# 04_CODING_STANDARDS.md

**Architecture Decisions & Coding Standards**
**BAT Integration Project**

---

**Consolidates:** CODING_SYSTEM_INTEGRATION_SUMMARY.md, CODING_SYSTEM_QUICK_REF.md, LEARNING_FIRST_BAT_SYSTEM.md, Decision Templates
**Created:** November 10, 2025
**Last Updated:** November 10, 2025
**Version:** 2.0
**Status:** Active - Architecture Reference

---

## 🎯 EXECUTIVE SUMMARY

### Purpose
Document the architectural decisions and coding standards that form the foundation for the BAT integration project. These decisions, made in Week 1, determine the database structure, import strategy, and system scalability for all 12 weeks.

### Why This Matters
**Without these decisions:** Import chaos, inconsistent data, 4-6 weeks of rework
**With these decisions:** Clean architecture, smooth imports, scalable system

### The Three Critical Decisions
1. **Plan-Pack Relationship** - How materials relate to plans and packs
2. **Plan-Elevation Model** - How to handle the triple-encoding problem
3. **Internal Option Codes** - Standard format for tracking options

### Current Status
- **Monday Analysis:** ☑ Complete (746 items analyzed, 45 pages documentation)
- **Tuesday Decisions:** ⏳ To be made (6 hours scheduled)
- **Architecture Lock:** Friday after team validation

---

## 🚨 THE PROBLEM

### Issue #1: Triple-Encoding of Elevation Data

**Current State in Richmond BAT:**
```
Pack ID:              |10.82BCD OPT DEN FOUNDATION
Location Column:      "- ELVB - ELVC - ELVD"
Option Codes:         ELVB, ELVC, ELVD
```

SAME INFORMATION ENCODED 3 DIFFERENT WAYS! ❌

```

**Why This Is a Problem:**
- Which is the source of truth?
- What happens when they conflict?
- How do you query by elevation?
- Import logic becomes complex
- Maintenance nightmare

**Impact on Project:**
- Can't reliably determine which elevations a pack applies to
- Query: "Show all den options for Elevation B" becomes impossible
- Different import batches might interpret differently
- Data integrity issues

---

### Issue #2: Plan-Pack Relationship Unclear

**The Question:**
When pack "12.x5" (2-car garage 5' extension) appears on plans G603 and G914, are the materials identical or different?

**Current State:**
```
Unknown! Could be either:
- Universal Pack: Same materials on any plan
- Plan-Specific Pack: Different materials per plan
```

**Why This Matters:**
```sql
-- If Universal Pack:
SELECT * FROM materials
WHERE pack_id = '12.x5'  -- Returns ONE set of materials

-- If Plan-Specific:
SELECT * FROM materials
WHERE plan_id = 'G603' AND pack_id = '12.x5'  -- Returns plan-specific materials
```

**Impact on Database Design:**
- Determines primary keys
- Affects table relationships
- Changes query patterns
- Influences storage size

---

### Issue #3: Elevation Model Undefined
```

**The Question:**
Is "G603B" one thing or two dimensions?

**Option A: Elevation as Variant**
```
"G603B" is THE plan (single identity)
Database: plan_id = "G603B"
Customer says: "I want G603B"
```

**Option B: Elevation as Dimension**
```
G603 = plan, B = elevation (separable)
Database: plan_id = "G603", elevation = "B"
Customer says: "I want G603, elevation B"
```

**Why This Matters:**
- Determines how you query across elevations
- Affects sheet naming conventions
- Influences how triple-encoding gets solved
- Changes import logic

---

### Issue #4: No Internal Option Code Standard

**Current Reality:**
```
Richmond uses: XGREAT, 2CAR5XA, FPSING01 (descriptive)
Holt uses: 167010100, 164910105 (numeric hierarchical)
Internal standard: ??? (undefined)
```

**Why This Matters:**
- Need consistent way to track options internally
- Must translate between Richmond ⇔ Holt
- Future Manor Homes integration needs standard
- User interface depends on this

---

## 📋 DECISION FRAMEWORK

### How to Make Each Decision

Every decision document must include:

1. **The Question** - What are we deciding?

2. **The Options** - What choices do we have?
3. **Pros & Cons** - Honest assessment of each path
4. **Testing** - Validate with real data
5. **Recommendation** - What we choose
6. **Rationale** - Why we chose it
7. **Examples** - Show it working
8. **Edge Cases** - What might break it

### Testing Methodology

**For each decision:**
1. Get real data from both BATs
2. Test each option with actual examples
3. Write sample queries
4. Measure complexity
5. Consider edge cases
6. Get team input
7. Document findings

---

## 🔧 DECISION 1: Plan-Pack Relationship

### The Question
Can pack "12.x5" (2-car garage 5' extension) work on multiple plans with the same materials (Universal Pack), or does each plan have its own version (Plan-Specific Pack)?

### The Options

#### Option A: Universal Packs
```sql
-- Database Model
CREATE TABLE packs (
    pack_id TEXT PRIMARY KEY,        -- '12.x5', '10.82'
    pack_name TEXT NOT NULL,
    description TEXT,
    is_active INTEGER DEFAULT 1
);

CREATE TABLE materials (
    material_id INTEGER PRIMARY KEY,
    pack_id TEXT NOT NULL,           -- Links to universal pack
    item_id TEXT NOT NULL,
    quantity REAL NOT NULL,
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id)
);

CREATE TABLE plan_packs (
    plan_pack_id INTEGER PRIMARY KEY,
```

```
    plan_id TEXT NOT NULL,
    pack_id TEXT NOT NULL,           -- Links plan to universal pack
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id)
);

-- Query: Show all plans using pack 12.x5
SELECT p.plan_id, p.plan_name
FROM plans p
JOIN plan_packs pp ON p.plan_id = pp.plan_id
WHERE pp.pack_id = '12.x5';
```

**Pros:**
- ☑ Simpler data model
- ☑ Less data duplication
- ☑ Easier to update pack globally
- ☑ Clear when packs are truly identical
- ☑ Smaller database size
- ☑ Maintenance efficient

**Cons:**
- ❌ Can't handle plan-specific variations
- ❌ Less flexible for customization
- ❌ May not match reality if packs differ by plan
- ❌ Requires override mechanism if exceptions exist

---

#### Option B: Plan-Specific Packs
```sql
-- Database Model
CREATE TABLE packs (
    pack_id INTEGER PRIMARY KEY,     -- Auto-increment
    plan_id TEXT NOT NULL,           -- Part of identity
    pack_code TEXT NOT NULL,         -- '12.x5', '10.82'
    pack_name TEXT NOT NULL,
    description TEXT,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    UNIQUE(plan_id, pack_code)
);

CREATE TABLE materials (
    material_id INTEGER PRIMARY KEY,
    pack_id INTEGER NOT NULL,        -- Links to plan-specific pack
    item_id TEXT NOT NULL,
    quantity REAL NOT NULL,
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id)
);
```

```sql
-- Query: Show pack 12.x5 for plan G603
SELECT * FROM packs
WHERE plan_id = 'G603' AND pack_code = '12.x5';
```

**Pros:**
- ☑ Handles all variations
- ☑ Maximum flexibility
- ☑ Matches physical reality if packs differ
- ☑ Can customize per plan easily
- ☑ No override logic needed

**Cons:**
- ❌ More complex data model
- ❌ More data duplication
- ❌ Harder to maintain consistency
- ❌ Larger database size
- ❌ Updates require touching multiple records

---

#### Option C: Hybrid Approach (RECOMMENDED)
```sql
-- Database Model
CREATE TABLE packs (
    pack_id TEXT PRIMARY KEY,        -- '12.x5', '10.82'
    pack_name TEXT NOT NULL,
    is_universal INTEGER DEFAULT 0,  -- Flag: universal or not
    description TEXT,
    is_active INTEGER DEFAULT 1
);

CREATE TABLE plan_packs (
    plan_pack_id INTEGER PRIMARY KEY,
    plan_id TEXT NOT NULL,
    pack_id TEXT NOT NULL,
    override_materials INTEGER DEFAULT 0,  -- This plan has custom materials?
    notes TEXT,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id),
    UNIQUE(plan_id, pack_id)
);

CREATE TABLE materials (
    material_id INTEGER PRIMARY KEY,
    plan_pack_id INTEGER NOT NULL,  -- Links to specific plan-pack combo
    item_id TEXT NOT NULL,
    quantity REAL NOT NULL,
    unit_cost REAL,
    FOREIGN KEY (plan_pack_id) REFERENCES plan_packs(plan_pack_id)
```

```
);

-- Logic:
-- IF pack.is_universal = TRUE AND plan_packs.override_materials = FALSE:
--    All plans share same materials
-- ELSE:
--    Materials are plan-specific

-- Query: Show all universal packs
SELECT * FROM packs WHERE is_universal = 1;

-- Query: Show plans with customized pack 12.x5
SELECT p.plan_id, pp.notes
FROM plans p
JOIN plan_packs pp ON p.plan_id = pp.plan_id
WHERE pp.pack_id = '12.x5' AND pp.override_materials = 1;
```

**Pros:**
- ☑ Best of both worlds
- ☑ Universal packs where appropriate
- ☑ Plan-specific overrides when needed
- ☑ Flexibility for future
- ☑ Matches reality (some universal, some custom)
- ☑ Clear flags indicate intent
- ☑ Scalable for Manor Homes

**Cons:**
- ❌ Slightly more complex than pure approaches
- ❌ Requires clear rules for universal vs specific
- ❌ Two-step logic for queries

---

### Testing Procedure

**Test 1: Compare Materials (30 min)**
```
1. Pick pack "12.x5" (2-car garage 5' extension)
2. Find it on plan G603 in Richmond BAT
3. Find it on plan G914 in Richmond BAT
4. Compare materials line by line:
   - Same item numbers?
   - Same quantities?
   - Same specifications?
5. Document findings

Results: [To be filled Tuesday]
```

**Test 2: Second Pack (15 min)**
```
1. Pick pack "10.82" (optional den foundation)
2. Find on 2-3 different plans
3. Compare materials
4. Is pattern consistent?

Results: [To be filled Tuesday]
```

**Test 3: Edge Cases (15 min)**
```
1. Find any packs that appear on only one plan
   (These are inherently plan-specific)
2. Find packs that appear on 5+ plans
   (Good candidates for universal)
3. Document ratio

Results: [To be filled Tuesday]
```

### Recommendation
**Hybrid Approach (Option C)** - To be confirmed after testing

**Rationale (Preliminary):**
- Most building projects have BOTH universal and custom components
- Foundation packs often universal (standard across plans)
- Interior/finish packs often customized (plan-specific)
- Hybrid model accommodates both without forcing everything into one pattern
- Clear flags (`is_universal`, `override_materials`) document intent
- Scalable for future builders (Manor Homes)

**Final Decision:** [To be filled Tuesday after testing]

---

## 🏗️ DECISION 2: Plan-Elevation Model

### The Question
Is "G603B" one plan (Elevation as Variant), or is it Plan G603 with Elevation B (Elevation as Dimension)? How do we solve the triple-encoding problem?

### The Problem Statement

**Current Triple-Encoding:**
```
Location 1: Pack name
|10.82BCD OPT DEN FOUNDATION
↓ "BCD" embedded in name
```

Location 2: Location column
"- ELVB - ELVC - ELVD"
↓ Elevation codes in description

Location 3: Option codes
ELVB, ELVC, ELVD
↓ Elevation in option codes

Question: Which is the source of truth?
Answer: All three must match or chaos ensues
```

### The Options

#### Option A: Elevation as Variant
```sql
-- Database Model
CREATE TABLE plans (
    plan_id TEXT PRIMARY KEY,        -- 'G603', 'G603B', 'G603C', 'G914A'
    base_plan TEXT,                  -- 'G603', 'G603', 'G603', 'G914'
    elevation TEXT,                  -- NULL, 'B', 'C', 'A'
    plan_name TEXT,
    builder_id TEXT NOT NULL,
    FOREIGN KEY (builder_id) REFERENCES builders(builder_id)
);

CREATE TABLE materials (
    material_id INTEGER PRIMARY KEY,
    plan_id TEXT NOT NULL,           -- 'G603B' (complete plan code)
    pack_id TEXT NOT NULL,
    item_id TEXT NOT NULL,
    quantity REAL NOT NULL,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id)
);

-- Query: Show all elevations of G603
SELECT * FROM plans WHERE base_plan = 'G603';

-- Query: Materials for G603B
SELECT * FROM materials WHERE plan_id = 'G603B';
```

**Pros:**
- ☑ Matches current sheet naming (LE93 G603B)
- ☑ Simple queries (plan_id is complete)
- ☑ Each elevation is distinct entity
- ☑ Easy to understand
- ☑ Natural fit for current data

**Cons:**
```

- ❌ Doesn't solve triple-encoding
- ❌ Still duplicates elevation in multiple places
- ❌ Hard to query "show all A elevations across plans"
- ❌ Elevation not reusable concept
- ❌ Pack elevation problem remains

**Triple-Encoding Status:** NOT SOLVED ❌

---

#### Option B: Elevation as Dimension (RECOMMENDED)
```sql
-- Database Model
CREATE TABLE plans (
    plan_id TEXT PRIMARY KEY,        -- 'G603', 'G914', '1670' (NO elevation)
    plan_name TEXT NOT NULL,
    builder_id TEXT NOT NULL,
    sq_ft INTEGER,
    is_active INTEGER DEFAULT 1,
    FOREIGN KEY (builder_id) REFERENCES builders(builder_id)
);

CREATE TABLE plan_elevations (
    elevation_id INTEGER PRIMARY KEY,
    plan_id TEXT NOT NULL,
    elevation_code TEXT NOT NULL,    -- 'A', 'B', 'C', 'D'
    elevation_name TEXT,              -- 'Elevation A', etc.
    is_active INTEGER DEFAULT 1,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    UNIQUE(plan_id, elevation_code)
);

CREATE TABLE packs (
    pack_id TEXT PRIMARY KEY,        -- '10.82' (NO elevation in name!)
    pack_name TEXT NOT NULL,         -- 'OPT DEN FOUNDATION' (clean)
    phase TEXT,
    is_active INTEGER DEFAULT 1
);

CREATE TABLE pack_elevations (
    pack_elevation_id INTEGER PRIMARY KEY,
    pack_id TEXT NOT NULL,
    elevation_code TEXT NOT NULL,    -- 'B', 'C', 'D' (which elevations apply)
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id),
    UNIQUE(pack_id, elevation_code)
);

CREATE TABLE materials (
    material_id INTEGER PRIMARY KEY,
    plan_id TEXT NOT NULL,           -- 'G603' (NOT 'G603B')
```

```
    elevation_id INTEGER,             -- Link to plan_elevations
    pack_id TEXT NOT NULL,
    item_id TEXT NOT NULL,
    quantity REAL NOT NULL,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    FOREIGN KEY (elevation_id) REFERENCES plan_elevations(elevation_id),
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id)
);

-- Query: Show all B elevations across all plans
SELECT p.plan_id, p.plan_name, pe.elevation_code
FROM plans p
JOIN plan_elevations pe ON p.plan_id = pe.plan_id
WHERE pe.elevation_code = 'B';

-- Query: Show packs for G603 Elevation B
SELECT pk.pack_id, pk.pack_name
FROM packs pk
JOIN pack_elevations pkv ON pk.pack_id = pkv.pack_id
JOIN plan_elevations pe ON pkv.elevation_code = pe.elevation_code
WHERE pe.plan_id = 'G603' AND pe.elevation_code = 'B';

-- Query: Materials for G603 Elevation B
SELECT m.*, i.item_description
FROM materials m
JOIN plan_elevations pe ON m.elevation_id = pe.elevation_id
JOIN items i ON m.item_id = i.item_id
WHERE pe.plan_id = 'G603' AND pe.elevation_code = 'B';
```

**Pros:**
- ☑ SOLVES triple-encoding (single source of truth!)
- ☑ Elevation stored once in dedicated table
- ☑ Can query across elevations easily
- ☑ Cleaner data model
- ☑ Easier to maintain
- ☑ Pack names clean (no elevation embedded)
- ☑ Scales better (add new elevations without changing pack names)

**Cons:**
- ❌ Requires more joins in queries
- ❌ Slightly more complex than variant approach
- ❌ Different from current sheet naming
- ❌ Need to explain dimension concept to team

**Triple-Encoding Status:** SOLVED ☑

**How It Solves Triple-Encoding:**
```
BEFORE (Triple-Encoded):
```

```
Pack name: |10.82BCD
Location: "ELVB, ELVC, ELVD"
Options: ELVB, ELVC, ELVD

AFTER (Single Source):
pack_id: 10.82 (clean, no elevation)
pack_elevations table:
  - 10.82 → B
  - 10.82 → C
  - 10.82 → D

ONE place to check which elevations a pack applies to!
```

---

### Testing Procedure

**Test 1: Current Usage (15 min)**
```
1. Ask William: How do customers select?
   - "I want G603B" (variant)?
   - "I want G603, elevation B" (dimension)?
2. Review customer contracts
3. Check quote format
4. Document findings

Results: [To be filled Tuesday]
```

**Test 2: Query Complexity (30 min)**
```
1. Write query for Option A (variant)
2. Write query for Option B (dimension)
3. Compare:
   - Which is clearer?
   - Which performs better?
   - Which handles edge cases?

Results: [To be filled Tuesday]
```

**Test 3: Triple-Encoding Fix (15 min)**
```
1. Can Option A solve triple-encoding? (No)
2. Can Option B solve triple-encoding? (Yes - how?)
3. Document solution clarity

Results: [To be filled Tuesday]
```

### Recommendation
**Elevation as Dimension (Option B)** - Strong recommendation

**Rationale:**
- **CRITICAL:** Solves triple-encoding problem completely
- Elevation is conceptually a dimension (attribute of plan, not identity)
- Enables powerful queries across elevations
- Cleaner pack names without embedded elevation codes
- More maintainable long-term
- Industry standard approach in database design
- Scales for future (Manor Homes, new builders)

**Migration Impact:**
- Sheet names change: "LE93 G603B" → "materialist_G603_B"
- Pack names cleaned: "|10.82BCD" → "|10.82"
- Import logic uses plan_elevations table
- Queries use joins (but cleaner logic)

**Final Decision:** [To be filled Tuesday after team input]

---

## 🔤 DECISION 3: Internal Option Codes

### The Question
Richmond uses descriptive codes (XGREAT, 2CAR5XA). Holt uses numeric codes (167010100). What should OUR internal standard be?

### Current State

**Richmond Codes:**
```
Format: Descriptive alphanumeric
Examples:
- XGREAT = Extended great room
- 2CAR5XA = 2-car garage, 5' extension, elevation A
- FPSING01 = Fireplace single, option 01
- DENOPTB = Den option, elevation B

Characteristics:
☑ Human-readable
☑ Self-documenting
☑ Easy to remember
❌ No structure
❌ Collisions possible
❌ Hard to validate format
```

**Holt Codes:**

```
Format: Numeric hierarchical
Pattern: [PLAN 4][PHASE 2][OPTION 2][ELEVATION 2]

Examples:
- 167010100 = Plan 1670, Phase 01, Option 01, Elevation A (100)
- 164910105 = Plan 1649, Phase 01, Option 01, Elevation A (100)
- 189010400 = Plan 1890, Phase 01, Option 04, All elevations

Elevation Encoding:
- XX100XX = Elevation A
- XX200XX = Elevation B
- XX300XX = Elevation C
- XX400XX = Elevation D

Characteristics:
☑ Systematic
☑ Plan embedded
☑ Phase grouping
☑ Easy to validate
❌ Not human-readable
❌ Long (9 digits)
❌ Requires decoder
```

### The Options

#### Option A: Keep Richmond Codes
```sql
-- Internal standard = Richmond format
CREATE TABLE options (
    option_code TEXT PRIMARY KEY,    -- 'XGREAT', '2CAR5XA'
    description TEXT NOT NULL,
    category TEXT,                    -- 'INTERIOR', 'GARAGE', 'STRUCTURAL'
    is_active INTEGER DEFAULT 1
);

CREATE TABLE option_translation (
    translation_id INTEGER PRIMARY KEY,
    richmond_code TEXT NOT NULL,      -- 'XGREAT'
    holt_code TEXT,                   -- '167010600'
    description TEXT,
    FOREIGN KEY (richmond_code) REFERENCES options(option_code)
);

-- Holt team needs translation table
```

**Pros:**
- ☑ Richmond team already knows these

- ☑ No retraining for Richmond
- ☑ Human-readable
- ☑ Self-documenting

**Cons:**
- ❌ Holt team needs translation
- ❌ No structure
- ❌ Hard to validate
- ❌ Collision risk

---

#### Option B: Keep Holt Codes
```sql
-- Internal standard = Holt format
CREATE TABLE options (
    option_code TEXT PRIMARY KEY,    -- '167010100'
    plan_portion TEXT,                -- '1670'
    phase_portion TEXT,               -- '01'
    option_portion TEXT,              -- '01'
    elevation_portion TEXT,           -- '00', '100', '200'
    description TEXT NOT NULL,
    is_active INTEGER DEFAULT 1
);

CREATE TABLE option_translation (
    translation_id INTEGER PRIMARY KEY,
    holt_code TEXT NOT NULL,          -- '167010600'
    richmond_code TEXT,               -- 'XGREAT'
    description TEXT,
    FOREIGN KEY (holt_code) REFERENCES options(option_code)
);

-- Richmond team needs translation table
```

**Pros:**
- ☑ Holt team already knows these
- ☑ No retraining for Holt
- ☑ Systematic structure
- ☑ Plan number embedded

**Cons:**
- ❌ Richmond team needs translation
- ❌ Not human-readable
- ❌ Long codes
- ❌ Requires decoder

---

#### Option C: Hybrid/Translation (RECOMMENDED Short-Term)
```sql
-- Keep BOTH systems, bridge with translation
CREATE TABLE options_richmond (
    option_code TEXT PRIMARY KEY,    -- 'XGREAT', '2CAR5XA'
    description TEXT NOT NULL,
    category TEXT,
    is_active INTEGER DEFAULT 1
);

CREATE TABLE options_holt (
    option_code TEXT PRIMARY KEY,    -- '167010100'
    plan_id TEXT,
    phase_code TEXT,
    option_id TEXT,
    elevation_code TEXT,
    description TEXT NOT NULL,
    is_active INTEGER DEFAULT 1
);

CREATE TABLE option_translation (
    translation_id INTEGER PRIMARY KEY,
    richmond_code TEXT,               -- 'XGREAT'
    holt_code TEXT,                   -- '167010600'
    universal_code TEXT,              -- 'OPT-INT-001' (future)
    description TEXT NOT NULL,
    category TEXT,                    -- 'INTERIOR', 'GARAGE'
    is_active INTEGER DEFAULT 1,
    UNIQUE(richmond_code),
    UNIQUE(holt_code)
);

-- Query by either code
SELECT * FROM option_translation
WHERE richmond_code = 'XGREAT' OR holt_code = '167010600';
```

**Pros:**
- ☑ Both teams use familiar codes
- ☑ No retraining during migration
- ☑ Preserves current workflows
- ☑ Translation enables interoperability
- ☑ Can phase in new standard gradually

**Cons:**
- ❌ Maintains two systems
- ❌ Translation overhead
- ❌ Potential sync errors
- ❌ More complex

---

#### Option D: New Universal System (RECOMMENDED Long-Term)
```sql
-- Create new standard everyone learns
CREATE TABLE options (
    option_code TEXT PRIMARY KEY,    -- 'OPT-GAR-001'
    category TEXT NOT NULL,          -- 'GAR', 'INT', 'STR'
    sequence INTEGER NOT NULL,       -- 001, 002, 003
    description TEXT NOT NULL,
    is_active INTEGER DEFAULT 1,
    UNIQUE(category, sequence)
);

CREATE TABLE option_translation (
    translation_id INTEGER PRIMARY KEY,
    universal_code TEXT NOT NULL,     -- 'OPT-GAR-001'
    richmond_code TEXT,               -- '2CAR5XA'
    holt_code TEXT,                   -- '167010100'
    description TEXT NOT NULL,
    FOREIGN KEY (universal_code) REFERENCES options(option_code)
);

-- Format: OPT-[CATEGORY]-[NUMBER]
-- Categories:
-- - GAR = Garage options
-- - INT = Interior options
-- - STR = Structural options
-- - EXT = Exterior options
-- - ELE = Electrical options
-- - PLU = Plumbing options

-- Examples:
-- OPT-GAR-001 = 2-car garage 4' extension
-- OPT-GAR-002 = 2-car garage 5' extension
-- OPT-INT-001 = Extended great room
-- OPT-STR-001 = Optional den
```

**Pros:**
- ☑ Clean, systematic
- ☑ Category grouping clear
- ☑ Short codes (11 characters)
- ☑ Easy to extend
- ☑ Neutral (neither Richmond nor Holt)
- ☑ Manor Homes can adopt
- ☑ Future-proof

**Cons:**
- ❌ Both teams need retraining

- ❌ Translation from both systems
- ❌ Migration effort
- ❌ Change management

---

### Recommendation (Phased Approach)

**SHORT-TERM (Weeks 1-12): Option C (Hybrid/Translation)**
- Keep Richmond codes in Richmond BAT
- Keep Holt codes in Holt BAT
- Build translation table (150-200 mappings)
- No retraining during migration
- Both teams productive immediately

**LONG-TERM (Post-Merger): Option D (Universal System)**
- Gradual migration to OPT-[CAT]-[NUM] format
- Phase in over 6-12 months
- Train both teams together
- Manor Homes adopts new standard
- Eventually deprecate old codes

**Translation Table Strategy:**
```sql
-- Week 2: Build translation table
INSERT INTO option_translation VALUES
  (1, 'OPT-INT-001', 'XGREAT', '167010600', 'Extended Great Room', 'INT', 1),
  (2, 'OPT-GAR-001', '2CAR4XA', '167010100', '2-Car Garage 4ft Ext Elev A', 'GAR',
1),
  (3, 'OPT-GAR-002', '2CAR5XA', '167010200', '2-Car Garage 5ft Ext Elev A', 'GAR',
1);

-- Weeks 5-8: Use during imports
-- Post-merger: Introduce universal codes
-- 6-12 months: Phase out old codes
```

**Final Decision:** [To be filled Tuesday after team input]

---

## 📐 DATABASE SCHEMA DESIGN

### Core Tables (Based on Decisions)

The following schema assumes:
- Decision 1: Hybrid Plan-Pack model
- Decision 2: Elevation as Dimension
- Decision 3: Hybrid Option Codes (short-term)

```sql
-- ============================================
-- CORE TABLES (10 tables)
-- ============================================

-- Table 1: Builders
CREATE TABLE builders (
    builder_id TEXT PRIMARY KEY,     -- 'RICHMOND', 'HOLT'
    builder_name TEXT NOT NULL,
    is_active INTEGER DEFAULT 1,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP
);
-- Prism: builder_id VARCHAR(50), is_active BOOLEAN, created_date TIMESTAMP

-- Table 2: Plans (Decision 2: NO elevation in plan_id)
CREATE TABLE plans (
    plan_id TEXT PRIMARY KEY,        -- 'G603', '1670', '1890'
    builder_id TEXT NOT NULL,
    plan_name TEXT,
    sq_ft INTEGER,
    bedrooms INTEGER,
    bathrooms REAL,
    is_active INTEGER DEFAULT 1,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (builder_id) REFERENCES builders(builder_id)
);
-- Prism: plan_id VARCHAR(50), builder_id VARCHAR(50), is_active BOOLEAN

-- Table 3: Plan Elevations (Decision 2: Elevation as separate dimension)
CREATE TABLE plan_elevations (
    elevation_id INTEGER PRIMARY KEY,
    plan_id TEXT NOT NULL,
    elevation_code TEXT NOT NULL,    -- 'A', 'B', 'C', 'D'
    elevation_name TEXT,              -- 'Elevation A'
    is_active INTEGER DEFAULT 1,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    UNIQUE(plan_id, elevation_code)
);
-- Prism: elevation_id SERIAL, plan_id VARCHAR(50), elevation_code VARCHAR(10)

-- Table 4: Packs (Decision 1 & 2: Clean pack names, no elevation)
CREATE TABLE packs (
    pack_id TEXT PRIMARY KEY,        -- '10.82', '12.x5'
    pack_name TEXT NOT NULL,         -- 'OPT DEN FOUNDATION' (clean!)
    phase TEXT,                       -- '10', '12'
    pack_type INTEGER,                -- 1=Foundation, 2=Framing, etc.
    is_universal INTEGER DEFAULT 0,   -- Decision 1: Hybrid flag
    description TEXT,
    is_active INTEGER DEFAULT 1,
```

```
    created_date TEXT DEFAULT CURRENT_TIMESTAMP
);
-- Prism: pack_id VARCHAR(50), is_universal BOOLEAN, is_active BOOLEAN

-- Table 5: Pack Elevations (Decision 2: Which elevations pack applies to)
CREATE TABLE pack_elevations (
    pack_elevation_id INTEGER PRIMARY KEY,
    pack_id TEXT NOT NULL,
    elevation_code TEXT NOT NULL,    -- 'B', 'C', 'D'
    notes TEXT,
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id),
    UNIQUE(pack_id, elevation_code)
);
-- Prism: pack_elevation_id SERIAL

-- Table 6: Plan Packs (Decision 1: Hybrid junction table)
CREATE TABLE plan_packs (
    plan_pack_id INTEGER PRIMARY KEY,
    plan_id TEXT NOT NULL,
    pack_id TEXT NOT NULL,
    override_materials INTEGER DEFAULT 0, -- Plan-specific override?
    display_order INTEGER,
    notes TEXT,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id),
    UNIQUE(plan_id, pack_id)
);
-- Prism: plan_pack_id SERIAL, override_materials BOOLEAN

-- Table 7: Items (Material catalog)
CREATE TABLE items (
    item_id TEXT PRIMARY KEY,        -- SKU or item number
    item_description TEXT NOT NULL,
    category TEXT,                    -- 'Lumber', 'Trusses', 'Windows'
    subcategory TEXT,
    unit_of_measure TEXT,             -- 'EA', 'LF', 'SQFT'
    vendor_id TEXT,
    is_active INTEGER DEFAULT 1,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP
);
-- Prism: item_id VARCHAR(50), is_active BOOLEAN

-- Table 8: Materials (Decision 1 & 2: Links everything together)
CREATE TABLE materials (
    material_id INTEGER PRIMARY KEY,
    plan_pack_id INTEGER NOT NULL,  -- Decision 1: Links to plan-pack combo
    elevation_id INTEGER,            -- Decision 2: Links to specific elevation
    item_id TEXT NOT NULL,
    quantity REAL NOT NULL,
```

```
    unit_cost REAL,
    total_cost REAL,
    notes TEXT,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (plan_pack_id) REFERENCES plan_packs(plan_pack_id),
    FOREIGN KEY (elevation_id) REFERENCES plan_elevations(elevation_id),
    FOREIGN KEY (item_id) REFERENCES items(item_id)
);
-- Prism: material_id SERIAL, quantity DECIMAL(10,2), unit_cost DECIMAL(10,2)

-- Table 9: Pricing (Price levels and history)
CREATE TABLE pricing (
    price_id INTEGER PRIMARY KEY,
    item_id TEXT NOT NULL,
    price_level TEXT NOT NULL,       -- 'L1', 'L2', 'PL01', 'PL12'
    price REAL NOT NULL,
    effective_date TEXT,              -- ISO8601: '2025-11-10'
    end_date TEXT,
    margin_pct REAL,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (item_id) REFERENCES items(item_id)
);
-- Prism: price_id SERIAL, price DECIMAL(10,2), effective_date TIMESTAMP

-- Table 10: Communities (Holt-specific)
CREATE TABLE communities (
    community_id TEXT PRIMARY KEY,   -- 'CR', 'GG', 'WR', 'HA', 'HH'
    community_name TEXT NOT NULL,
    builder_id TEXT NOT NULL,
    is_active INTEGER DEFAULT 1,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (builder_id) REFERENCES builders(builder_id)
);
-- Prism: community_id VARCHAR(10), builder_id VARCHAR(50), is_active BOOLEAN

-- ============================================
-- SUPPORTING TABLES (3 tables)
-- ============================================

-- Table 11: Option Translation (Decision 3: Hybrid approach)
CREATE TABLE option_translation (
    translation_id INTEGER PRIMARY KEY,
    richmond_code TEXT,               -- 'XGREAT', '2CAR5XA'
    holt_code TEXT,                   -- '167010600'
    universal_code TEXT,              -- 'OPT-INT-001' (future)
    description TEXT NOT NULL,
    category TEXT,                    -- 'GAR', 'INT', 'STR', 'EXT'
    is_active INTEGER DEFAULT 1,
    created_date TEXT DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(richmond_code),
```

```
    UNIQUE(holt_code),
    UNIQUE(universal_code)
);
-- Prism: translation_id SERIAL, is_active BOOLEAN

-- Table 12: Pack Hierarchy (Display/construction order)
CREATE TABLE pack_hierarchy (
    hierarchy_id INTEGER PRIMARY KEY,
    pack_id TEXT NOT NULL,
    parent_pack_id TEXT,
    display_order INTEGER,
    construction_sequence INTEGER,
    FOREIGN KEY (pack_id) REFERENCES packs(pack_id),
    FOREIGN KEY (parent_pack_id) REFERENCES packs(pack_id)
);
-- Prism: hierarchy_id SERIAL

-- Table 13: Plan Communities (Holt: which plans in which communities)
CREATE TABLE plan_communities (
    plan_community_id INTEGER PRIMARY KEY,
    plan_id TEXT NOT NULL,
    community_id TEXT NOT NULL,
    is_active INTEGER DEFAULT 1,
    FOREIGN KEY (plan_id) REFERENCES plans(plan_id),
    FOREIGN KEY (community_id) REFERENCES communities(community_id),
    UNIQUE(plan_id, community_id)
);
-- Prism: plan_community_id SERIAL, is_active BOOLEAN
```

### Sample Queries

**Query 1: Show all materials for Plan G603, Elevation B**
```sql
SELECT
    p.plan_id,
    pe.elevation_code,
    pk.pack_name,
    i.item_description,
    m.quantity,
    m.unit_cost,
    (m.quantity * m.unit_cost) AS total_cost
FROM materials m
JOIN plan_packs pp ON m.plan_pack_id = pp.plan_pack_id
JOIN plans p ON pp.plan_id = p.plan_id
JOIN plan_elevations pe ON m.elevation_id = pe.elevation_id
JOIN packs pk ON pp.pack_id = pk.pack_id
JOIN items i ON m.item_id = i.item_id
WHERE p.plan_id = 'G603'
  AND pe.elevation_code = 'B'
```

```
ORDER BY pk.pack_id, i.item_description;
```

**Query 2: Show all plans using pack 12.x5**
```sql
SELECT DISTINCT p.plan_id, p.plan_name, p.sq_ft
FROM plan_packs pp
JOIN plans p ON pp.plan_id = p.plan_id
WHERE pp.pack_id = '12.x5'
ORDER BY p.plan_id;
```

**Query 3: Translate Richmond option to Holt**
```sql
SELECT
    ot.richmond_code,
    ot.holt_code,
    ot.universal_code,
    ot.description,
    ot.category
FROM option_translation ot
WHERE ot.richmond_code = 'XGREAT';
```

**Query 4: Show all elevations across all plans**
```sql
SELECT
    p.plan_id,
    p.plan_name,
    GROUP_CONCAT(pe.elevation_code, ', ') AS elevations
FROM plans p
JOIN plan_elevations pe ON p.plan_id = pe.plan_id
WHERE p.is_active = 1
GROUP BY p.plan_id, p.plan_name
ORDER BY p.plan_id;
```

**Query 5: Show packs that apply to Elevation B only**
```sql
SELECT DISTINCT pk.pack_id, pk.pack_name
FROM packs pk
JOIN pack_elevations pkv ON pk.pack_id = pkv.pack_id
WHERE pkv.elevation_code = 'B'
  AND pk.pack_id NOT IN (
    SELECT pack_id FROM pack_elevations WHERE elevation_code != 'B'
  )
ORDER BY pk.pack_id;
```

---

## 📝 CODING STANDARDS

### Plan Coding
**Based on Decision 2: Elevation as Dimension**

```
Format: [BUILDER_PREFIX][NUMBER]

Richmond Examples:
- G603 (NO elevation in plan_id)
- G914
- LE93

Holt Examples:
- 1670 (NO elevation in plan_id)
- 1890
- 2676

Validation Rules:
- 4-6 characters
- Alphanumeric only
- No spaces
- Uppercase
- NO elevation suffix

Correct: G603, G914, 1670, LE93
Incorrect: G603B (has elevation), g603 (lowercase), G 603 (space)
```

### Elevation Coding
**Based on Decision 2: Separate Dimension**

```
Format: Single letter

Valid Values: A, B, C, D

Storage: plan_elevations table
- plan_id = 'G603'
- elevation_code = 'B'

NOT stored as: G603B (old triple-encoding problem)

Sheet Naming (for reference):
- materialist_G603_B (plan_id + _ + elevation_code)
- bidtotals_1670_CR_A (plan + community + elevation)

Validation Rules:
- Must be A, B, C, or D
```

- Uppercase only
- One character
```

### Pack Coding
**Based on Decisions 1 & 2: Clean names, no elevation**

```
Format: |[PHASE].[VARIANT] [DESCRIPTION]

Examples:
- |10 FOUNDATION
- |10.82 OPT DEN FOUNDATION (clean, no BCD!)
- |12.x5 OPT 2 CAR GARAGE 5' EXT FOUNDATION

Pack Types (database):
1 = Foundation
2 = Framing
3 = Exterior
4 = Interior
5 = Garage
6 = Electrical
7 = Plumbing
8 = HVAC
9 = Special

Validation Rules:
- Starts with | (pipe character)
- Phase: 2 digits
- Variant: optional, format .XX or .xX
- Description: clear, concise
- NO elevation codes in name!

Elevation Applicability:
- Stored in pack_elevations table
- NOT in pack name
```

### Option Coding
**Based on Decision 3: Hybrid (Short-Term)**

```
Richmond Format:
- Descriptive alphanumeric
- Variable length (6-10 characters)
- Examples: XGREAT, 2CAR5XA, FPSING01

Holt Format:
- Numeric hierarchical
- 9 digits: [PLAN 4][PHASE 2][OPTION 2][ELEVATION 2]

```
- Examples: 167010600, 164910105

Internal Database:
- Use option_translation table
- Bridge between both systems
- Query by either code

Future Universal Format:
- OPT-[CATEGORY]-[NUMBER]
- Examples: OPT-GAR-001, OPT-INT-001
- Phase in post-merger
```

### Sheet/Table Naming Convention

```
Format: [TYPE]_[PLAN]_[COMMUNITY?]_[ELEVATION?]

Types:
- materialist = Material list for plan
- bidtotals = Bid totals
- pricing = Pricing table
- ref = Reference data

Richmond Examples:
- materialist_G603_B (plan + elevation)
- pricing_base (shared)

Holt Examples:
- materialist_1670_CR_A (plan + community + elevation)
- bidtotals_1890_GG_B

Validation Rules:
- Lowercase with underscores
- No spaces
- No special characters except underscore
- Consistent order
```

---

## ☑ SUCCESS CRITERIA

### Week 1 Complete When:

**Decisions Made:**
- [ ] Decision 1 documented with rationale
- [ ] Decision 2 documented with rationale
- [ ] Decision 3 documented with rationale
- [ ] All three tested with real data

- [ ] Edge cases identified

**Schema Designed:**
- [ ] 10 core tables defined
- [ ] 3 supporting tables defined
- [ ] All relationships documented
- [ ] Sample queries tested
- [ ] Prism migration notes included

**Standards Documented:**
- [ ] Plan coding rules clear
- [ ] Elevation coding rules clear
- [ ] Pack coding rules clear
- [ ] Option coding rules clear
- [ ] Examples for all formats

**Team Validated:**
- [ ] William reviewed (Richmond perspective)
- [ ] Alicia reviewed (Holt perspective)
- [ ] Feedback incorporated
- [ ] Team understands decisions
- [ ] Team agrees with approach

**Foundation Locked:**
- [ ] All documents finalized
- [ ] Schema approved
- [ ] Standards published
- [ ] Reference sheets in BATs
- [ ] Ready for Week 2

---

## 🚀 NEXT STEPS

### Immediate (Tuesday)
1. Map hierarchies (2 hours)
2. Make 3 decisions (2 hours)
3. Design schema (2 hours)

### Wednesday-Thursday
1. Document standards based on decisions
2. Create import mapping rules
3. Prepare team review materials

### Friday
1. Team validation session
2. Incorporate feedback
3. Finalize and lock foundation

### Week 2 and Beyond

1. Build pricing tools using this schema
2. Import plans following these standards
3. Maintain consistency across all weeks

---

**Document Owner:** Corey Boser
**Last Updated:** November 10, 2025
**Status:** Active - To be completed Tuesday
**Next Review:** After Week 1 completion

---

**Current Action:** Ready for Tuesday's architecture decisions →
**See 03_FOUNDATION_GUIDE.md for detailed Tuesday execution plan**