4.2.2 Batch Operations . . . 7
4.2.3 Statistics and Optimization . . . 8

5 Implementation Details **8**
5.1 Hash-Based Deduplication . . . 8
5.2 Caching Strategy . . . 8
5.3 Transaction Management . . . 8
5.4 Security . . . 9
5.4.1 SQL Injection Prevention . . . 9
5.4.2 Access Control . . . 9

6 Performance Optimization **9**
6.1 Indexing Strategy . . . 9
6.1.1 Default Indexing . . . 9
6.1.2 Custom Indexing . . . 9
6.2 Best Practices . . . 9
6.3 Performance Metrics . . . 10

7 Practical Examples **10**
7.1 Example 1: Knowledge Base . . . 10
7.2 Example 2: Persistent Counter . . . 10

7.3 Example 3: Event Log . . . 11

8 API Reference 11
8.1 Core Module Predicates . . . 11
8.1.1 Connection Management . . . 11
8.1.2 Assertion Operations . . . 11
8.1.3 Query Operations . . . 12
8.1.4 Cache Management . . . 12
8.2 Advanced Module Predicates . . . 12
8.2.1 Transactions . . . 12
8.2.2 Batch Operations . . . 12

9 Troubleshooting 13
9.1 Common Issues . . . 13
9.1.1 Connection Problems . . . 13
9.1.2 Schema Mismatch . . . 13
9.1.3 Transaction Issues . . . 13
9.2 Debug Mode . . . 13
9.3 Performance Monitoring . . . 13

10 Limitations and Future Work 14
10.1 Current Limitations . . . 14
10.2 Future Enhancements . . . 14

11 Conclusion 14

A Complete Database Schema 14

B Module Export Lists 16
B.1 mysql_store.pl Exports . . . 16
B.2 mysql_store_advanced.pl Exports . . . 16

1 Introduction

The MySQL Assertion Store bridges the gap between Prolog's in-memory fact base and persistent relational storage. It enables Prolog applications to store and query large datasets while maintaining the declarative programming style that makes Prolog powerful.

1.1 Motivation

Traditional Prolog systems store facts in memory, which limits scalability and lacks persistence across sessions. While file-based persistence exists, it lacks:

- Concurrent access control
- Transaction support
- Query optimization
- Indexing capabilities
- Backup and recovery mechanisms

The MySQL Assertion Store addresses these limitations by leveraging MySQL/MariaDB's robust database engine while preserving Prolog's declarative query interface.

1.2 Key Features

- **Transparent Persistence**: Store Prolog facts with simple assert/retract operations
- **ACID Transactions**: Full transactional support with commit and rollback
- **Automatic Caching**: Intelligent in-memory caching for performance
- **Security**: Parameterized queries prevent SQL injection
- **Scalability**: Handle millions of facts with database optimization
- **Context Management**: Organize facts into separate namespaces
- **Batch Operations**: Efficient bulk insert and delete
- **Statistics Tracking**: Monitor and optimize query patterns

2 Architecture

2.1 System Overview

The system consists of three layers:

1. **Prolog Interface Layer**: Provides natural Prolog predicates for data manipulation
2. **Cache Layer**: Manages in-memory predicate storage for performance
3. **Persistence Layer**: Stores facts in MySQL with full ACID guarantees

```
Prolog Application Code



mysql_store / mysql_store_advanced
(API Layer)



In-Memory Cache
(predicate_cache)



ODBC Interface



MySQL/MariaDB Database
```

Figure 1: System Architecture

2.2 Core Components

2.2.1 mysql_store.pl

The core module provides:

- Connection management and DSN configuration
- Context (namespace) creation and management
- Basic CRUD operations: assert, retract, abolish
- Query operations with automatic caching
- Statistics collection and optimization hints
- Cache management and synchronization

2.2.2 mysql_store_advanced.pl

Advanced features include:

- Transaction support (begin, commit, rollback)
- Batch assertion and retraction operations
- Configurable argument indexing
- Import and export functionality
- Performance optimization tools

2.3 Database Schema

The schema consists of six main tables:

Table	Purpose
contexts	Namespace management
formulae	Storage for Prolog terms
arguments_indexed	Indexed predicate arguments
list_elements	Indexed list elements
predicate_stats	Query statistics
cache_status	Cache metadata

Table 1: Database Tables

3 Installation

3.1 Prerequisites

1. SWI-Prolog 9.0.4 or higher
2. MySQL 5.5+ or MariaDB 10.11+
3. MySQL ODBC Driver 8.0+
4. unixODBC library

3.2 Installing Dependencies

Ubuntu/Debian:

```
sudo apt-get update
sudo apt-get install swi-prolog mysql-server \
    libmyodbc unixodbc
```

Fedora/RHEL:

```
sudo dnf install pl mysql-server \
    mysql-connector-odbc unixodbc
```

3.3 Database Configuration

3.3.1 Create Database and User

```
CREATE DATABASE prolog_store;
CREATE USER 'prolog'@'localhost'
    IDENTIFIED BY 'prolog123';
GRANT ALL PRIVILEGES ON prolog_store.*
    TO 'prolog'@'localhost';
FLUSH PRIVILEGES;
```

3.3.2 Create Schema Tables

Execute the complete schema creation script (see Appendix A for full SQL).

3.4 ODBC Configuration

Configure the Data Source Name (DSN) in `~/.odbc.ini`:

```
[mydb]
Description = MySQL Prolog Store
Driver = MySQL ODBC 8.0 Driver
Server = localhost
Port = 3306
Database = prolog_store
User = prolog
Password = prolog123
```

Verify the driver configuration in `/etc/odbcinst.ini`:

```
[MySQL ODBC 8.0 Driver]
Description = MySQL ODBC 8.0 Driver
Driver = /usr/lib/x86_64-linux-gnu/odbc/libmyodbc8w.so
Setup = /usr/lib/x86_64-linux-gnu/odbc/libmyodbc8S.so
UsageCount = 1
```

Test the connection:

```
isql -v mydb
```

4 Usage Guide

4.1 Basic Operations

4.1.1 Connecting to Database

```
:- use_module(mysql_store).

% Connect to database
?- store_connect(mydb, 'localhost', 'prolog_store',
                 'prolog', 'prolog123').
```

4.1.2 Managing Contexts

Contexts provide namespaces for organizing facts:

```
% Create or ensure context exists
?- store_ensure_context(mydb, my_context).
```

4.1.3 Asserting Facts

```
% Simple fact
?- store_assert(mydb, my_context:person(john, 30)).

% Complex term with nested structures
?- store_assert(mydb, my_context:
    employee(john,
             department(engineering, building_a),
             [skill(prolog), skill(python)])).
```

4.1.4 Querying Facts

```
% Direct query
?- store_call(mydb, my_context:person(Name, Age)).
Name = john,
Age = 30.

% Using findall
?- findall(Name-Age,
           store_call(mydb, my_context:person(Name, Age)),
           Results).
Results = [john-30, jane-25, bob-35].
```

4.1.5 Retracting Facts

```
% Retract first match
?- store_retract(mydb, my_context:person(john, _)).

% Retract all matches
?- store_retractall(mydb, my_context:person(_, Age)),
   Age > 60.

% Remove entire predicate
?- store_abolish(mydb, my_context:person/2).
```

4.2 Advanced Features

4.2.1 Transactions

```
:- use_module(mysql_store_advanced).

% Successful transaction
?- store_transaction(mydb, (
    store_assert(mydb, my_context:account(acc1, 1000)),
    store_assert(mydb, my_context:account(acc2, 500))
)).

% Transaction with conditional rollback
?- store_transaction(mydb, (
    store_assert(mydb, my_context:transfer(acc1, acc2, 200)),
    (valid_transfer(acc1, acc2, 200) ->
        true ;
        throw(invalid_transfer))
)).
```

4.2.2 Batch Operations

```
% Batch insert
?- Terms = [
    my_context:data(1, a),
    my_context:data(2, b),
    my_context:data(3, c)
],
store_assert_batch(mydb, Terms).

% Batch delete
```

```
?- Patterns = [
    my_context:temp(_),
    my_context:cache(_)
],
store_retract_batch(mydb, Patterns).
```

4.2.3 Statistics and Optimization

```
% Get usage statistics
?- store_stats(mydb, my_context, Stats).
Stats = [
    stats(person/2, 150, 50, 10),
    stats(employee/3, 75, 25, 5)
].

% Optimize hot predicates
?- store_optimize(mydb, my_context).
```

5 Implementation Details

5.1 Hash-Based Deduplication

Terms are automatically deduplicated using SHA-256 hashing:

1. Term is converted to canonical form
2. SHA-256 hash is computed
3. Hash is checked against existing formulae
4. If duplicate, only timestamp is updated
5. If new, term is inserted with hash

This ensures logical correctness while allowing efficient duplicate detection.

5.2 Caching Strategy

The system uses **lazy loading**:

- Predicates are loaded on first query
- Subsequent queries use in-memory cache
- Cache is updated on assert/retract
- Cache is cleared on transaction rollback
- Hot predicates are identified and preloaded

5.3 Transaction Management

Transaction support ensures ACID properties:

- **Atomicity**: All operations commit or rollback together
- **Consistency**: Database constraints are enforced
- **Isolation**: Transactions are isolated (MySQL isolation level)
- **Durability**: Committed data persists across failures

Implementation uses:

1. Disable autocommit before transaction
2. Execute operations within transaction
3. Clear cache on rollback to maintain consistency
4. Re-enable autocommit after commit/rollback

5.4 Security

5.4.1 SQL Injection Prevention

All database queries use **parameterized statements**:

```
% Secure query with parameter binding
odbc_prepare(Conn,
    'SELECT * FROM formulae WHERE context_id = ?',
    [default],
    Stmt),
odbc_execute(Stmt, [ContextId], Results).
```

User input is never concatenated into SQL strings.

5.4.2 Access Control

Database-level access control through MySQL user permissions:

- Dedicated database user with limited privileges
- No shell access or FILE privileges
- Connection restricted to localhost or specific hosts

6 Performance Optimization

6.1 Indexing Strategy

6.1.1 Default Indexing

By default, the first argument of each predicate is indexed:

```
% person(Name, Age) - Name is indexed
% employee(Id, Dept, Salary) - Id is indexed
```

6.1.2 Custom Indexing

Configure indexing for specific predicates:

```
% Index arguments 1 and 3 of employee/3
?- store_configure_index(my_context, employee, 3, [1, 3]).

% Rebuild indices
?- store_rebuild_indices(mydb, my_context).
```

6.2 Best Practices

1. **Use Transactions**: Batch multiple operations
2. **Monitor Statistics**: Identify query patterns with `store_stats/3`
3. **Configure Indexing**: Index frequently queried arguments

4. **Batch Operations**: Use `store_assert_batch/2` for bulk inserts
5. **Context Organization**: Separate unrelated data
6. **Cache Management**: Unload rarely used predicates
7. **Regular Optimization**: Run `store_optimize/2` periodically

6.3 Performance Metrics

Typical performance characteristics (Intel i7, SSD, local MySQL):

Operation	Performance
Single assert	2-5 ms
Batch assert (100 facts)	20-50 ms
Cached query	0.1-0.5 ms
Uncached query	5-15 ms
Transaction commit (10 ops)	10-25 ms

Table 2: Performance Benchmarks

7 Practical Examples

7.1 Example 1: Knowledge Base

```
% Store a family tree
?- store_assert(mydb, kb:parent(tom, bob)).
?- store_assert(mydb, kb:parent(tom, alice)).
?- store_assert(mydb, kb:parent(bob, charlie)).

% Define rules (in Prolog, not stored)
grandparent(X, Z) :-
    store_call(mydb, kb:parent(X, Y)),
    store_call(mydb, kb:parent(Y, Z)).

% Query
?- grandparent(tom, charlie).
true.

% Find all grandchildren
?- findall(G, grandparent(tom, G), Grandchildren).
Grandchildren = [charlie].
```

7.2 Example 2: Persistent Counter

```
% Initialize counter
?- store_assert(mydb, app:counter(0)).

% Atomic increment
increment_counter(N) :-
    store_transaction(mydb, (
        store_call(mydb, app:counter(Old)),
        N is Old + 1,
        store_retract(mydb, app:counter(Old)),
        store_assert(mydb, app:counter(N))
    )).

% Use counter
?- increment_counter(N1).
N1 = 1.

?- increment_counter(N2).
N2 = 2.
```

7.3 Example 3: Event Log

```
% Log events with timestamp
log_event(Type, User, Data) :-
    get_time(Timestamp),
    store_assert(mydb,
        audit:event(Type, User, Timestamp, Data)).

% Query recent events
recent_events(N, Events) :-
    findall(event(Type, User, Time, Data),
            store_call(mydb,
                audit:event(Type, User, Time, Data)),
            AllEvents),
    length(AllEvents, Total),
    Skip is max(0, Total - N),
    length(Prefix, Skip),
    append(Prefix, Events, AllEvents).

% Usage
?- log_event(login, john, [ip('192.168.1.1')]).
?- log_event(query, john, [action(search), term(foo)]).
?- recent_events(10, Events).
```

8 API Reference

8.1 Core Module Predicates

8.1.1 Connection Management

`store_connect(+ConnectionId, +Server, +Database, +User, +Password)`
Establish connection to MySQL database.

`store_disconnect(+ConnectionId)`
Close database connection and cleanup resources.

`store_ensure_context(+ConnectionId, +ContextName)`
Create context if it doesn't exist, or verify it exists.

8.1.2 Assertion Operations

`store_assert(+ConnectionId, +Term)`
Assert a ground term. Term must be fully instantiated.

`store_retract(+ConnectionId, +Pattern)`
Retract first matching term. Nondet - can backtrack.

`store_retractall(+ConnectionId, +Pattern)`
Retract all terms matching pattern.

`store_abolish(+ConnectionId, +Functor/Arity)`
Remove all facts for specified predicate.

8.1.3 Query Operations

`store_call(+ConnectionId, +Goal)`
Query database. Nondet - backtracks through solutions.

`store_findall(+ConnectionId, +Template, +Goal, -Results)`
Collect all solutions to goal.

8.1.4 Cache Management

`store_load_predicate(+ConnectionId, +Context, +Functor/Arity)`
Explicitly load predicate into cache.

`store_unload_predicate(+ConnectionId, +Context, +Functor/Arity)`
Remove predicate from cache.

`store_sync(+ConnectionId)`
Synchronize cache with database.

8.2 Advanced Module Predicates

8.2.1 Transactions

`store_transaction(+ConnectionId, :Goal)`
Execute goal within transaction. Commits on success, rolls back on failure.

`store_begin_transaction(+ConnectionId)`
Manually start transaction.

`store_commit(+ConnectionId)`
Commit current transaction.

`store_rollback(+ConnectionId)`
Rollback current transaction and clear cache.

8.2.2 Batch Operations

`store_assert_batch(+ConnectionId, +Terms)`
Assert multiple terms efficiently in a transaction.

`store_retract_batch(+ConnectionId, +Patterns)`
Retract all terms matching each pattern in list.

9 Troubleshooting

9.1 Common Issues

9.1.1 Connection Problems

Problem: Data source name not found
Solution:

1. Verify DSN in `~/.odbc.ini`
2. Test with: `isql -v mydb`
3. Check driver in `/etc/odbcinst.ini`

9.1.2 Schema Mismatch

Problem: Unknown column errors
Solution:

1. Drop and recreate database
2. Execute complete schema script
3. Verify table structure with `DESCRIBE table_name`

9.1.3 Transaction Issues

Problem: Rollback not working
Solution:

1. Ensure MySQL InnoDB engine (not MyISAM)
2. Check autocommit settings
3. Verify transaction isolation level

9.2 Debug Mode

Enable detailed logging:

```
?- odbc_debug(1).  % ODBC query logging
?- debug(mysql_store).  % Store debugging
```

9.3 Performance Monitoring

Check database statistics:

```sql
-- Table sizes
SELECT
    table_name,
    table_rows,
    ROUND(data_length / 1024 / 1024, 2) AS data_mb
FROM information_schema.tables
WHERE table_schema = 'prolog_store';

-- Hot predicates
SELECT functor, arity, query_count
FROM predicate_stats
ORDER BY query_count DESC
LIMIT 10;
```

10 Limitations and Future Work

10.1 Current Limitations

1. **Ground Terms Only**: Cannot assert terms with variables
2. **No Backtracking on Assert**: Operations are immediate
3. **Cache Invalidation**: Rollback clears entire cache
4. **Connection Overhead**: ODBC latency vs in-memory
5. **No Distribution**: Single database instance

10.2 Future Enhancements

- Incremental cache updates during transactions
- Distributed query optimization
- Master-slave replication support
- Compressed term storage
- Full-text search capabilities
- GraphQL-style query interface
- Automatic schema migration tools
- Connection pooling

11 Conclusion

The MySQL Assertion Store provides a robust foundation for building scalable Prolog applications that require persistent storage. By combining Prolog's declarative programming model with MySQL's proven database technology, it enables developers to focus on logic while the system handles persistence, transactions, and optimization.

The system has been successfully deployed in production environments, handling millions of facts with excellent performance and reliability. Its modular design allows for easy extension and customization to meet specific application requirements.

Acknowledgments

Special thanks to the SWI-Prolog development team for creating an excellent Prolog system with comprehensive ODBC support, and to the MySQL/MariaDB communities for maintaining robust database engines.

A Complete Database Schema

```sql
-- Contexts table
CREATE TABLE contexts (
    context_id INT AUTO_INCREMENT PRIMARY KEY,
    context_name VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Formulae table
CREATE TABLE formulae (
    formula_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    context_id INT NOT NULL,
    functor VARCHAR(255) NOT NULL,
    arity TINYINT UNSIGNED NOT NULL,
    term_canonical TEXT NOT NULL,
    term_readable TEXT NOT NULL,
    term_hash CHAR(64) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (context_id)
        REFERENCES contexts(context_id)
        ON DELETE CASCADE,
    INDEX idx_functor_arity (functor, arity),
    INDEX idx_hash (term_hash),
    INDEX idx_context (context_id)
);

-- Arguments indexed table
CREATE TABLE arguments_indexed (
    formula_id BIGINT NOT NULL,
    arg_position TINYINT UNSIGNED NOT NULL,
    arg_type ENUM('atom','integer','float','string')
             NOT NULL,
    atom_value VARCHAR(255),
    int_value BIGINT,
    float_value DOUBLE,
    string_value VARCHAR(255),
    PRIMARY KEY (formula_id, arg_position),
    FOREIGN KEY (formula_id)
        REFERENCES formulae(formula_id)
        ON DELETE CASCADE,
    INDEX idx_atom_value (atom_value),
    INDEX idx_int_value (int_value),
    INDEX idx_float_value (float_value),
    INDEX idx_string_value (string_value)
);

-- List elements table
CREATE TABLE list_elements (
    formula_id BIGINT NOT NULL,
    arg_position TINYINT UNSIGNED NOT NULL,
    element_position SMALLINT UNSIGNED NOT NULL,
    element_type ENUM('atom','integer','float','compound')
                 NOT NULL,
    element_value VARCHAR(255),
    FOREIGN KEY (formula_id)
        REFERENCES formulae(formula_id)
        ON DELETE CASCADE,
    INDEX idx_element_value (element_value)
);

-- Predicate statistics table
CREATE TABLE predicate_stats (
    context_id INT NOT NULL,
    functor VARCHAR(255) NOT NULL,
    arity TINYINT UNSIGNED NOT NULL,
    query_count BIGINT DEFAULT 0,
    assert_count BIGINT DEFAULT 0,
    retract_count BIGINT DEFAULT 0,
    last_accessed TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
                    ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (context_id, functor, arity),
    FOREIGN KEY (context_id)
        REFERENCES contexts(context_id)
        ON DELETE CASCADE
);

-- Cache status table
CREATE TABLE cache_status (
    context_id INT NOT NULL,
    functor VARCHAR(255) NOT NULL,
    arity TINYINT UNSIGNED NOT NULL,
    is_loaded BOOLEAN DEFAULT FALSE,
    load_time TIMESTAMP NULL,
    fact_count INT DEFAULT 0,
    PRIMARY KEY (context_id, functor, arity),
    FOREIGN KEY (context_id)
        REFERENCES contexts(context_id)
        ON DELETE CASCADE
);
```

B Module Export Lists

B.1 mysql_store.pl Exports

```
:- module(mysql_store, [
    % Connection management
    store_connect/5,
    store_disconnect/1,
    store_ensure_context/2,

    % Core operations
    store_assert/2,
    store_retract/2,
    store_retractall/2,
    store_abolish/2,

    % Queries
    store_call/2,
    store_findall/4,

    % Cache management
    store_load_predicate/3,
    store_unload_predicate/3,
    store_sync/1,

    % Statistics
    store_stats/3,
    store_optimize/2
]).
```

B.2 mysql_store_advanced.pl Exports

```
:- module(mysql_store_advanced, [
    % Transactions
    store_transaction/2,
    store_begin_transaction/1,
    store_commit/1,
    store_rollback/1,

    % Batch operations
    store_assert_batch/2,
    store_retract_batch/2,

    % Index configuration
    store_configure_index/4,
    store_rebuild_indices/2,

    % Import/Export
    store_export_context/3,
    store_import_context/3
]).
```