TSQL.APP Enterprise Architecture: Metadata-Driven Development

Building Large-Scale Enterprise Solutions Through Standardized Database Metadata

---

Executive Summary

TSQL.APP transforms enterprise application development through a revolutionary **two-database architecture** that separates business data from application interface definitions. Every TSQL.APP application consists of an unchanged business database and a standardized project database containing identical metadata structures. This enables the creation of sophisticated enterprise solutions like Warehouse Management Systems using only T-SQL expertise while preserving existing business investments.

This document explores the actual technical architecture based on the standardized metadata schema demonstrated in the tracy_wms_proj database.

Core Two-Database Architecture

Universal Application Pattern

Every TSQL.APP Application Has Exactly Two Databases:

1. **Business Database (x)**
    - Your existing or new business data (customers, orders, inventory, etc.)
    - Business logic and stored procedures
    - **Completely untouched by TSQL.APP**
    - Accessed via synonyms from the project database
2. **TSQL.APP Project Database (x_proj)**
    - **Identical metadata schema across ALL applications**
    - UI definitions, cards, actions stored as data in standardized tables
    - Framework procedures (sp_api_*)
    - Application behavior configured through metadata

Example Applications

Warehouse Management System:

- Business: tracy_wms (warehouse operations data)
- Project: tracy_wms_proj (UI metadata + WMS configurations)

Customer Relationship Management:

- Business: sales_crm (customer and sales data)
- Project: sales_crm_proj (UI metadata + CRM configurations)

Manufacturing ERP:

- Business: manufacturing_db (production data)
- Project: manufacturing_db_proj (UI metadata + ERP configurations)

Key Insight: All _proj databases have **identical table structures** but contain different **data configurations** that define each unique application.

Universal Metadata Schema: Every Project Database Contains These Exact Tables

Database Schema Analysis: Standard Metadata Structure

Reference Example: tracy_wms_proj

- **Server**: wmsx.nl,51624
- **Generated**: June 2, 2025
- **Purpose**: Standard TSQL.APP metadata schema with WMS-specific configurations

Core Application Definition Tables (Universal)

Every x_proj Database Contains These Exact Tables:

sql

-- Primary application building blocks (identical schema in every project)

dbo.api_card *-- UI screens/forms (PK: id)*

├── copy_from_card → api_card.id (template inheritance)

├── lookup_field_id → api_card_fields.id

├── menu_group_id → api_menu_group.id

└── run_always_card_actions_id → api_card_actions.id

dbo.api_card_fields *-- Field definitions (PK: id)*

├── card_id → api_card.id

dbo.api_card_actions *-- Button/action definitions (PK: id)*

├── card_id → api_card.id

├── group_id → api_action_groups.id

└── master_card_action_id → api_card_actions.id (hierarchical actions)

User Interface Framework (Universal Schema)

sql

-- Modal and form components (same tables in every project)

dbo.api_card_modal *-- Modal dialog definitions*

├── card_id → api_card.id

dbo.api_card_modal_fields -- *Modal field layout*

├── field_id → api_card_fields.id

├── modal_id → api_card_modal.id

dbo.api_card_field_attributes -- *Field behavior and validation*

├── field_id → api_card_fields.id

├── attr_id → api_attributes.id

Workflow Engine (Universal Schema)

sql

-- *State machine implementation (same structure everywhere)*

dbo.api_card_states -- *Business process states*

├── card_id → api_card.id

dbo.api_card_states_actions -- *State transition rules*

├── card_state_id → api_card_states.id

├── card_action_id → api_card_actions.id

Security & User Management (Universal Schema)

sql

-- *Role-based access control (identical in every project)*

dbo.api_user -- *User accounts (PK: id)*

dbo.api_role -- *Role definitions (PK: id)*

dbo.api_user_role -- *User-role assignments*

├── api_user_id → api_user.id

├── api_role_id → api_role.id

How Applications Differ: Configuration Data, Not Schema

What Makes Each Application Unique

Applications built with TSQL.APP differ **only** in:

1. **Metadata Content** (data stored in the standard tables)
2. **Action Script Content** (T-SQL code in api_card_actions.script_code)

3. **Business Database Structure** (the x database schema)

Example: WMS vs CRM Configuration Differences

Same Table Structure, Different Data Content:

sql

```
-- WMS Application Configuration (in tracy_wms_proj.api_card)
INSERT INTO api_card (name, tablename, card_type) VALUES
   ('InboundReceiving', 'receipts', 'form'),
   ('InventoryLookup', 'inventory', 'list'),
   ('ShippingDock', 'shipments', 'workflow');

-- CRM Application Configuration (in sales_crm_proj.api_card)
INSERT INTO api_card (name, tablename, card_type) VALUES
   ('CustomerManagement', 'customers', 'form'),
   ('SalesOpportunities', 'opportunities', 'list'),
   ('QuoteGeneration', 'quotes', 'workflow');
```

Same Action Framework, Different Business Logic:

sql

```
-- WMS Receiving Action (in tracy_wms_proj.api_card_actions)
UPDATE api_card_actions
SET script_code = '
   DECLARE @ASN NVARCHAR(50);
   EXEC sp_api_modal_get_value @name="@ASN", @value=@ASN OUT;
   -- WMS-specific receiving logic here...
'
WHERE action_name = 'ProcessReceipt';

-- CRM Quote Action (in sales_crm_proj.api_card_actions)
UPDATE api_card_actions
SET script_code = '
   DECLARE @CustomerId INT;
   EXEC sp_api_modal_get_value @name="@CustomerId", @value=@CustomerId OUT;
   -- CRM-specific quoting logic here...
```

'

WHERE action_name = 'GenerateQuote';

Universal Framework API: sp_api_* Stored Procedures

Identical Framework in Every Project Database

Every x_proj database contains the **exact same** sp_api_* stored procedures:

Core Modal Functions (Universal)

sql

-- *These procedures exist identically in every x_proj database*

-- *Initialize modal container*

EXEC sp_api_modal_modal @class=@ModalClass;

-- *Display text and headers*

EXEC sp_api_modal_text @text=@ModalTitle, @class=N'h2';

-- *Input controls*

EXEC sp_api_modal_input @name=N'@FieldName', @value=@Value OUT, @placeholder=N'Enter value';

-- *Button controls with state management*

EXEC sp_api_modal_button

@name=N'@ButtonName',

@value=N'Button Text',

@valueout=@ButtonName OUT,

@class=N'btn-primary',

@key=N'Enter',

@inline=1;

Data Table Management (Universal)

sql

-- *Identical procedures in every project for data handling*

-- *Editable data tables from temp tables*

EXEC sp_api_modal_table_update

@tmptable=N'#DataTable',

```
    @focusx=4,
    @addnew=0;

-- Display-only tables with formatting
EXEC sp_api_modal_table
    @tmptable=N'#DisplayTable',
    @print=1,
    @excel=1,
    @orderby=N'ORDER BY column1, column2';
```

State Persistence System (Universal)

```
sql
-- Same state management across all applications
-- Get current modal state
EXEC sp_api_modal_get_value @name=N'@Variable', @value=@Variable OUT;

-- JSON state management for complex workflows
SET @JsonState = (
    SELECT * FROM #TempTable FOR JSON PATH
);
SET @RestartValues = N'{"@JsonData": ' + @JsonState + N'}';

-- Restart modal with preserved state
EXEC sp_api_modal_restart @values=@RestartValues;
```

Enterprise Integration Architecture (Universal Schema)

API Data Set Framework (Standard in Every Project)

External API Management Tables (Identical Structure)

```
sql
-- These tables exist in every x_proj database with same schema
dbo.api_data_set            -- API endpoint definitions
├── apikey                  → api_data_set_keys.id
├── ip_address_id           → api_ip_addresses.id
```

dbo.api_data_set_keys -- *API authentication*

├── apikey_owner_card_id → api_card.id

├── parent_apikey_id → api_data_set_keys.id (hierarchical keys)

dbo.api_data_set_posts -- *API request logging*

├── data_set_id → api_data_set.id

dbo.api_data_set_stats -- *API usage analytics*

├── data_set_id → api_data_set.id

├── ip_address_id → api_ip_addresses.id

IP Address and Security Management (Universal)

sql

-- *Network security framework (same tables in every project)*

dbo.api_ip_addresses -- *IP address registry*

dbo.api_ip_address_hits -- *Access logging*

├── ip_address_id → api_ip_addresses.id

├── first_card_id → api_card.id

dbo.api_user_ip_address -- *User-IP associations*

├── api_user_id → api_user.id

├── ip_address_id → api_ip_addresses.id

File Management System (Universal Schema)

Enterprise File Handling (Standard Tables)

sql

-- *File management tables (identical in every x_proj database)*

dbo.api_files -- *File registry*

├── file_category_id → api_file_category.id

dbo.api_file_category -- *File categorization*

dbo.api_file_auto_detect -- *Automatic file classification*

├── set_category_id → api_file_category.id

├── test_api_files_id → api_files.id

dbo.api_file_token -- *Secure file access*

├── api_files_id → api_files.id

dbo.api_file_token_hits -- *File access tracking*

├── api_file_token_id → api_file_token.id

├── ip_address_id → api_ip_addresses.id

Business Database Integration Pattern

How Project Databases Connect to Business Databases

Synonym-Based Access Pattern:

sql

-- In x_proj database, create synonyms to access x database

-- Example: tracy_wms_proj accessing tracy_wms

-- Business data access (read-only through synonyms)

CREATE SYNONYM [inventory_items]

FOR [tracy_wms].[dbo].[inventory_items];

CREATE SYNONYM [warehouse_locations]

FOR [tracy_wms].[dbo].[warehouse_locations];

-- Action scripts in tracy_wms_proj can now access business data:

SELECT product_code, quantity_on_hand

FROM inventory_items

WHERE location_code = @SelectedLocation;

Dynamic Business Database Reference:

sql

-- Every project database includes this function

SELECT dbo.main_db() *-- Returns 'tracy_wms' when called from tracy_wms_proj*

```
-- Use in dynamic SQL for cross-database operations
DECLARE @BusinessDB NVARCHAR(128) = dbo.main_db();
DECLARE @SQL NVARCHAR(MAX) = CONCAT(
    N'SELECT * FROM ', QUOTENAME(@BusinessDB), N'.dbo.customers'
);
EXEC sp_executesql @SQL;
```

Development Process: Configuration vs. Coding

Application Development Workflow

1. Business Database Setup

```
sql
-- Create or use existing business database
CREATE DATABASE inventory_system;
GO

-- Create business tables (inventory, products, locations, etc.)
-- Business logic and stored procedures
```

2. Project Database Creation

```
sql
-- Create standardized TSQL.APP project database
CREATE DATABASE inventory_system_proj;
GO

-- Deploy standard TSQL.APP metadata schema (identical to all projects)
-- Results in ~100 standard metadata tables
```

3. Application Configuration (Data Entry, Not Coding)

```
sql
-- Configure cards (UI screens) through metadata
INSERT INTO api_card (name, tablename, card_type) VALUES
    ('InventoryManagement', 'inventory_items', 'form'),
    ('LocationLookup', 'warehouse_locations', 'list');
```

```
-- Configure fields through metadata
INSERT INTO api_card_fields (card_id, field_name, field_type) VALUES
  (@InventoryCardId, 'product_code', 'barcode_input'),
  (@InventoryCardId, 'quantity', 'decimal_input');

-- Configure actions through metadata
INSERT INTO api_card_actions (card_id, action_name, script_code) VALUES
  (@InventoryCardId, 'UpdateQuantity', '
    DECLARE @ProductCode NVARCHAR(50);
    DECLARE @NewQuantity DECIMAL(18,2);

    EXEC sp_api_modal_get_value @name=N"@ProductCode", @value=@ProductCode OUT;
    EXEC sp_api_modal_get_value @name=N"@NewQuantity", @value=@NewQuantity OUT;

    -- Business logic here using synonyms to access inventory_system
    UPDATE inventory_items
    SET quantity_on_hand = @NewQuantity
    WHERE product_code = @ProductCode;
  ');
```

What Developers Actually Work With

Developers Never Modify:

- Database schemas in x_proj (always identical)
- Framework procedures (sp_api_*)
- Metadata table structures

Developers Configure:

- **Data content** in metadata tables
- **T-SQL scripts** in api_card_actions.script_code
- **Business database** structure and logic

Real-World Enterprise Example: WMS Implementation

Tracy WMS: Two-Database Architecture

Business Database: tracy_wms

sql

```
-- Warehouse business data (completely independent of TSQL.APP)
CREATE TABLE inventory_items (
    item_id INT PRIMARY KEY,
    product_code NVARCHAR(50),
    quantity_on_hand DECIMAL(18,2),
    location_code NVARCHAR(20)
);

CREATE TABLE warehouse_locations (
    location_id INT PRIMARY KEY,
    location_code NVARCHAR(20),
    zone_type NVARCHAR(50)
);

-- Business stored procedures
CREATE PROCEDURE sp_process_receipt
    @asn_number NVARCHAR(50),
    @items XML
AS
BEGIN
    -- Warehouse-specific receiving logic
END
```

Project Database: tracy_wms_proj

sql

```
-- Standard TSQL.APP metadata with WMS-specific configurations

-- WMS Cards Configuration (data in standard api_card table)
INSERT INTO api_card (name, tablename, description) VALUES
    ('InboundReceiving', 'receipts', 'ASN and receipt processing'),
    ('InventoryLookup', 'inventory_items', 'Real-time inventory inquiry'),
```

```
    ('CycleCountEntry', 'cycle_counts', 'Physical inventory counting');

-- WMS Action Scripts (T-SQL in standard api_card_actions table)
INSERT INTO api_card_actions (card_id, action_name, script_code) VALUES
    (@ReceivingCardId, 'ProcessASN', '
        DECLARE @ASNNumber NVARCHAR(50);
        DECLARE @ItemsJson NVARCHAR(MAX);

        -- Get input from modal interface
        EXEC sp_api_modal_get_value @name=N"@ASNNumber", @value=@ASNNumber OUT;
        EXEC sp_api_modal_get_value @name=N"@ItemsJson", @value=@ItemsJson OUT;

        -- Call business database procedure through synonym
        EXEC sp_process_receipt @asn_number=@ASNNumber, @items=@ItemsJson;

        -- Provide user feedback
        EXEC sp_api_toast @text=N"Receipt processed successfully", @class=N"btn-success";
    ');
```

Portable Application Definition

Export/Import Configuration:

```
sql
-- Export WMS configuration from tracy_wms_proj
SELECT * FROM api_card WHERE name LIKE '%Receiving%' FOR JSON PATH;
SELECT * FROM api_card_actions WHERE action_name LIKE '%ASN%' FOR JSON PATH;

-- Import into new_warehouse_proj for different warehouse
-- Same metadata schema, different business database connection
```

Universal System Administration Framework

Identical Monitoring in Every Project

System Health Tables (Same Schema Everywhere)

```
sql
```

-- *These tables exist identically in every x_proj database*

dbo.api_database_check -- *Health check definitions*

dbo.api_database_check_result -- *Health check results*

├── database_check_id → api_database_check.id

dbo.api_deadlocks -- *Deadlock monitoring*

dbo.api_errors -- *Error logging*

dbo.api_errors_ignore -- *Error filtering rules*

Performance and Audit Framework (Universal)

sql

-- *Performance monitoring (same tables in every project)*

dbo.api_action_stats -- *Action performance metrics*

dbo.api_action_debug -- *Debug information*

dbo.api_history -- *System activity history (bigint PK for scale)*

-- *Audit and compliance (universal schema)*

dbo.api_card_audit -- *Card access auditing*

dbo.api_track_changes_table -- *Change tracking configuration*

dbo.api_track_changes_column -- *Column-level change tracking*

├── track_changes_table_id → api_track_changes_table.id

Platform Benefits: Standardization at Scale

Enterprise Advantages

1. Consistent Development Experience

- Same metadata schema across all applications
- Identical sp_api_* procedures everywhere
- Standardized development patterns
- Transferable skills between projects

2. Application Portability

- Export metadata from one project
- Import into another project

- Adapt to different business databases
- Rapid application templates

3. Maintenance Efficiency

- Framework updates apply to all projects
- Consistent troubleshooting procedures
- Shared knowledge base across applications
- Unified monitoring and administration

4. Business Data Protection

- Business databases remain untouched
- UI changes don't affect core data
- Legacy systems preserved
- Risk-free interface modernization

Development Team Benefits

Database Experts Can Build Full Applications:

- No frontend/backend separation
- T-SQL skills directly applicable
- Immediate access to business data
- Familiar debugging and optimization tools

Rapid Enterprise Development:

- Standard patterns accelerate development
- Proven framework reduces risks
- Business logic stays in database layer
- Quick adaptation to changing requirements

Conclusion: The Power of Standardized Metadata

The TSQL.APP enterprise architecture revolutionizes business application development through its **universal two-database pattern**. Every application built with TSQL.APP follows the same architectural principles:

Universal Elements (Identical Everywhere):

- Project database metadata schema (~100 standard tables)
- Framework procedures (sp_api_*)
- Development patterns and practices

- Administration and monitoring tools

Application-Specific Elements (Configuration Data):

- Metadata table content defining UI and behavior
- T-SQL action scripts implementing business logic
- Business database structure and data

This standardization enables:

- **Rapid Enterprise Development**: Standard patterns accelerate delivery
- **Consistent Quality**: Proven framework reduces defects
- **Scalable Architecture**: Same patterns work from simple forms to complex ERP
- **Future-Proof Investment**: Applications built on stable, standardized foundation

The tracy_wms_proj example demonstrates how a sophisticated Warehouse Management System can be built using this standardized metadata approach, proving that complex enterprise applications can be developed and maintained efficiently by database professionals using familiar T-SQL expertise.

TSQL.APP transforms enterprise application development from a complex multi-tier challenge into a systematic, database-driven process that leverages existing SQL skills while providing modern web application capabilities.