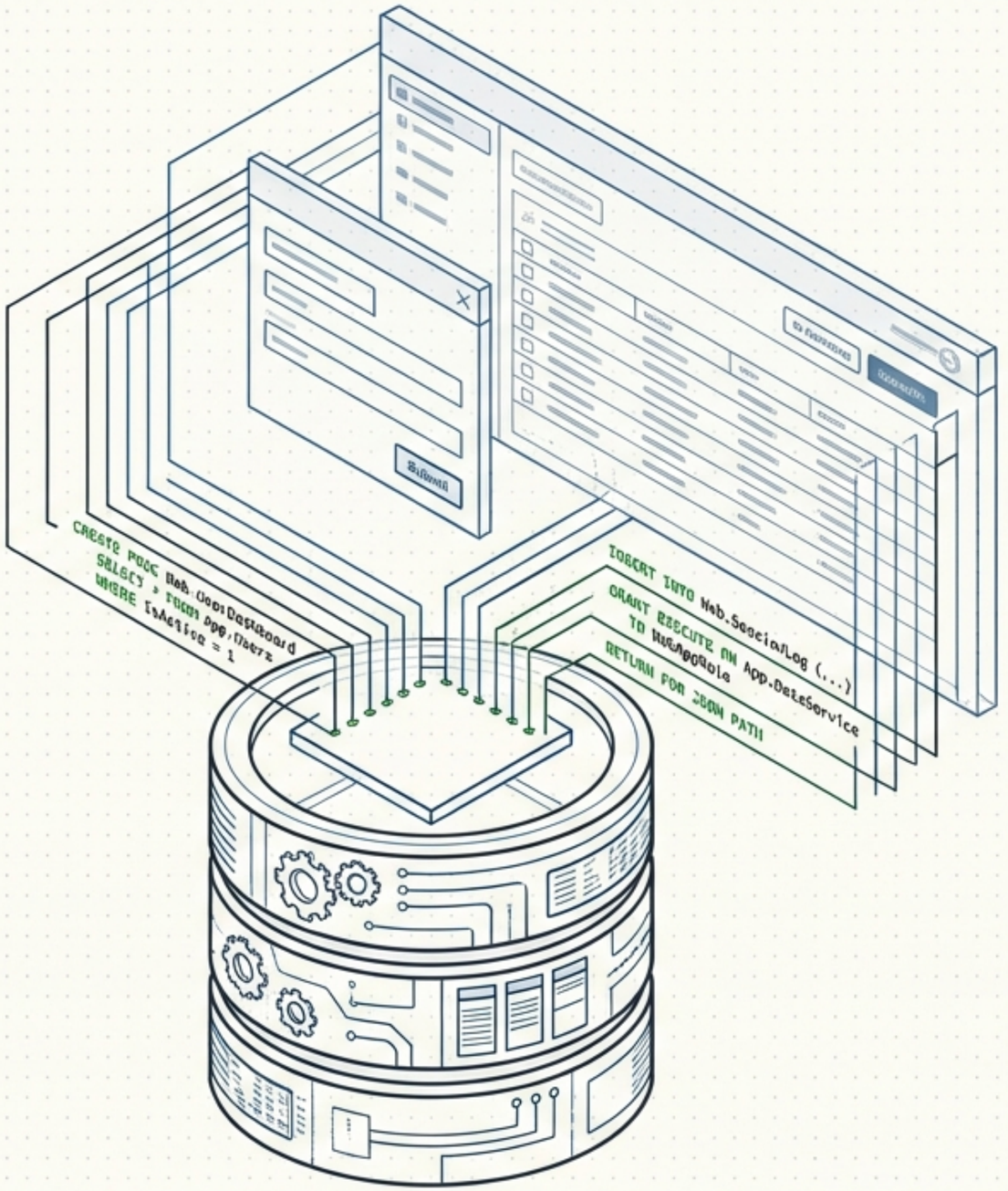
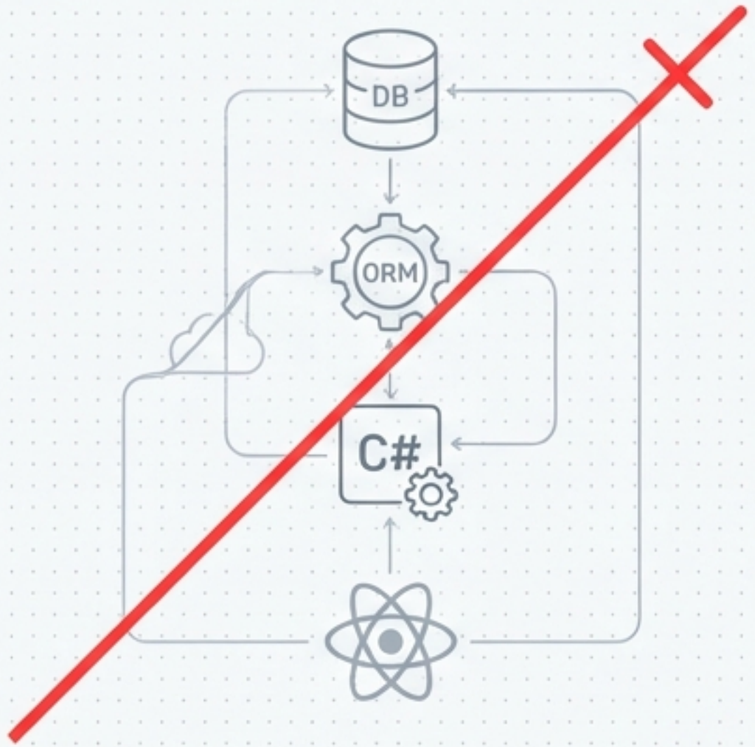
# Database-Only Architecture

Unpacking the TSQL.APP mental model: How SQL Server becomes a complete application platform.



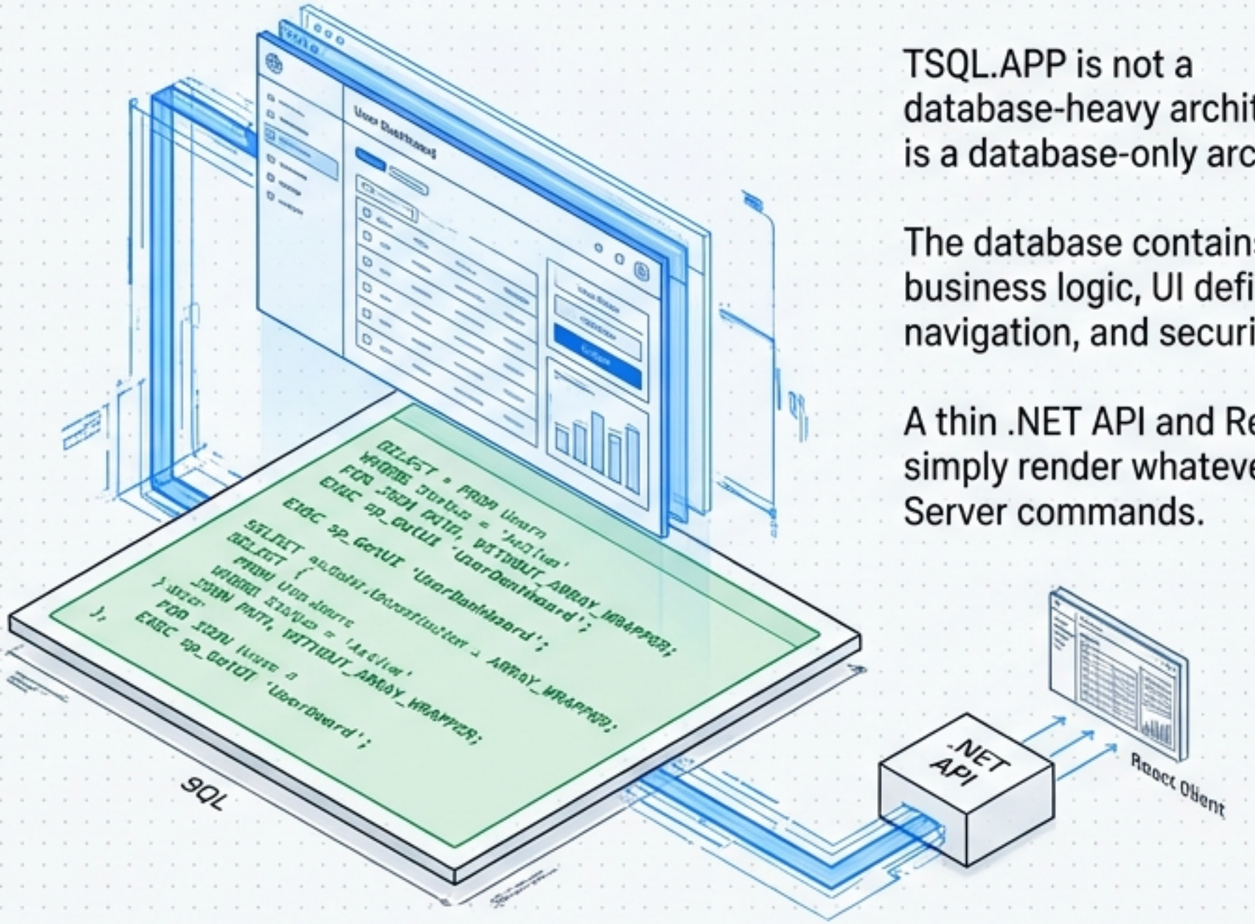
# The database is the complete application

(The Old Model)




~~No Backend Language.~~
~~No ORM. No Service Layer.~~

(The Paradigm Shift)



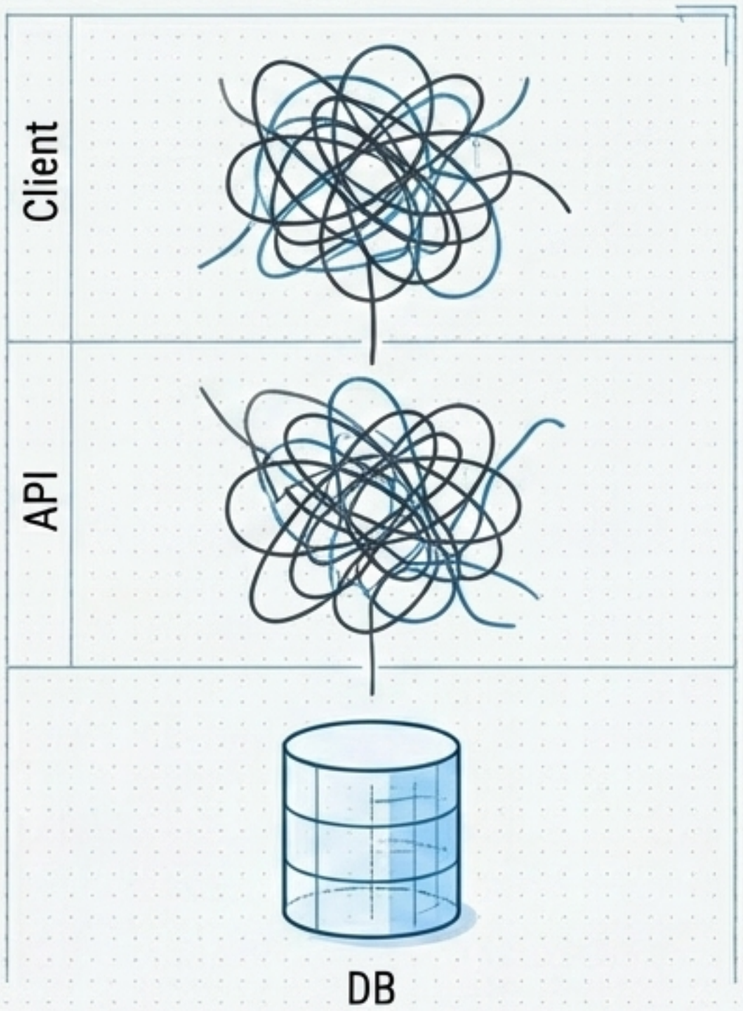

TSQL.APP is not a database-heavy architecture. It is a database-only architecture.

The database contains all business logic, UI definitions, navigation, and security.

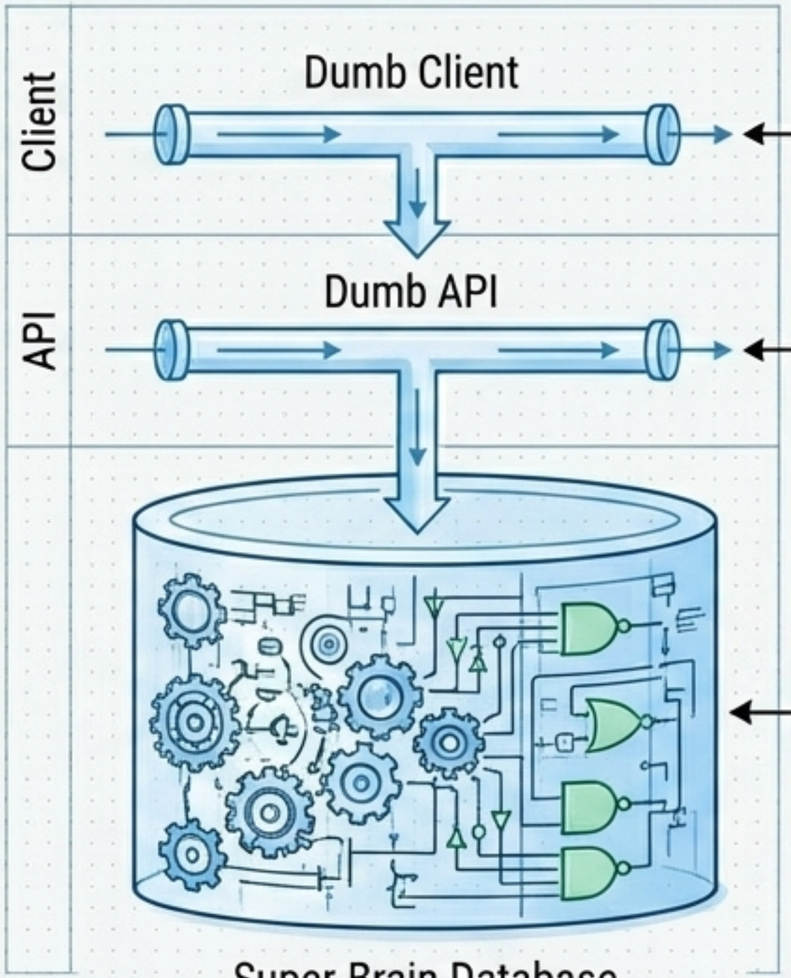
A thin .NET API and React client simply render whatever SQL Server commands.

# Reversing the traditional stack

## Traditional Stack




## TSQL.APP Stack

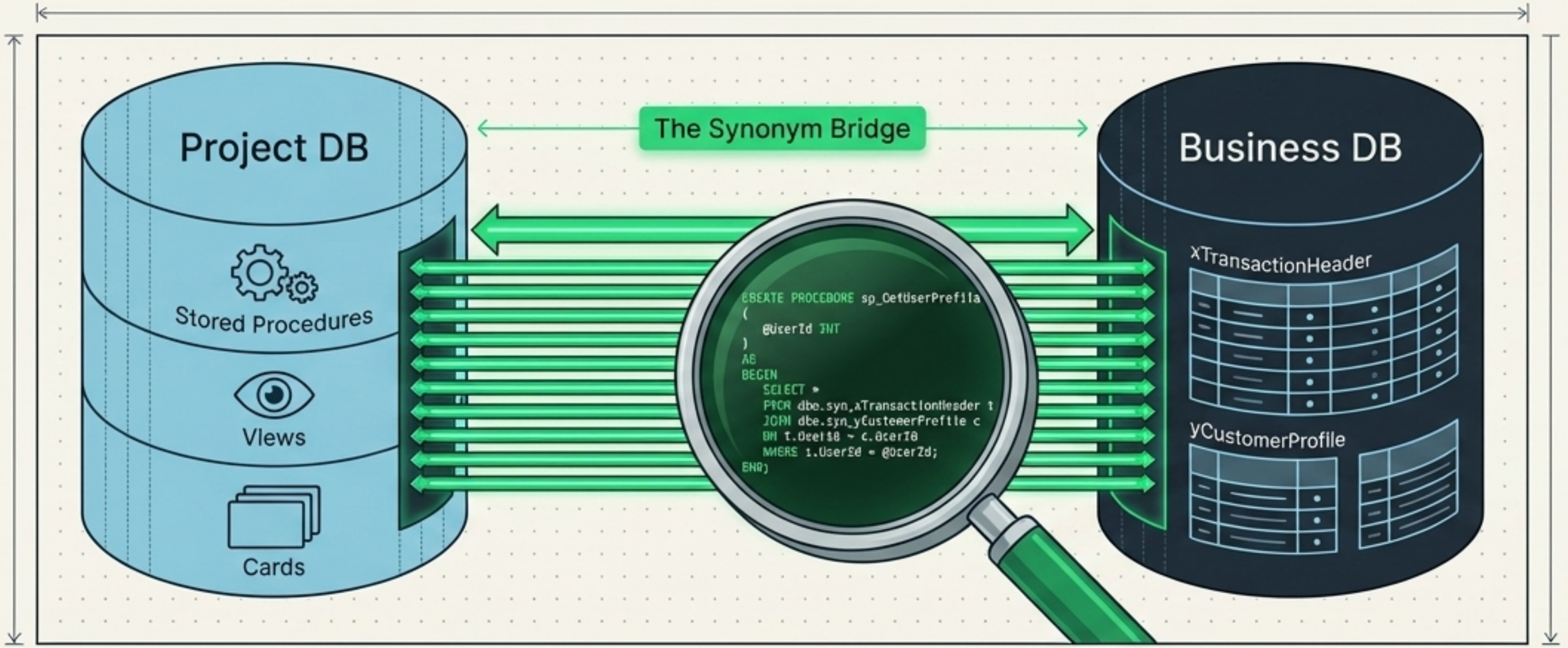



**The React Client:** Zero Opinions.
Receives JSON describing components (Listview, Modalview).
Never decides what to show, only knows how to show it.

**The .NET API:**
Zero Business Logic.
A dumb pipe. It packages HTTP requests into a context table and hands them to a stored procedure.

**The SQL Database:**
The Absolute Brain.
Executes logic, defines layouts, determines state, validates security, and builds the UI.

# The Twin Database architecture




## Business DB

Contains the actual application data tables.

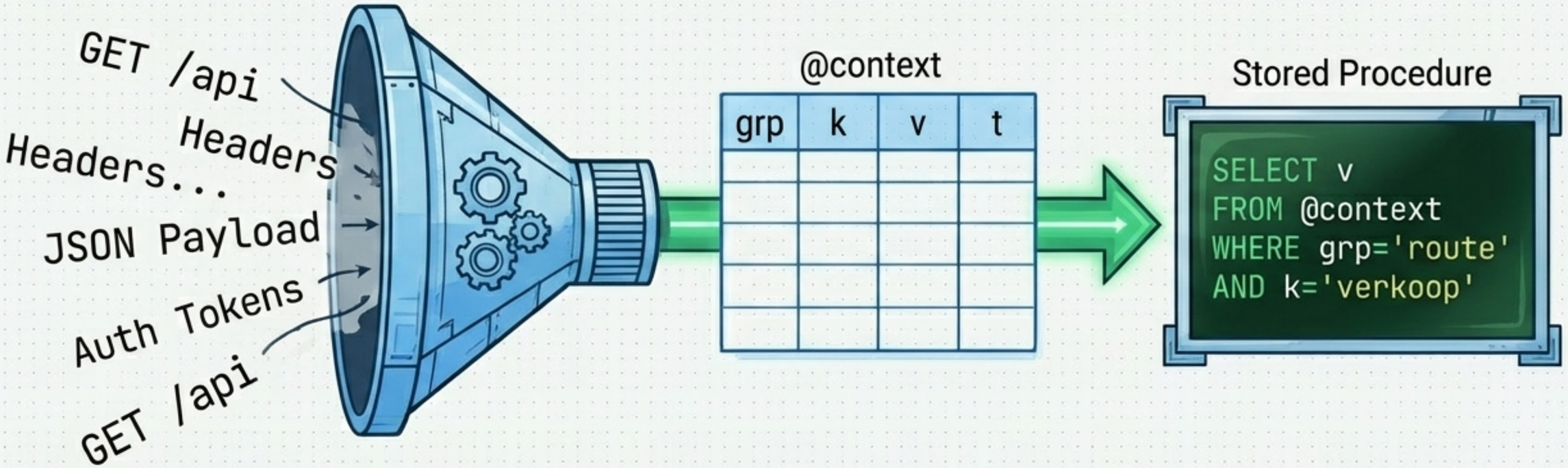
## Project DB

Contains the application logic (Stored Procedures, Views, Cards).

## The Synonym Bridge

The Project DB connects to the Business DB through synonyms—one for every business table. This allows UI stored procedures to query and join business tables transparently, as if they were local, with zero cross-database latency.

# Flattening the web request




### The Context Table

Every single HTTP request is flattened into a universal 4-column temp table (grp, k, v, t).

### Zero Parsing

The stored procedure never sees HTTP headers, URLs, or REST paradigms.

### The Execution

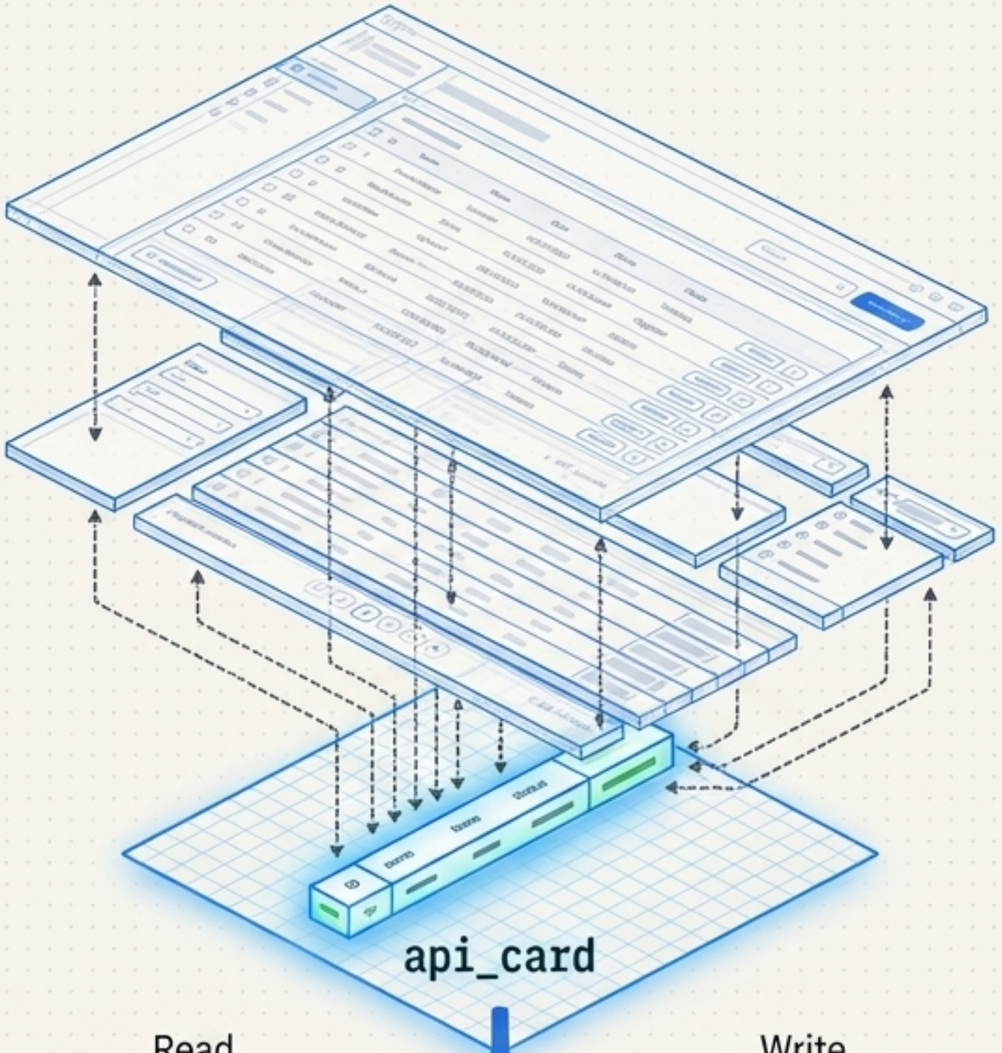
The SP simply queries its state dynamically from the context table.

# The Card is the universal building block

**The Split Architecture:**

`"tablename"` (often a rich view with `JOIN`s) dictates what the user sees.

`"basetable"` dictates where `INSERT/UPDATE/DELETE` operations actually write data.




**Reducers:**

Permanent `WHERE` clauses applied to all queries.

One table (`xTransactionHeader`) creates multiple distinct Cards (Sales vs. Purchases) simply by applying a reducer (TDT = 2 vs TDT = 1).

# Dynamic UI driven by row-level SQL

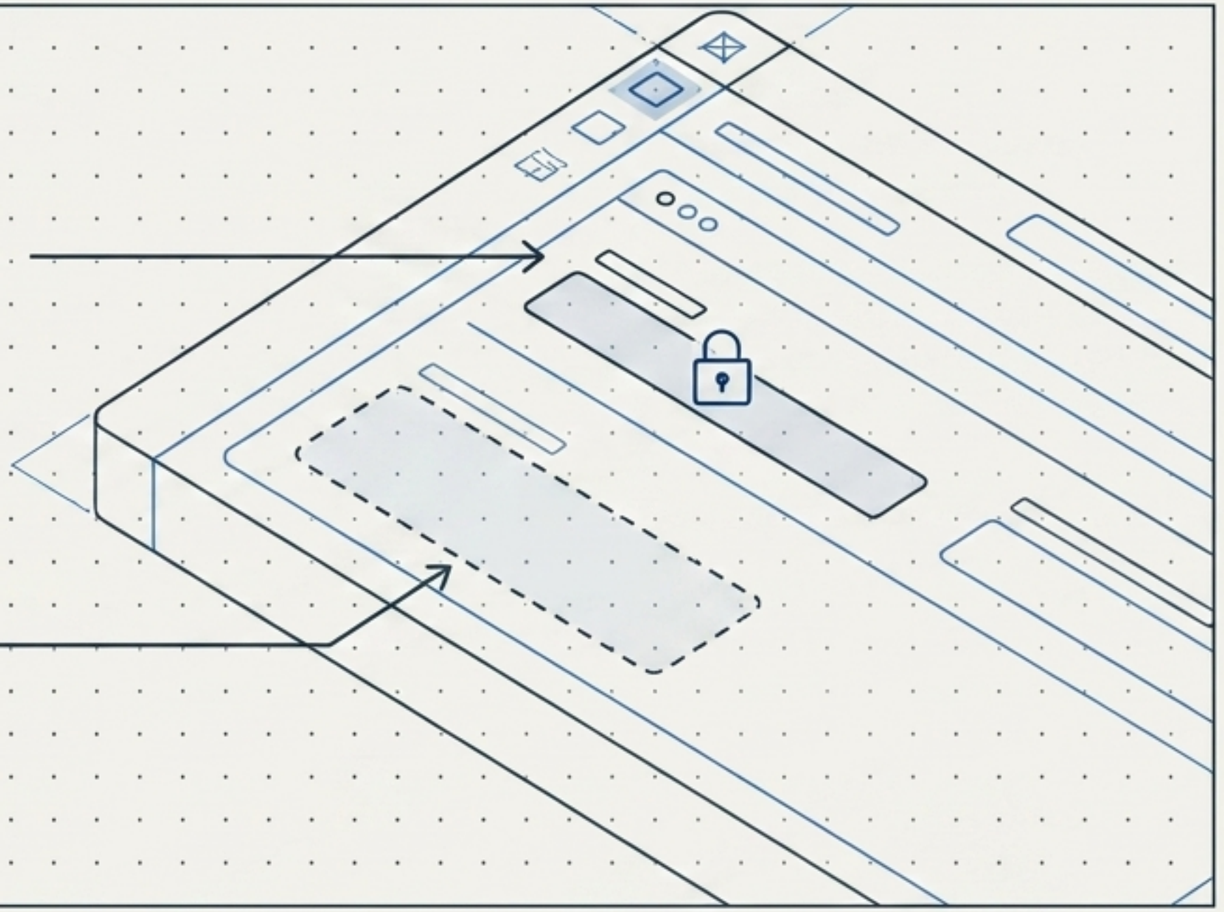
## Code-to-UI mapping

Code Block 1

```
is_updateable = iif(dbo.hasRole('admin')>0, 1, 0)
```

Code Block 2

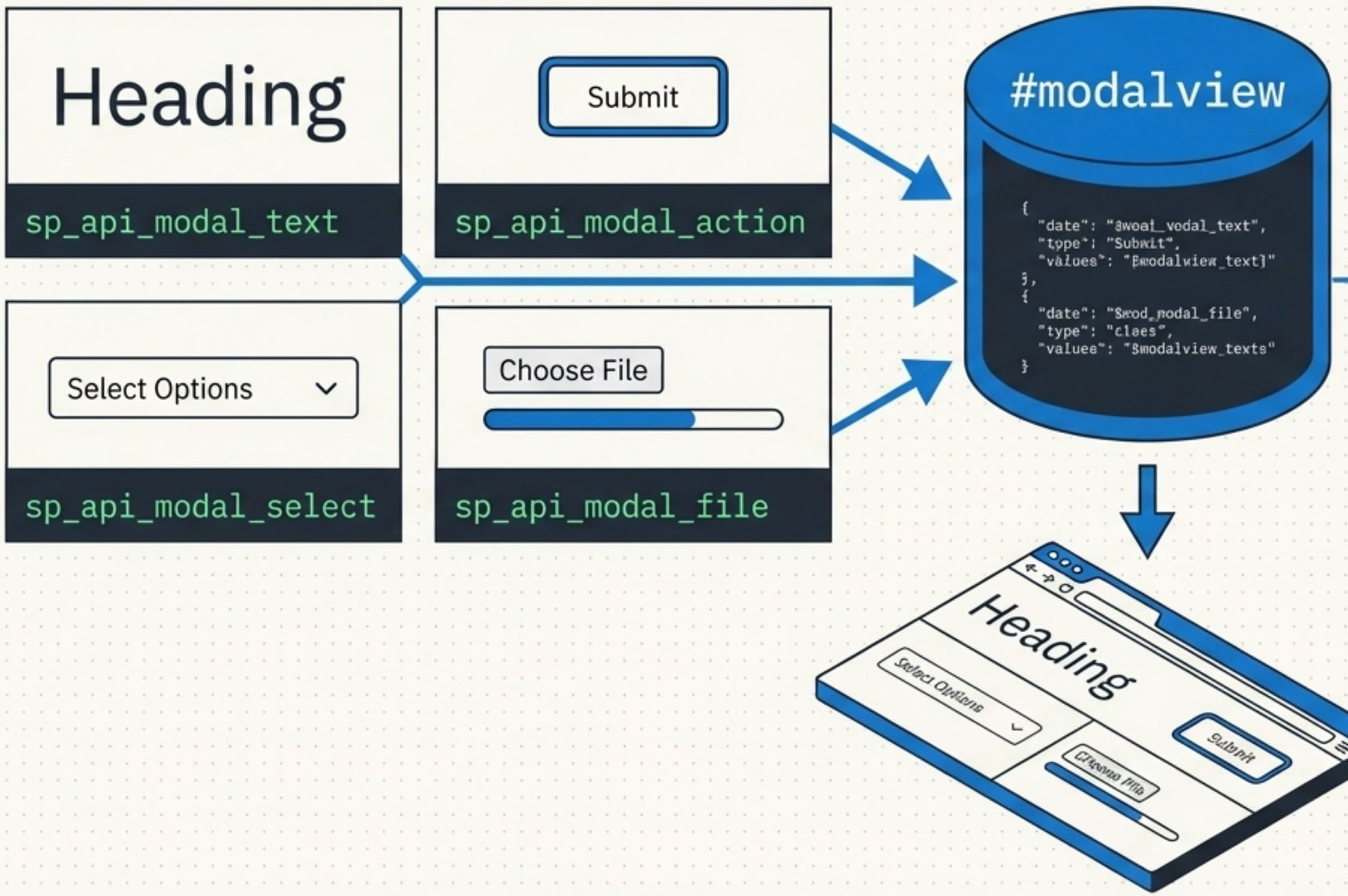
```
is_hidden = iif(@status='closed', 1, 0)
```



**Field Attributes:** UI behavior (api_card_field_attributes) is dictated by SQL expressions evaluated at render time, per record.

**The Result:** A field can be editable for one row and read-only for the next, entirely driven by database-level SQL evaluation without a single line of JavaScript.

# A complete UI toolkit inside T-SQL




**No HTML/CSS:**

Developers call stored procedures to define every component on the screen.

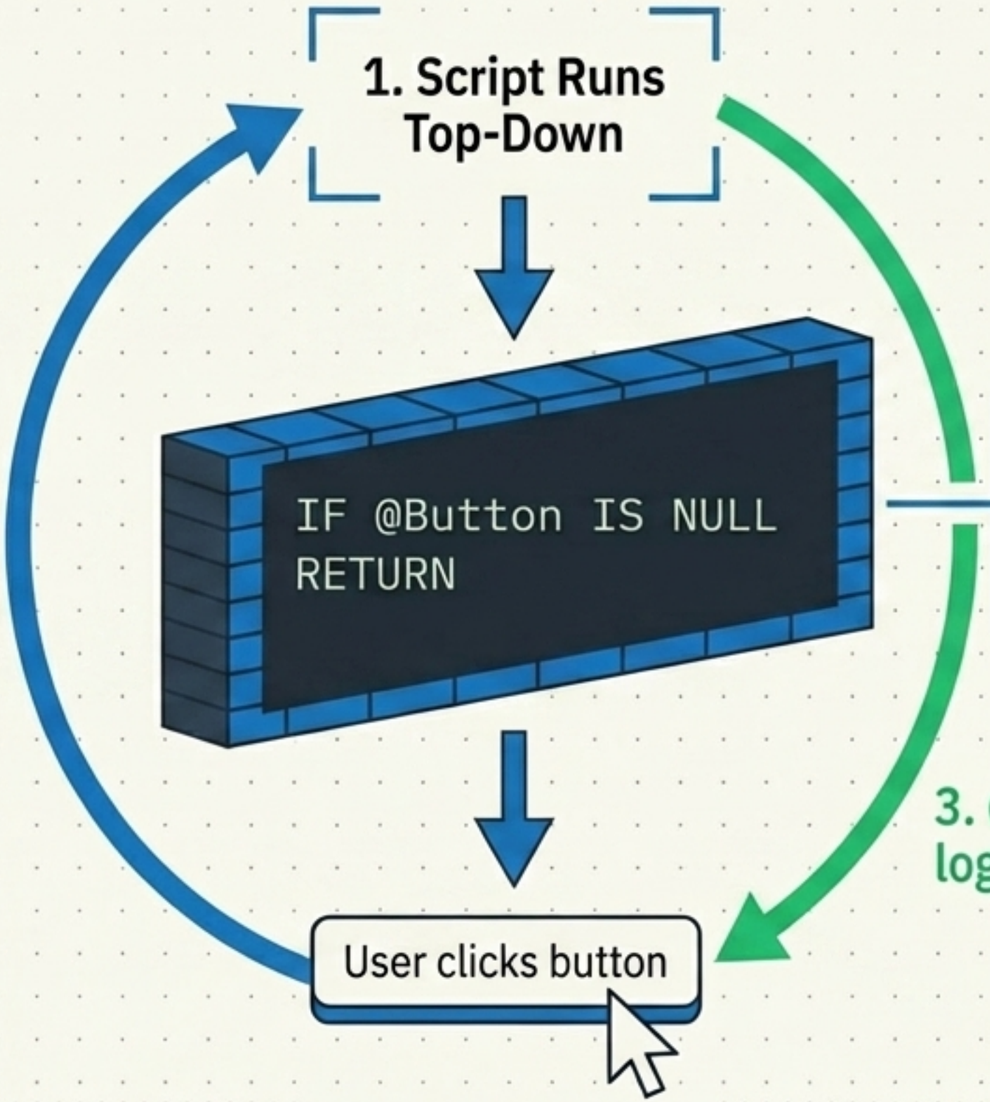
**The #modalview Engine:**

Every procedure call simply writes a JSON element to a temporary table in sequence.

**Blind Rendering:**

Upon execution completion, the React client reads `#modalview` and renders the exact sequence of elements as a full-screen modal overlay.

# Re-entrant scripts handle complex logic




**The Execution Model:**

Action scripts are not simple endpoints; they are multi-pass T-SQL batches.

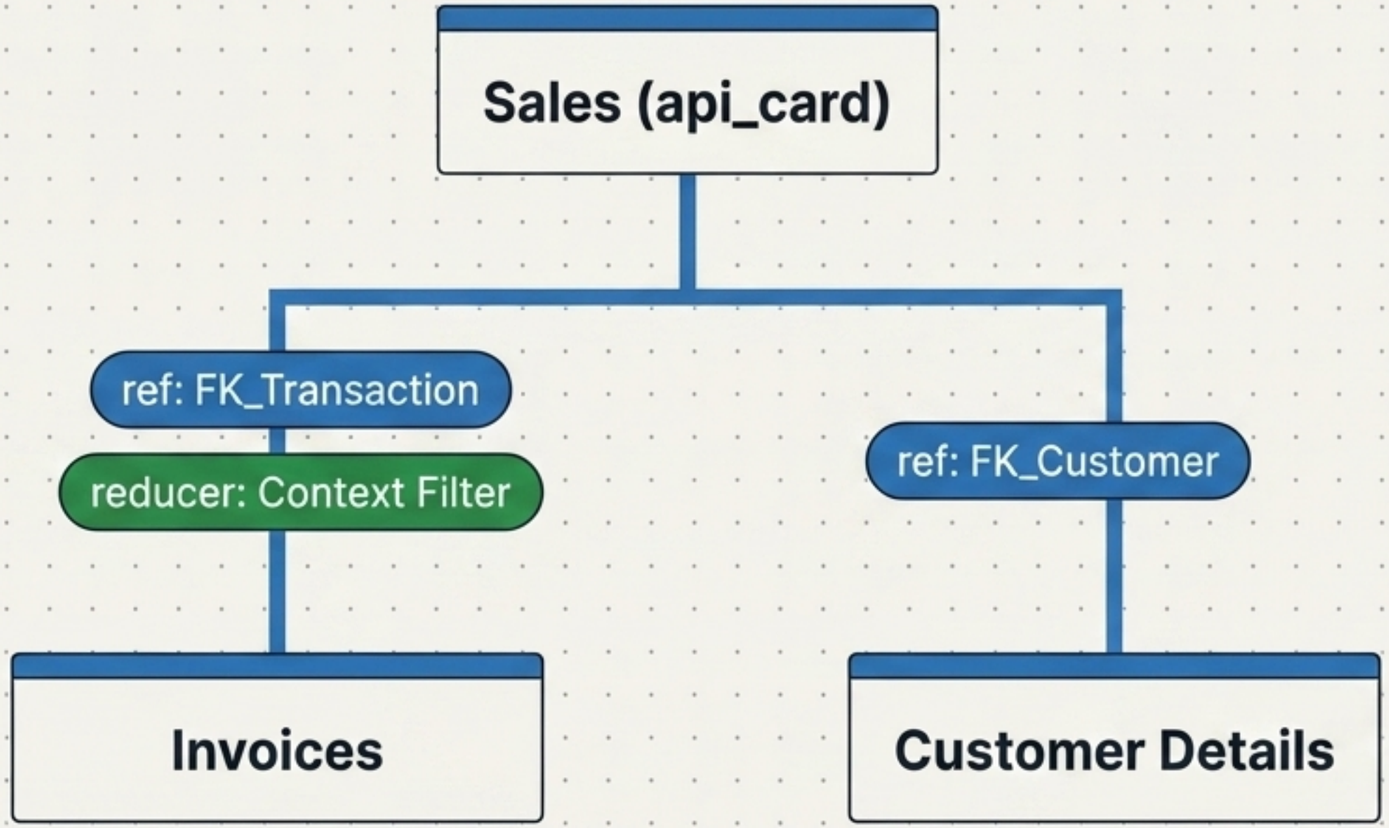
**State Preservation:**

'sp_api_modal_get_value' restores previous state automatically upon re-execution.

**Wizard Workflows:**

This 'Pause and Wait' pattern enables multi-step wizards and complex confirmations to exist entirely in pure T-SQL without any frontend state management.

# Routing without a frontend router




### Cards as a Tree:

Navigation is achieved simply by traversing the `'api_card_children'` relational table.

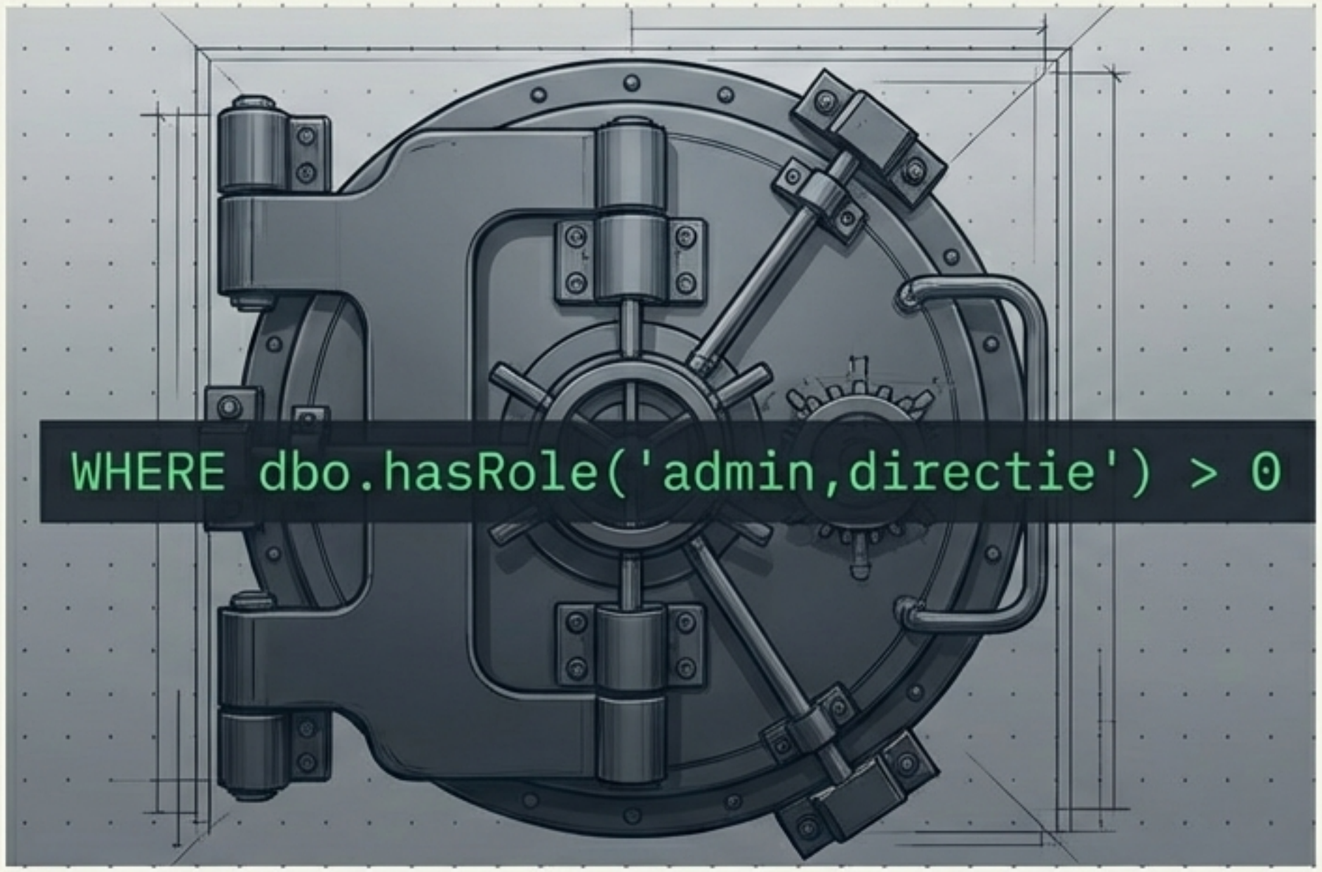
### Recursive Resolution:

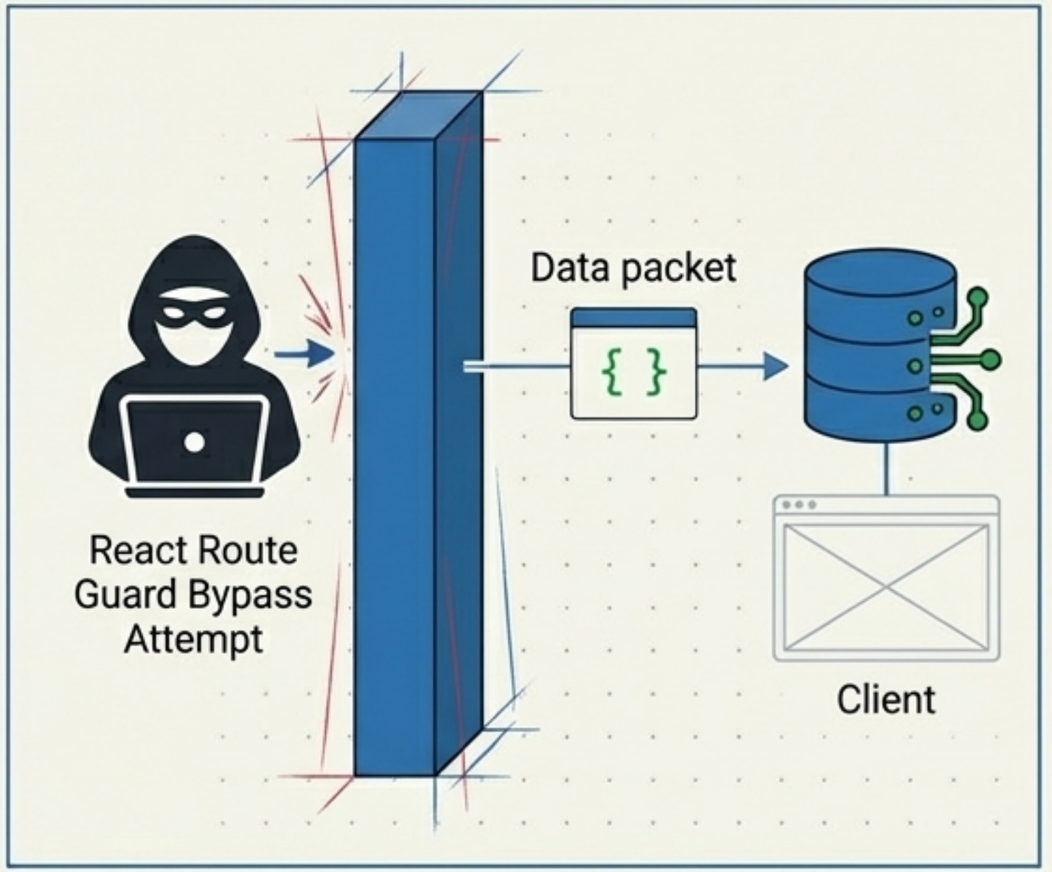
`'sp_api_card_list'` calls itself recursively to resolve deeply nested parent-child segments into a single view.

### Contextual Linking:

The `'ref'` column dictates the foreign key relationship, automatically injecting the correct `WHERE` clause when a user clicks into a child card.

# Absolute server-side security



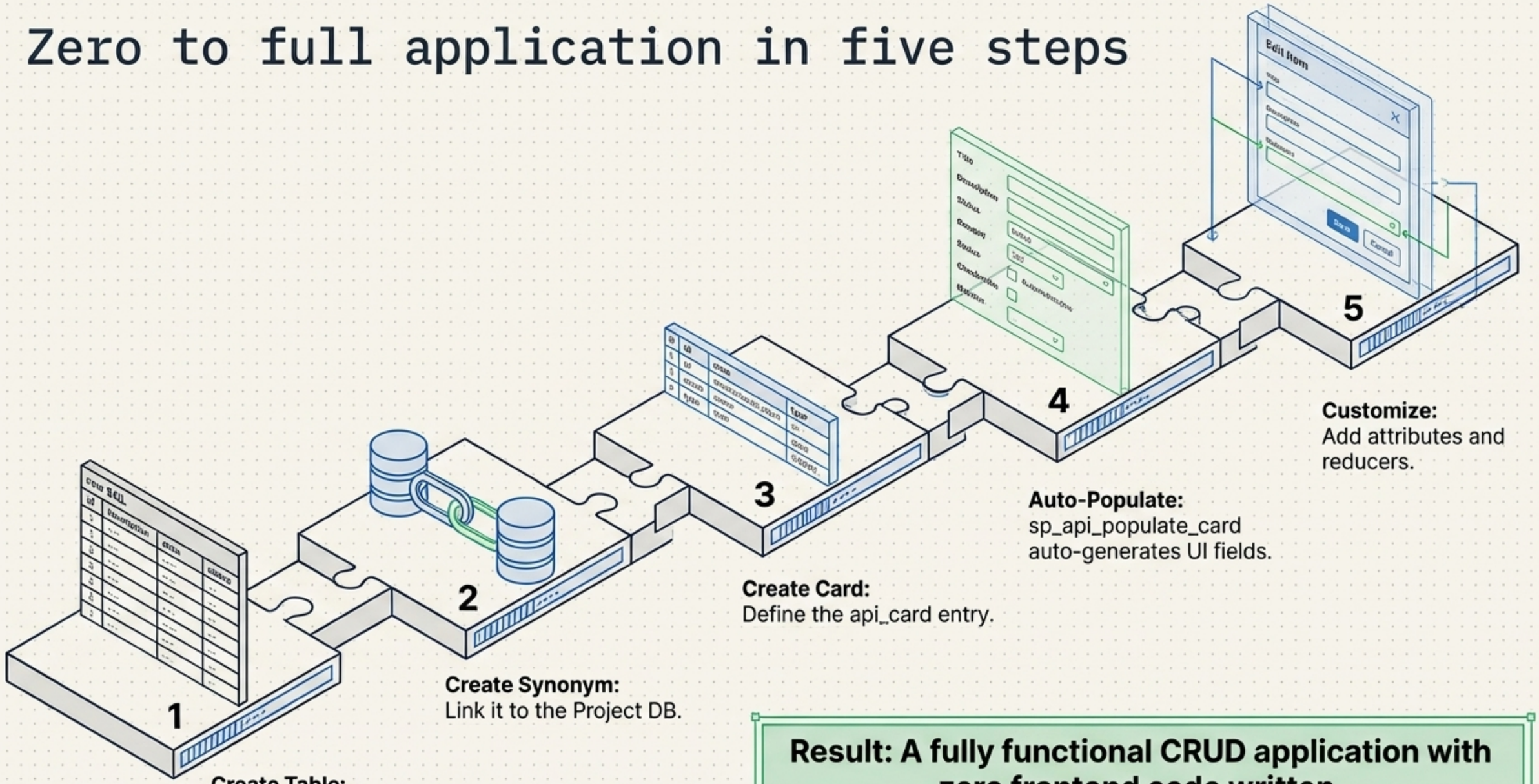




**No Client Secrets:**
The React client doesn't hide elements; it never receives them. There is no client-side route guard to bypass.

**Layered Evaluation:**
Security is enforced natively in the SP at the Global, Card, Row, and Field levels.

**Dynamic Permissions:**
If the SQL evaluation fails natively, the data simply never leaves the database engine.

# Zero to full application in five steps




**Create Table:**
Define your business data.

**Create Synonym:**
Link it to the Project DB.

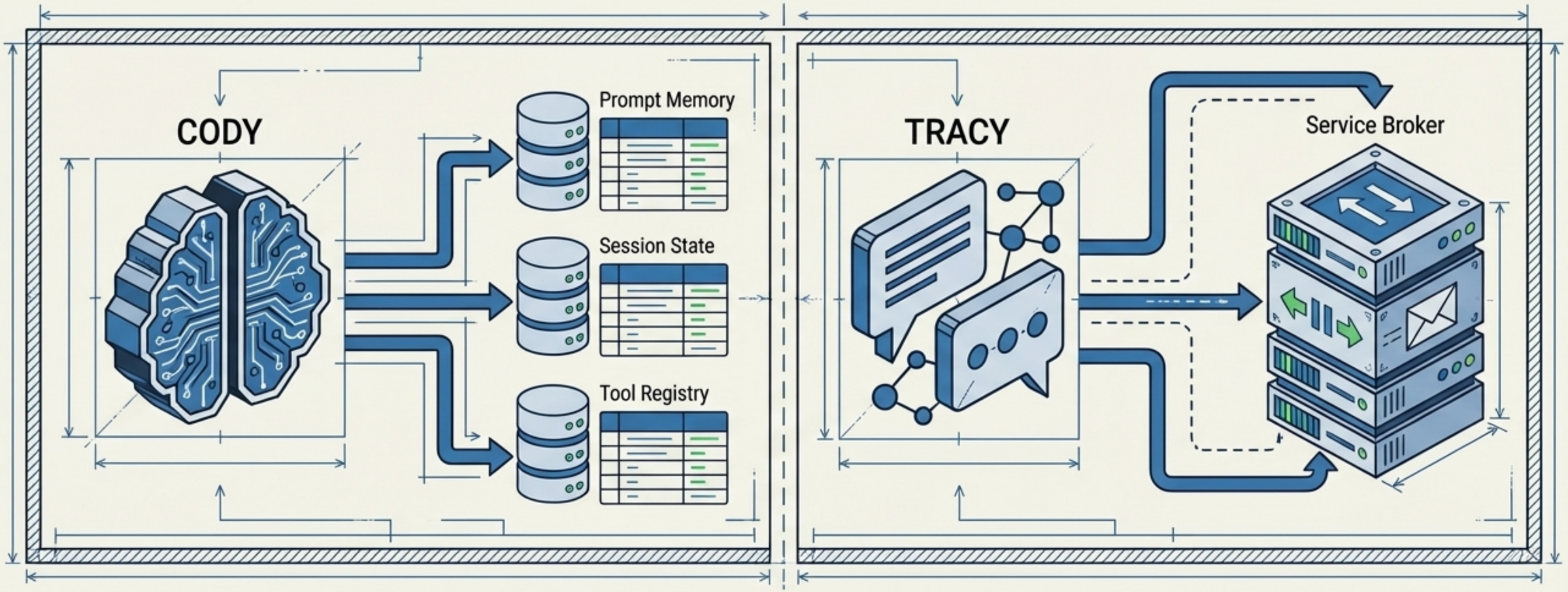
**Create Card:**
Define the api_card entry.

**Auto-Populate:**
sp_api_populate_card
auto-generates UI fields.

**Customize:**
Add attributes and reducers.

**Result: A fully functional CRUD application with zero frontend code written.**

# AI agents living natively in SQL Server

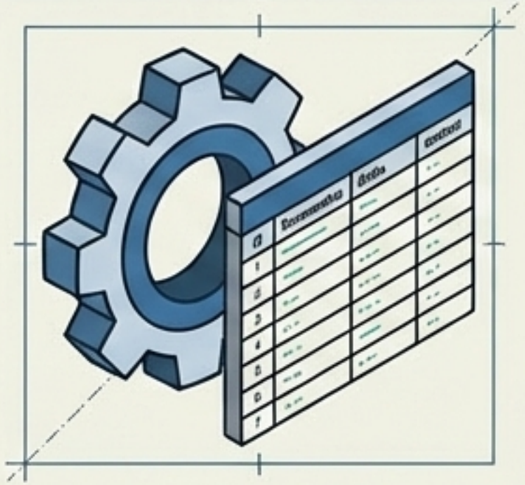



**CODY (Action Script Developer):** Writes T-SQL action scripts via chat. Uses database tables for persistent memory, session state, and dynamic tool registries.

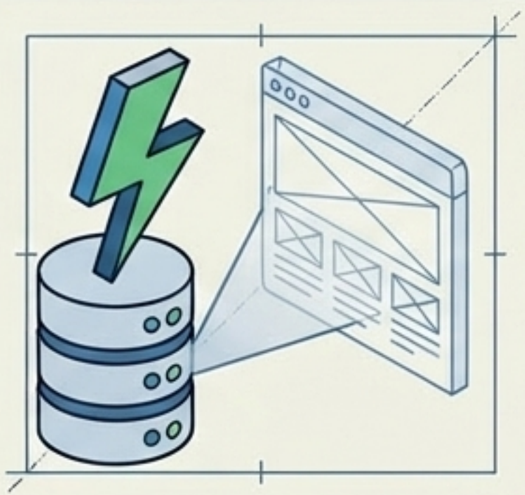
**Native Integration:** Prompts, memory, tools, and outputs are all stored as standard SQL Server tables and procedures, completely internal to the database.

**TRACY (Conversational Orchestrator):** Multi-model orchestrator (Grok/Claude). Executes SQL, inspects schemas, and orchestrates async operations via Service Broker.

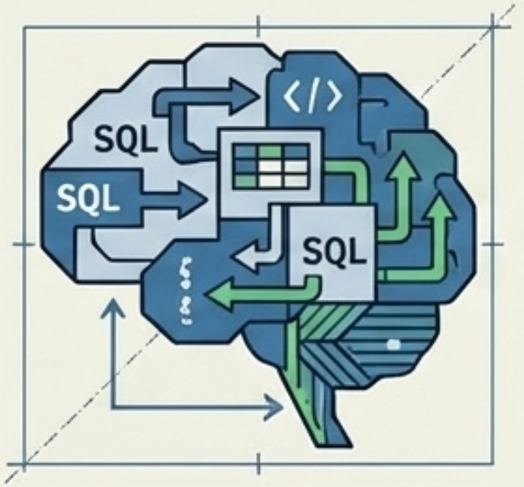
# The Database-Only advantage



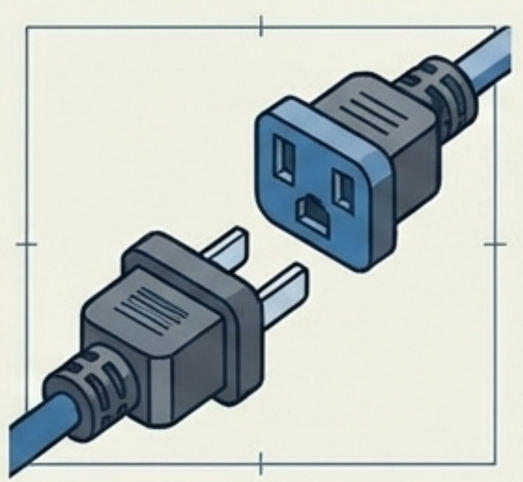
**Single Source of Truth:** No ORM impedance mismatch. No stale middle-tier cache.



**Instant Customization:** Add a column to a table, and the app updates immediately. No builds, no deployments.




**SQL as Logic:** Every developer who knows T-SQL is automatically a full-stack developer. Complex logic stays exactly where the data lives.



**Replaceable APIs:** The .NET pipe can be upgraded (e.g., .NET 2.2 to .NET 9) instantly without changing a single line of application logic.