# SuperGrid Feature Specification

**Status:** Draft — For Claude Code Implementation
**Created:** 2026-02-07
**Updated:** 2026-02-07
**Owner:** Michael Shaler

## Related Documents

- `cardboard-architecture-truth.md` — PAFV + LATCH + GRAPH taxonomy
- `SQLITE-MIGRATION-PLAN-v2.md` — Native data layer
- `specs/SuperGrid.md` — Original discussion & Q&A
- `.planning/SUPER-FEATURES.md` — Full Super* catalog
- `CLAUDE.md` — Architecture truth for Claude Code sessions
- `docs/INTEGRATION-CONTRACT.md` — Provider/hook architecture
- `src/types/supergrid.ts` — Current type definitions
- `phase-36-discussion-template.md` — Header hierarchy design decisions

---

## 1. What SuperGrid Is

SuperGrid is Isometry's signature view: a PAFV-projected multidimensional grid where nested headers create dimensional depth along both row and column axes, with orthogonal density controls and direct sql.js→D3.js rendering. It is a LATCH-primary view (separation) that can display n axes across stacked PAFV headers.

**The key differentiator:** The z-axis stacked headers create dimensional depth that no existing tool handles well. A 2D grid maps 2 axes to x/y planes. SuperGrid maps n axes with visual spanning, progressive disclosure, and density controls.

### Architecture Position

```
Layer 3 (D3.js Data Plane):  SuperGrid renderer —
SVG/canvas cells, headers, spans
Layer 4 (React Chrome):       FilterNav, density sliders,
axis pickers, toolbars
Layer 2 (sql.js):              LATCH queries → JSON → D3
data binding
```

**Rendering rule:** D3 shows the truth, React lets you change it.

---

# 2. Feature Catalog

Each Super* feature below includes: what it does, why it matters, how it behaves, and how to verify it works.

---

### 2.1 SuperStack — Nested PAFV Headers

**What:** Multi-level headers that visually span child dimensions. Parent headers group their children with merged visual boundaries.

```
Row Headers (3 axes deep):
┌─────────┬─────────┬─────────┐
│  Q1     │ Jan     │ Week 1 │ ← Level 0: Quarter
│         │         │ Week 2 │ ← Level 1: Month
│         │ Feb     │ Week 1 │ ← Level 2: Week
│         │         │ Week 2 │
│  Q2     │ Apr     │ Week 1 │
```

**Why:** This is the visual expression of PAFV's "any axis maps to any plane." Without SuperStack, users can only see flat 2D grids.

**Behavior:**

- Nesting depth is dynamic/unlimited — determined by data complexity
- Purely visual CSS/SVG spanning — the data model has individual rows
- When depth exceeds visual threshold, automatic grouping kicks in (semantic grouping first, then data density)
- Progressive disclosure: only 2-3 levels visible at once, with level picker tabs and zoom controls for navigation
- Headers form the primary breadcrumb navigational device (3D camera stairstepping down hierarchy)
- Morphing boundary animations on collapse/expand using D3 transitions
- Context menu for "Expand All" / "Collapse All" operations
- State persistence per-dataset and per-app
- Progressive rendering for expand/collapse unless consistently exceeding performance budgets, then fall back to lazy rendering (virtualize off-screen headers)

**Decisions (from phase-36 discussion):**

- Dynamic/unlimited nesting depth
- Automatic grouping + progressive disclosure for complexity management
- Semantic grouping → data density as grouping strategy, with user-configurable settings
- Level picker tabs + zoom controls for navigation (not breadcrumbs initially)
- Context menu for multi-level expand/collapse (starting point; Shift+click and double-click to be explored later)
- Per-dataset, per-app state persistence
- Morphing boundary animation style (D3 transitions)
- Progressive rendering with lazy rendering fallback

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Render 2-level header | Time(Quarter) → Time(Month) on y-axis | Q1 spans Jan/Feb/Mar rows | Visual span correct, click Q1 selects all children |
| Render 3-level header | Time(Year) → Quarter → Month | Year spans quarters, quarters span months | All spans mathematically correct |
| Auto-group at depth 5 | 5-axis deep hierarchy | Intermediate levels collapse to "Time" semantic group | ≤3 visible levels, expand available |
| Progressive disclosure | Click level picker tab | Headers morph to show | D3 transition < 300ms, no layout jank |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| | | selected depth level | |
| Collapse animation | Collapse Q1 header | Morphing boundary animation, children hidden | Animation at 60fps, state persisted |
| Empty span handling | Quarter with no data in one month | Span still renders correctly with empty indicator | No visual corruption |
| State persistence | Collapse Q1, navigate away, return | Q1 still collapsed | State loaded from SQLite |
| Context menu expand all | Right-click Q1 → "Expand All" | All nested levels within Q1 expand | All children visible recursively |

---

## 2.2 SuperDynamic — Axis Repositioning

**What:** Drag-and-drop axis repositioning across planes. Grab a column header axis and drag it to become a row header axis (transpose and beyond).

**Why:** Enables the PAFV insight that "any axis maps to any plane" as a direct manipulation interaction, not just a menu option.

**Behavior:**

- Drag column header → row header area triggers grid reflow from insertion point
- MiniNav serves as staging area for axes being repositioned
- Visual cues during drag: ghost header, insertion indicator, drop zone highlighting
- Grid reflows with D3 transition after drop
- Axis assignments update PAFV state (propagates to FilterNav and all consumers)

**Decisions (from SuperGrid Q&A):**

- Every Card needs a selection checkbox for drag/drop disambiguation
- Z-axis selection filter mechanism determines which cards are under consideration
- Insertion point reflows cell grid (like Numbers column drag behavior)
- MiniNav doubles as axis staging area

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Transpose 2D grid | Drag x-axis header to y-axis | Rows become columns, columns become rows | Data integrity maintained, all cells accounted for |
| Add 3rd axis | Drag Category axis from FilterNav to row headers | New nesting level appears with SuperStack spans | Correct grouping, no data loss |
| Remove axis | Drag axis header to MiniNav staging | Axis removed from grid, data aggregated | Grid collapses correctly |

| Test | Input | Expected | Pass Criteria |
| --- | --- | --- | --- |
| Cancel drag | Start drag, press Escape | Header returns to original position | No state change |
| Reflow animation | Complete axis drop | Grid reflows with D3 transition | Transition < 500ms, 60fps |

---

## 2.3 SuperSize — Cell & Header Sizing

**What:** Direct manipulation control over header cell sizing via drag handles, shift-drag for bulk resize, and lower-right corner for universal sizing.

**Why:** Different data densities require different cell proportions. Financial data needs wide columns; kanban-style data needs tall rows.

**Behavior:**

- Drag right edge of header cell to resize single column/row
- Shift+drag to resize all sibling cells at that level proportionally
- Lower-right corner drag for universal cell sizing (both axes)
- Minimum cell dimensions enforced (content must remain readable)
- Sizes persist per-dataset

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
| --- | --- | --- | --- |
| Single column resize | Drag right edge of "January" header | January column widens, others unchanged | Smooth drag, snap to pixel grid |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Bulk resize | Shift+drag any month header | All month-level headers resize equally | Proportional sizing maintained |
| Universal resize | Drag lower-right corner | All cells scale proportionally | Aspect ratio preserved if constrained |
| Minimum enforcement | Resize column to < 40px | Column stops at minimum width | Content remains legible |
| Persistence | Resize, navigate away, return | Saved sizes restored | Sizes match after reload |

---

## 2.4 SuperZoom — Cartographic Navigation

**What:** Zoom and pan controls that pin the upper-left corner (contrary to D3's default center-zoom behavior), following Apple Numbers-style behavior.

**Why:** D3's default zoom centers on cursor, which is disorienting for grid data. Users expect spreadsheet-like zoom where the anchor stays fixed.

**Behavior:**

- Pinch/scroll zoom anchors to upper-left corner of visible grid
- Does NOT follow D3.js native zoom behavior
- Cannot drag past table boundaries or drop past window edge (Numbers-style)
- Separate controls for value zoom (density) vs. extent pan
- Separate zoom slider and pan slider in distinct UI areas (starting point)

- Smooth animation for zoom transitions ("quiet app" aesthetic)
- All controls to be explored experimentally: separate controls, combined widget, contextual controls near headers, keyboard shortcuts — but separate controls are the v1 starting point

**Decisions (from phase-36 discussion):**

- Separate controls initially (combined widget later via experimentation)
- Smooth animation transitions (not instant swap, not fade)
- User toggle for sparse/dense display via SparsityDensity slider
- Fixed corner anchor (selection-based and user-definable to explore later)

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Zoom in | Pinch zoom on grid | Upper-left cell stays pinned, grid scales around it | A1 cell position unchanged |
| Zoom out | Reverse pinch | Grid shrinks toward upper-left | No content shifts past viewport edge |
| Pan right | Scroll/drag right | Grid scrolls, row headers stay visible (frozen) | Headers sticky, smooth 60fps |
| Pan past boundary | Try to scroll past last column | Elastic bounce-back or hard stop | Cannot overscroll |
| Zoom + pan combo | Zoom to 200%, pan to center of data | Upper-left anchor maintained throughout | Position stable across operations |

---

## 2.5 SuperDensitySparsity — Unified Aggregation Control

**What:** The Janus density model — a unified control for semantic zoom across LATCH dimensions. Density/sparsity is aggregation level, not just visual compactness.

**Why:** This is how Isometry applies Janus to LATCH: reducing dimensions along a semantic z-axis of precision/generalization without loss of accuracy.

**4-Level Density Model:**

| Level | Name | Control | Effect |
|---|---|---|---|
| 1 | Value Density | Per-facet zoom slider | Collapse hierarchy: Jan,Feb,Mar → Q1 |
| 2 | Extent Density | Extent pan slider | Hide/show empty rows and columns |
| 3 | View Density | View selector | Spreadsheet (1 card/row) ↔ Matrix (cards at intersections) |
| 4 | Region Density | Region config | Mix sparse + dense columns on shared axis |

**Key Insight:** Pan × Zoom are independent. All four quadrants are valid:

- Sparse extent + leaf values: Jan, Feb, Mar with all empty cells
- Populated only + collapsed: Q1 with only populated data
- Sparse extent + collapsed: Q1 with empty cells shown
- Populated only + leaf values: Jan, Feb, Mar without empty rows

**Behavior:**

- Maximum sparsity on Contacts → each contact visible in own row, Name facet auto-reveals as child header

- Maximum density on Projects → sparse stages collapse to dense stages (Capture/Backlog/TODO → "TO DO")
- Density changes are lossless — sparse view sees full precision, dense view sees aggregated truth
- Conflicting updates across density levels trigger conflict resolution (e.g., dense view moves card to Done, sparse view had it at Blocked → conflict)
- Aggregation rows/columns rendered explicitly (like spreadsheet SUM rows)

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Value density collapse | Slide Time density from Month → Quarter | Month headers merge into quarter spans | Data counts preserved (Q1 count = Jan + Feb + Mar) |
| Value density expand | Slide Time density from Quarter → Month | Quarter spans split into month children | Individual month data restored |
| Extent hide empty | Enable "populated only" | Empty intersections removed, grid compresses | Total visible cells = populated cell count |
| Extent show all | Disable "populated only" | Full Cartesian product visible with empty cells | Grid dimensions = x-count × y-count |
| Cross-density accuracy | Modify card at sparse level, | Dense aggregation reflects change | Consistency across all density views |

| Test | Input | Expected | Pass Criteria |
| --- | --- | --- | --- |
| | check dense level | | |
| Region mixing | Set Time to dense, Category to sparse | Time axis shows quarters, Category shows individual tags | Both density levels coexist correctly |

---

### 2.6 SuperSelect — Z-Axis Aware Selection

**What:** Card and SuperCard selection that understands the z-axis depth of the grid. Lasso select, click select, and checkbox select all need z-context.

**Why:** With stacked headers and overlapping z-layers (D3 sparsity layer, React density layer, overlay layer), selection needs to know which layer the user is targeting.

**Behavior:**

- Every Card has a small selection checkbox
- Z-axis selection filter mechanism determines active selection layer
- Click on data cell → select that card
- Click on parent header → select all children at that grouping level
- Lasso select respects current z-level (doesn't accidentally select headers when meaning to select data)
- Multi-select with modifier keys (Cmd+click, Shift+click for range)

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Single cell select | Click data cell | Cell highlighted, checkbox checked | Selection state updates in SelectionProvider |
| Header group select | Click "Q1" parent header | All Q1 child cells selected | Selection count = Q1 child count |
| Lasso data cells | Drag lasso across data area | Only data cells in region selected, not headers | No headers in selection array |
| Multi-select | Cmd+click three cells | All three selected independently | Selection array contains exactly 3 items |
| Range select | Click cell A, Shift+click cell B | All cells between A and B selected | Rectangular range selection |
| Z-layer filtering | Select cells while overlay is open | Overlay doesn't interfere with data selection | Correct z-targeting |

---

## 2.7 SuperPosition — Coordinate Tracking

**What:** Card position tracking system that enables Janus polymorphic view transitions. Each card maintains PAFV coordinates that survive view switches.

**Why:** When switching from SuperGrid to Kanban, cards need to know where they came from and where they should go. SuperPosition is the translation layer.

**Behavior:**

- Each card has logical PAFV coordinates (axis values, not pixel positions)
- View transitions recompute position based on new view's axis mappings (stateless approach)
- Custom sort orders within categories are tracked contextually
- Position state stored in SQLite alongside card data

**Decision (from SuperGrid Q&A):** Janus state is recomputed, not canonical. When switching views, positions are derived from the card's LATCH properties and the new view's axis assignments.

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
| --- | --- | --- | --- |
| Grid position | Card with status=Active, month=Jan | Card at intersection of Active column and Jan row | Position matches LATCH values |
| View transition | Switch Grid → Kanban | Card appears in "Active" kanban column | Card visible in correct column |
| Custom sort preserved | Reorder cards within a column, switch views and back | Original custom order restored | Sort order matches pre-transition |
| Position after filter | Apply filter that excludes card, remove filter | Card returns to original position | No position drift |

### 2.8 SuperCards — Generated Cards

**What:** SuperStack header cards, SuperAudit overlay cards, and aggregation cards are "generated" cards distinct from user data cards. They're rendered by D3 but aren't persisted as nodes in SQLite.

**Why:** Headers and overlays participate in the grid layout and can be selected/resized, but they're computed from metadata, not user data. Clear separation prevents confusion.

**Behavior:**

- SuperCards have a distinct visual style (muted/chrome appearance vs. data card appearance)
- Header SuperCards are the primary interactive surface for axis navigation
- Aggregation SuperCards (SUM, COUNT, AVG rows/columns) appear at density boundaries
- SuperCards don't appear in search results or node counts

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Header renders as SuperCard | Grid with 2 axes | Headers visually distinct from data cells | Different styling applied |
| Aggregation row | Enable "Show totals" | SUM row appears at bottom | Values = sum of column data |
| SuperCard not in search | Search for header text | Header not in FTS5 results | Result count excludes SuperCards |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| SuperCard selection | Click header SuperCard | Triggers header behavior, not card selection | Correct event handling |

---

### 2.9 SuperCalc — Category-Aware Calculations

**What:** HyperFormula integration for spreadsheet-style calculations that understand PAFV context. Formulas are scope-aware — `=SUM(Q1)` aggregates down the hierarchy axis.

**Why:** Multidimensional formulas have ambiguous scope. SuperCalc makes scope explicit through PAFV awareness.

**Behavior:**

- HyperFormula engine provides Excel-compatible formula evaluation
- Formula scope is PAFV-aware: `=SUM(Q1)` sums down the time hierarchy (Jan+Feb+Mar)
- Cross-axis formulas: `=SUM(Engineering:*)` sums across the category axis
- Calculated cells highlighted via SuperAudit styling
- Formula bar appears in React chrome, results render in D3 data plane

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Hierarchy sum | `=SUM(Q1)` in yearly row | Q1 value = Jan + Feb + Mar | Mathematically correct |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Cross-axis sum | `=SUM(Engineering:*)` | Sum of all Engineering values across time | Correct aggregation |
| Formula update | Change Jan value | Q1 sum automatically recalculates | Reactive update < 100ms |
| Circular reference | A1=B1, B1=A1 | Error indicator, no infinite loop | Graceful error handling |
| Density-aware calc | Collapse to Q1 density, formula still correct | Aggregation math unchanged | Same result at any density |

---

### 2.10 SuperAudit — Computed Value Highlighting

**What:** Visual distinction between raw data values, computed/enriched values, and CRUD operation indicators.

**Why:** Users need to know at a glance which cells contain their data vs. which contain calculated or system-generated values.

**Behavior:**

- Raw data cells: standard appearance
- Computed cells (formulas, aggregations): distinct highlight (e.g., subtle background tint)
- Enriched cells (ETL-derived data): different indicator
- Recent CRUD operations: brief flash/highlight on change

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
| --- | --- | --- | --- |
| Computed cell styling | Cell with formula | Visual indicator present | Visually distinct from raw cells |
| Raw cell styling | Cell with user-entered data | No audit indicator | Clean default appearance |
| CRUD flash | Create new card | Brief highlight animation on new cell | Flash visible < 2s, then fades |

---

### 2.11 SuperTime — Smart Time Handling

**What:** Intelligent time series parsing, header generation, and non-contiguous time selection.

**Why:** Time is the most common LATCH axis and the hardest to get right. Automatic parsing of date formats, smart header generation (Year → Quarter → Month → Week → Day), and the ability to select non-contiguous time ranges (Q1 + Q3, skipping Q2) are essential.

**Behavior:**

- Auto-parse various date formats from imported data
- Generate time hierarchy headers automatically based on date range span
- Non-contiguous time selection (multi-select time ranges)
- Progressive disclosure applies to time headers (same as SuperStack — and per Michael's phase-36 answer, the same auto-grouping +

progressive disclosure principle applies to SuperTime as a LATCH filter navigation control)
- Smart defaults: if data spans 2 years, default to Quarter view; if 2 months, default to Day view

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Auto-parse ISO dates | Column of ISO 8601 dates | Correctly parsed into time hierarchy | Year/Quarter/Month/Day all available |
| Auto-parse mixed formats | "Jan 15, 2024" and "2024-01-15" | Both parsed to same canonical date | Consistent hierarchy position |
| Smart default level | Data spanning 18 months | Default to Quarter view | Appropriate density for range |
| Non-contiguous select | Select Q1 and Q3 | Q2 hidden, Q1 and Q3 visible | Data integrity maintained |

---

## 2.12 SuperSort — PAFV-Aware Sorting

**What:** Sort by header that respects PAFV axis context. Sorting within a grouped column sorts only that group's children.

**Behavior:**

- Click header to sort ascending, click again for descending, third click to clear
- Sort within parent group context (sort months within Q1 only)
- Multi-sort with priority ordering

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Sort ascending | Click "Priority" header | Rows sorted by priority low→high | Sort order correct |
| Sort within group | Sort months within Q1 | Only Q1 months reorder | Q2+ months unchanged |
| Multi-sort | Sort by Status then Priority | Primary sort by status, secondary by priority | Both sort levels applied |

---

## 2.13 SuperFilter — PAFV-Aware Filtering

**What:** Excel-style auto-filter dropdown on headers that compiles to SQL WHERE clauses through the LATCH filter pipeline.

**Behavior:**

- Click filter icon on any header → dropdown with unique values and checkboxes
- Filter compiles to `WHERE` clause via `compileFilters()` in FilterProvider
- Active filters shown as indicators on headers
- Clear filter per-column or clear all

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
| --- | --- | --- | --- |
| Filter single value | Select "Active" in Status filter | Only Active rows visible | SQL WHERE generated correctly |
| Filter multiple values | Select "Active" and "Pending" | Both statuses visible | OR clause in SQL |
| Clear filter | Click "Clear" on Status filter | All statuses visible again | No WHERE clause for status |
| Filter indicator | Apply filter on Status column | Visual indicator on Status header | Indicator visible |

---

### 2.14 SuperSearch — Faceted Full-Text Search

**What:** FTS5-powered search across all card content, with PAFV-aware result highlighting in the grid.

**Behavior:**

- Search bar in React chrome
- FTS5 query with porter tokenizer and prefix matching
- Results highlighted in-situ within the grid (not a separate results list)
- Faceted filtering: search within a specific axis or across all

**Testing Criteria:**

| Test | Input | Expected | Pass Criteria |
| --- | --- | --- | --- |
| Basic search | Type "project alpha" | Matching cards highlighted in grid | FTS5 match returns results |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Prefix search | Type "proj" | Cards containing "project" highlighted | Prefix matching works |
| No results | Type "xyznonexistent" | No highlights, "no results" indicator | Graceful empty state |
| Performance | Search across 10k cards | Results appear in < 100ms | FTS5 index performant |

---

# 3. The Four-Quadrant Layout

```
┌───────────┬──────────────────────────────────────┐
│  MiniNav  │         Column Headers               │
│  (staging │  [Axis B₁] [Axis B₂] [Axis B₃]    │
│   area)   │  SuperStack spans                    │
├───────────┼──────────────────────────────────────┤
│   Row     │                                      │
│  Headers  │          Data Cells                  │
│ [Axis A₁] │    D3.js rendered grid               │
│ [Axis A₂] │    sql.js → D3 data binding          │
│ [Axis A₃] │                                      │
└───────────┴──────────────────────────────────────┘
```

**Quadrant responsibilities:**

- **MiniNav (upper-left):** Axis staging area, overall grid controls, intersection of row/column axis labels

- **Column Headers (upper-right):** SuperStack column axes, SuperDynamic drop target, SuperFilter dropdowns
- **Row Headers (lower-left):** SuperStack row axes, SuperDynamic drop target, SuperSize drag handles
- **Data Cells (lower-right):** D3.js rendered cards, SuperSelect lasso area, SuperCalc formula results

---

## 4. State Flow

```
User Action
    │
    ├── FilterNav change ──→ FilterProvider ──→
compileFilters() ──→ SQL WHERE
    ├── Axis drag ───────────→ PAFVProvider ───→ axis
remapping ───→ D3 re-layout
    ├── Density slider ─────→ DensityState ──→ hierarchy
collapse → SQL GROUP BY + D3 re-render
    ├── Cell edit ───────────→ sql.js UPDATE ─→ D3
enter/update/exit
    └── View switch ─────────→ PAFVProvider ──→
SuperPosition recompute → new renderer
```

**Key constraint:** D3's enter/update/exit IS state management. No Redux, no Zustand. React controls dispatch intents; D3 executes them.

---

## 5. View Transition State Machine

View transitions use a three-tier state model that distinguishes data-level state (always persists), view-family state (persists within LATCH or GRAPH families), and ephemeral rendering state (always resets).

### Tier 1: Always Persists (survives any transition)

Properties of the *data*, not the *view*:

| State | Provider | Rationale |
|---|---|---|
| Active LATCH filters (WHERE clauses) | FilterProvider | Filters are about the dataset, not the layout |
| Card selection | SelectionProvider | Selected cards stay selected across views |
| Search query (FTS5 active search) | SearchState | Search results stay highlighted |
| Density level (Value + Extent) | DensityProvider | Semantic zoom level is a data perspective |
| Sort order (primary + secondary) | SortState | Sort preference is about data ordering |

### Tier 2: Persists Within Family, Suspends Across Families

**LATCH family** (Grid ↔ List ↔ Kanban ↔ Calendar):

| State | Transfer Behavior |
|---|---|
| Axis-to-plane assignments | Transfers where sensible (Grid x-axis → Kanban column), recomputes where not |

| State | Transfer Behavior |
|---|---|
| Scroll/viewport position | SuperPosition translates logical coordinates (looking at Q1/Engineering → Kanban scrolls to Engineering column with Q1 cards) |
| SuperStack header expand/collapse | Suspends when target view lacks headers (e.g., Grid → List), restores on return |
| Custom sort order within groups | Persists — if you reordered cards within a Kanban column, that order survives Grid ↔ Kanban round-trips |
| `lastActiveView` | Stored so returning to family restores exact previous state |

**GRAPH family** (Network ↔ Tree ↔ Sankey):

| State | Transfer Behavior |
|---|---|
| Focus node (centered node) | Transfers directly between graph views |
| Traversal depth (hop count) | Transfers directly |
| Edge type filter | Transfers directly |
| `lastActiveView` | Stored so returning to family restores exact previous state |

**Cross-family transitions** (LATCH → GRAPH or GRAPH → LATCH):

- LATCH axis assignments **suspend** (not reset) when entering GRAPH family
- GRAPH focus node **suspends** when entering LATCH family
- Returning to a family restores its last Tier 2 state via `lastActiveView`

### Tier 3: Always Resets

| State | Rationale |
|---|---|
| Pixel-level positions (cell coords, force sim positions) | Rendering artifacts, recomputed from logical state |
| Hover/tooltip state | Ephemeral UI |
| Drag-in-progress state | Cancels on transition |
| Animation state | Completes or cancels |
| Open context menus, dropdown filters mid-selection | Temporary UI |

### State Machine Diagram

```
┌─────────────────────────────────────────────────────────┐
│                    Tier 1 (Global)
│
│  filters, selection, search, density, sort
│
│  ───────────────── always persists ────────────────────
│
└─────────────────────────────────────────────────────────┘


┌──────────────────────────┐   ┌──────────────────────────┐
│    LATCH Family           │  │     GRAPH Family
│
│                          │   │                          │
│  Grid ←——→ List          │   │  Network ←——→ Tree       │
│   ↕           ↕          │   │     ↕            ↕       │
│  Kanban ←→ Calendar      │   │  Sankey ←——→ (future)    │
```

```
│                          │   │                           │
│  Shared Tier 2:          │   │  Shared Tier 2:           │
│  • axis assignments      │   │  • focus node             │
│  • header states         │   │  • traversal depth        │
│  • viewport position     │   │  • edge type filter       │
│  • custom sort order     │   │                           │
└────────────┬─────────────┘   └──────────┬────────────────┘
             │                            │
             │   Cross-family transition   │
             │  Tier 2 suspends/restores   │
             └─────────────┬──────────────┘
                           │
              ┌────────────┴──────────┐
              │  Tier 3 (Ephemeral)   │
              │  Always resets        │
              │  pixel pos, hover,    │
              │  drag, animation      │
              └───────────────────────┘
```

## Implementation Types

```
// Tier 2 — stored per-family, persisted to SQLite
interface LATCHViewState {
  axisAssignments: Map<LATCHAxis, Plane>;
  headerStates: Map<string, ExpandCollapseState>;
  viewportAnchor: PAFVCoordinate;  // logical position, not pixels
  customSortOrders: Map<string, string[]>;  // groupKey → ordered nodeIds
  lastActiveView: 'grid' | 'list' | 'kanban' | 'calendar';
```

```
}

interface GRAPHViewState {
  focusNodeId: string;
  traversalDepth: number;
  edgeTypeFilter: EdgeType[];
  lastActiveView: 'network' | 'tree' | 'sankey';
}
```

### SQLite Persistence

```
CREATE TABLE view_state (
    id TEXT PRIMARY KEY,
    dataset_id TEXT NOT NULL,
    app_id TEXT NOT NULL,
    family TEXT NOT NULL,  -- 'LATCH' or 'GRAPH'
    state_json TEXT NOT NULL,  -- serialized Tier 2 state
    updated_at TEXT NOT NULL DEFAULT (strftime('%Y-%m-
%dT%H:%M:%SZ', 'now')),
    UNIQUE(dataset_id, app_id, family)
);
```

### Edge Case: Cross-View Data Mutations

When a user moves a card between Kanban columns, that's a Category axis value change (e.g., status: "Todo" → "In Progress"). This is a **data mutation**, not a view state change — it writes to SQLite via normal CRUD. Grid will show the updated status automatically because both views read from the same sql.js data. No special state transfer needed. The database is the source of truth; views just project it.

## View Transition Testing Criteria

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Filter survives Grid → Kanban | Filter to "Active", switch view | Kanban shows only Active cards | Filter persists (Tier 1) |
| Filter survives Grid → Network | Filter to "Active", switch family | Network shows only Active nodes | Filter persists (Tier 1) |
| Axis state restores in family | Grid with custom axes → List → Grid | Grid axes match pre-transition | Tier 2 LATCHViewState restored |
| Axis state suspends cross-family | Grid → Network → Grid | Grid axes match pre-Network state | Tier 2 suspend/restore works |
| Header collapse restores | Collapse Q1 in Grid → List → Grid | Q1 still collapsed | headerStates persisted |
| Selection survives transition | Select 3 cards in Grid, switch to Kanban | Same 3 cards selected | Tier 1 selection persists |
| Pixel state resets | Scroll to bottom of Grid, switch to Kanban | Kanban starts at logical anchor, not pixel offset | Tier 3 resets, Tier 2 anchor translates |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Kanban column move updates Grid | Move card to "Done" in Kanban, switch to Grid | Card in Done column in Grid | Data mutation via SQLite, not state transfer |
| Custom sort survives round-trip | Reorder cards in Grid column, switch to Kanban and back | Original custom order restored | customSortOrders in SQLite |
| Density survives any transition | Set Quarter density, switch Grid → Network → List | Quarter density still active | Tier 1 density persists |

---

## 6. Header Click Zone Hierarchy

SuperGrid has overlapping interactive surfaces at different z-depths. The governing principle is **geometric disambiguation** — the cursor's position within the cell geometry determines what gets hit. No modal switching required. This follows the same pattern as macOS Finder's disclosure triangles: the triangle is the parent's click target, the indented content belongs to children.

## The Geometric Rule

**Innermost interactive element wins, with a parent label exclusion zone.**

The parent header's text label and padding (~32px height for row headers, ~24px width for column headers) is exclusively the parent's click target. Everything below/beside that label within the visual span belongs to child elements. Users never need to understand z-layers — cursor changes and hover highlighting tell them what they'll hit.

## Click Target Zones

```
┌───────────────────────────────────────┐
│  Q1  ← Parent Label Zone (~32px)    │  Level 0: structural interaction
├───────────┬───────────┬────────────────┤
│ January   │ February │   March        │  Level 1: child header (data group selection)
│           │           │                │
│ [card]    │ [card]    │  [card]        │  Level 2: data cell (individual card)
│ [card]    │ [card]    │  [card]        │
│           │           │                │
│ [=SUM]    │ [=SUM]    │  [=SUM]        │  Level 3: SuperCard (generated/computed)
└───────────┴───────────┴────────────────┘
```

## Hit-Test Priority Order

When zones overlap, highest priority wins:

1. **Resize edge handles** (narrow 4px strips along cell borders) — always highest priority
2. **Parent header label zone** (top/left ~32px of spanning header)
3. **Child header cell body**
4. **Data cell body**
5. **SuperCard body** — lowest priority (only reached if no data cell occupies the position)

### Interaction Table

| Target | Single Click | Double Click | Right Click / Long Press | Dr |
|---|---|---|---|---|
| **Parent Header Label** (Level 0) | Toggle expand/collapse children | Expand/collapse entire branch | Context menu: Expand All, Collapse All, Sort within, Filter to, Hide | SuperD axis reposit |
| **Child Header Cell** (Level 1) | Select all data cells in this group | Edit header label (if user-defined category) | Context menu: Sort asc/desc, Filter, Rename, Hide | SuperS resize ( edge), SuperD reorder |
| **Data Cell** (Level 2) | Select card (checkbox toggle) | Open card detail/edit view | Context menu: Edit, Delete, Move to, Link to | Move c reposit suppor |

| Target | Single Click | Double Click | Right Click / Long Press | Dı |
|---|---|---|---|---|
| **SuperCard** (Level 3) | Show computation tooltip ("Sum of 12 values") | Open SuperCalc formula editor | Context menu: Edit formula, Change aggregation type, Remove | Not dra |
| **Resize Edge** (any level) | — | Auto-fit to content width | — | Resize columr width |

### Visual Disambiguation Cues

Users don't need to understand z-layers if the cursor and hover feedback tell them what they'll hit:

- **Cursor changes** on zone boundary crossing (see cursor column above) — immediate, no delay
- **Hover highlighting:** Entering a parent header's label zone highlights the entire span boundary with a subtle outline. Moving to a child header within that span shifts the highlight to just that child's cell. This gives clear visual preview of click scope.
- **Selection preview:** On hover, a brief visual indication of what *would* be selected (e.g., all January cells light up when hovering the January child header)

- **Resize affordance:** Resize edges show a thin line indicator on hover before drag begins

### Design Rationale

This geometric approach was chosen over alternatives:

- **Different click zones** (left/right/center within a single header) — rejected because invisible hotspots require user memorization
- **Modal selection** (toggle between "structure mode" and "data mode") — rejected because mode switches are disorienting and easy to forget
- **Hover expansion** (hover to show deeper levels, click to pin) — deferred as it adds latency and complexity; may revisit post-MVP

The geometric rule means parent headers naturally have less clickable area as nesting gets deeper (their label zone stays ~32px while the span grows), which is actually correct — deeper parents are structural and interact less often.

### Header Click Zone Testing Criteria

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Parent label click | Click "Q1" text in parent header | Q1 children expand/collapse | Toggle state changes, animation plays |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Parent double-click | Double-click "Q1" label | Entire Q1 branch expands/collapses recursively | All nested levels affected |
| Child header click | Click "January" in child header | All January data cells selected | Selection count matches January cell count |
| Child header double-click | Double-click "January" | Inline rename editor appears (if user-defined) | Editor appears with current text selected |
| Data cell click | Click cell at Jan/Engineering | Single card selected | Exactly 1 item in selection array |
| Data cell double-click | Double-click data cell | Card detail/edit view opens | Detail panel opens with correct card data |
| SuperCard click | Click SUM aggregation cell | Tooltip shows "Sum of N values" | No selection change, tooltip visible |
| SuperCard double-click | Double-click SUM cell | SuperCalc formula editor opens | Formula editor pre- |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| | | | populated with current formula |
| Resize edge priority | Hover on border between January and February | col-resize cursor appears | Cursor correct in 4px edge zone |
| Resize edge double-click | Double-click resize edge | Column auto-fits to content width | Width adjusts to fit longest content |
| Parent vs child disambiguation | Click in Q1 span area below Q1 label | Child header (not parent) receives click | Hit-test routes to innermost target |
| Cursor zone transitions | Move cursor: data → child header → parent label | Cursor changes: default → pointer → grab | Each transition immediate, no flicker |
| Hover highlight scope | Hover over Q1 label, then move to January | Highlight shifts: full Q1 span → January only | Smooth visual transition, no flash |
| Right-click parent | Right-click "Q1" label | Context menu: Expand All, Collapse All, Sort within, Filter to, Hide | All menu items present and functional |

| Test | Input | Expected | Pass Criteria |
|---|---|---|---|
| Right-click data cell | Right-click data cell | Context menu: Edit, Delete, Move to, Link to | All menu items present and functional |
| Right-click SuperCard | Right-click SUM cell | Context menu: Edit formula, Change aggregation, Remove | All menu items present and functional |

---

## 7. Performance Requirements

| Metric | Target | Measurement |
|---|---|---|
| Initial grid render (1k cards) | < 200ms | Performance.now() around render call |
| Grid render (10k cards) | < 500ms | Performance.now() around render call |
| View transition animation | < 300ms | D3 transition duration |
| FTS5 search (10k cards) | < 100ms | Query execution time |
| Density slider response | < 100ms | Input → visual update |
| Axis drag-drop reflow | < 500ms | Drop → stable layout |

| Metric | Target | Measurement |
|---|---|---|
| Header click → visual response | < 50ms | Click → first visual change |
| Cursor change on zone crossing | < 16ms | Mousemove → cursor update (1 frame) |
| Frame rate during interaction | 60fps | requestAnimationFrame measurement |
| Memory (10k cards loaded) | < 100MB | Browser DevTools heap snapshot |

---

## 8. Implementation Sequence

### Phase 35: PAFV Grid Core (Current Target)

1. **SuperPosition** — Coordinate tracking for all future features
2. **SuperDensitySparsity** — Unified data structure supporting density spectrum
3. **SuperDynamic** — Drag-and-drop axis repositioning
4. **SuperSize** — Cell expansion and resize
5. **SuperSelect** — Z-axis aware selection

### Phase 36: SuperGrid Headers

6. **SuperStack** — Nested PAFV headers with hierarchical spanning
7. **SuperZoom** — Cartographic navigation with pinned anchor
8. **Header Click Zones** — Geometric z-layer disambiguation

### Phase 37: Grid Continuum

9. **SuperCards** — Generated card distinction
10. **SuperSort / SuperFilter** — Header-based data operations
11. **SuperSearch** — FTS5 integration
12. **View Transition State Machine** — Three-tier state model implementation

### Post-MVP

13. **SuperCalc** + **SuperAudit** — Formula bar with audit highlighting
14. **SuperTime** + **SuperReplay** — Time series and animation
15. **SuperVersion / SuperTemplates** — Collaboration

---

## 9. Resolved Design Decisions

### From Phase 36 Discussion Template

| Question | Decision | Rationale |
|---|---|---|
| Maximum nesting depth | Dynamic/unlimited | Data complexity should determine depth, not arbitrary limits |
| Visual complexity management | Automatic grouping + progressive disclosure | Both mechanisms complement each other and apply to other LATCH controls like SuperTime |
| Grouping strategy | Semantic grouping → data density, user-configurable | Semantic grouping first (Year+Quarter+Month → "Time"), then data density (collapse levels |

| Question | Decision | Rationale |
|---|---|---|
| | | with low cardinality). User settings override defaults. **GitHub issue needed for user application settings in SQLite.** |
| Progressive disclosure navigation | Level picker tabs + zoom controls | Headers are the primary breadcrumb device (3D camera stairstepping down hierarchy). Explicit breadcrumb trail deferred. Hover expansion deferred (complicated). |
| Header click behavior | Geometric click zones (see Section 6) | Parent label zone for structural operations, child body for data selection, cursor feedback eliminates guesswork |
| Orthogonal control separation | Separate controls first, all approaches to explore | Start simple with distinct zoom/pan sliders, iterate toward combined/contextual/keyboard |
| Value zoom transition | Smooth animation | "Quiet app" aesthetic — morphing headers with easing transitions |
| Extent pan behavior | User toggle (SparsityDensity slider) | Users choose between sparse and dense display modes |
| Zoom anchor point | Fixed corner | Pin to top-left. Selection-based and user-definable to explore later. |

| Question | Decision | Rationale |
|---|---|---|
| Collapse animation style | Morphing boundaries | D3 transition-driven boundary reshaping |
| Performance during state changes | Progressive rendering, lazy rendering fallback | Show skeleton/placeholder while loading; if consistently exceeding performance budgets, virtualize off-screen headers |
| Multi-level operations | Context menu first | "Expand All" / "Collapse All" via right-click. Shift+click and double-click to experiment with later. |
| State persistence | Per-dataset and per-app | An Isometry app can combine multiple views and datasets; state scopes to that combination |

### Visual Spanning Layout

| Question | Decision | Rationale |
|---|---|---|
| Span calculation method | Hybrid with minimum widths | Data-proportional for primary sizing (columns with more data get more space), content-based minimums prevent illegibility (header text must fit), equal distribution as fallback when data counts are uniform. This mirrors how Numbers and modern pivot tables handle column sizing. |
| Alignment strategy | Content-aware | Center for short spans (1-3 words like "Q1", "Jan"), left-align for long spans (multi-word categories like "Engineering |

| Question | Decision | Rationale |
|---|---|---|
| | | & Design"). Numeric content right-aligns regardless of span length. Dates left-align. This follows the principle that alignment should serve scannability, not uniformity. |
| Layout conflict resolution | Dynamic reflow with scroll fallback | Automatically adjust layout to prevent conflicts while respecting minimum widths. When reflow can't resolve (screen too narrow for all visible axes), fall back to horizontal scroll rather than truncation. Priority-based was rejected because rigid hierarchies break with user customization; compression was rejected because it sacrifices legibility. |
| Responsive behavior | Breakpoint adaptation with semantic grouping | On smaller screens, reduce visible header depth (e.g., collapse to 2 levels on tablet, 1 on phone) using the same semantic grouping mechanism from auto-grouping. Proportional scaling alone makes headers illegible on mobile. The semantic grouping contract ensures collapsed views remain meaningful. |

## View Transitions

| Question | Decision | Rationale |
|---|---|---|
| State machine | Three-tier: Global persists, Family | Matches user mental model: "I filtered to Active tasks" |

| Question | Decision | Rationale |
|---|---|---|
| model | suspends/restores, Ephemeral resets | (global), "I was looking at the Engineering column" (family-specific), pixel positions are rendering details |
| Custom sort persistence | Per-view-family sort order stored in SQLite `view_state` table | Custom reordering within a Kanban column is LATCH-family state; it should survive Grid ↔ Kanban transitions but need not transfer to Network view |
| Cross-view data mutations | Database is source of truth, no special state transfer | Moving a Kanban card writes to SQLite. Grid reads from SQLite. Boring stack wins. |

### Header Click Zones

| Question | Decision | Rationale |
|---|---|---|
| Click disambiguation | Geometric zones with innermost-wins + parent label exclusion | Avoids invisible hotspots. Users see exactly what they'll click. Parent label zone (~32px) is explicit and predictable. |
| Cursor feedback | Zone-specific cursor changes on hover | Visual affordance eliminates guesswork about click targets without requiring user understanding of z-layers |

| Question | Decision | Rationale |
|---|---|---|
| Parent click behavior | Single-click: expand/collapse (structural). Double-click: expand/collapse entire branch. | Parents control grid shape. Child clicks control data selection. Clean separation of concerns. |
| Child header click behavior | Single-click: select group. Double-click: inline rename (if user-defined). | Consistent with spreadsheet conventions. |
| Data cell click behavior | Single-click: select. Double-click: open detail/edit. | Consistent with every grid/table UI convention. |
| SuperCard click behavior | Single-click: tooltip (informational). Double-click: formula editor. | SuperCards aren't user data — clicking them shouldn't select anything. Double-click opens editor for power users. |
| Resize edge behavior | Drag: resize. Double-click: auto-fit to content. | Matches Excel/Numbers convention exactly. |

---

## 10. Open Questions for Claude Code

> These are implementation-specific questions to resolve during development.

### SuperCalc Scope

- [ ] **Formula reference syntax:** How does the user express PAFV-scoped formulas? Need to define syntax that distinguishes hierarchy aggregation from cross-axis aggregation. Consider `=SUM(Time:Q1)` for hierarchy and `=SUM(Category:Engineering, *)` for cross-axis.
- [ ] **Quantrix-style formula model:** Review https://help.idbs.com/Quantrix/Modeler_Help/LATEST/en/working-with-formulae.html for dimensional formula patterns.

### Progressive Rendering

- [ ] **Virtualization strategy:** For grids with 10k+ cells, which virtualization approach? Options: D3 viewport culling (only render visible cells + buffer), or canvas rendering for the data plane with SVG only for headers.
- [ ] **Header virtualization:** When SuperStack has 100+ parent headers, do we virtualize the header column too? Or is header count always small enough to render fully?

### Density Conflict Resolution

- [ ] **Cross-density conflict UX:** When a dense-view user and sparse-view user make conflicting edits (dense moves to "Done", sparse had it at "Blocked"), what does the conflict resolution modal look like? Show both density contexts side-by-side?

### User Application Settings

- [ ] **Settings storage:** GitHub issue needed for user application settings stored in SQLite (referenced in phase-36 grouping strategy answer). Settings table already exists in schema ( `settings` table in

SQLITE-MIGRATION-PLAN-v2.md) but needs expansion for per-user grouping thresholds, complexity preferences, and density defaults.

---

## 11. Acceptance Criteria (MVP Gate)

A SuperGrid MVP is shippable when:

- [ ] 2D grid renders with correct PAFV axis mapping from sql.js data
- [ ] At least 2-level SuperStack headers render with correct visual spanning
- [ ] Density slider collapses/expands one axis level (Value Density level 1)
- [ ] Extent slider hides/shows empty intersections (Extent Density level 2)
- [ ] Axis drag-and-drop transposes rows ↔ columns
- [ ] Cell selection works with single-click and Cmd+click multi-select
- [ ] Header click zones follow the geometric rule (parent label vs. child vs. data)
- [ ] Cursor changes correctly across all zone boundaries
- [ ] Column resize with drag handle persists
- [ ] Zoom pins upper-left corner
- [ ] FTS5 search highlights matching cells in-grid
- [ ] View transitions preserve Tier 1 state (filters, selection, search, density)
- [ ] View transitions within LATCH family preserve/restore Tier 2 state
- [ ] All operations maintain 60fps with 1,000 cards
- [ ] All operations complete within performance targets at 10,000 cards

---

# 12. Testing Strategy

## Unit Tests (Vitest)

- Header span calculation algorithms (hybrid sizing with minimums)
- PAFV coordinate ↔ pixel position mapping
- Density hierarchy collapse/expand logic
- Filter compilation (LATCH → SQL WHERE)
- SuperPosition coordinate transforms across view types
- Hit-test zone calculations for header click disambiguation
- View state serialization/deserialization for Tier 2 persistence
- Responsive breakpoint calculations for header depth adaptation

## Integration Tests

- sql.js query → D3 render pipeline (data appears correctly)
- FilterProvider state change → grid re-render (filter actually filters)
- Axis drag-drop → PAFVProvider update → grid reflow (transpose works end-to-end)
- Density slider → SQL GROUP BY → D3 re-render (aggregation is correct)
- View transition round-trips (Grid → Kanban → Grid state preservation)
- Header click → correct handler dispatched (parent expand vs. child select vs. data select)
- Cursor feedback across zone boundaries in nested header scenarios

## Visual Regression Tests

- Screenshot comparison for header spanning at various depths
- Grid layout at different density levels

- Responsive behavior at multiple viewport sizes (desktop, tablet, phone breakpoints)
- Cursor changes across click zone boundaries
- Hover highlighting transitions (parent span → child cell)

### Performance Tests

- Render benchmark at 1k, 5k, 10k card counts
- FTS5 search latency measurement
- View transition latency (including state serialize/restore)
- Hit-test calculation speed (must be < 16ms for 60fps mousemove)
- Memory usage monitoring during long sessions

---

## 13. Rollback Plan

SuperGrid features are additive layers on the base 2D grid. Rollback strategy:

1. **SuperStack:** Fall back to flat headers (single row/column headers)
2. **SuperDensity:** Fall back to showing all data ungrouped
3. **SuperDynamic:** Disable drag-drop, use menu-based axis assignment
4. **SuperZoom:** Fall back to D3 default zoom behavior
5. **SuperCalc:** Disable formula evaluation, show raw data only
6. **Header Click Zones:** Fall back to simple click-to-select everywhere (all clicks → select)
7. **View Transitions:** Fall back to full state reset on every transition (lose Tier 2 persistence)

Each Super* feature can be independently disabled via feature flags without affecting the base grid rendering.

---

*This specification synthesizes decisions from the phase-36 discussion template, SuperGrid Q&A, SUPER-FEATURES catalog, architecture truth documents, and prior Claude conversations. All phase-36 questions are now resolved in Section 9. Remaining implementation-specific questions are in Section 10 for resolution during development.*