Lesson 3 — Allowlist Rules and Filtering Logic

Overview

This lesson teaches you how to implement robust filtering logic that enforces business rules on data selection. You'll learn to design inclusion/exclusion patterns, handle edge cases in string matching, and build deterministic filtering systems that comply with project guidelines (AGENTS.md).

Television campaign analytics systems generate dozens of table types. Not all tables are appropriate for every use case. This lesson focuses on implementing a **table allowlist** that:

- **Includes** TRP (Target Rating Point) tables and TabSummary tables
- **Excludes** plot visualizations and 30-second equivalent metrics
- Handles variations in table naming conventions
- Provides clear feedback on why tables are rejected

Learning Outcomes

After completing this lesson, you will be able to:

1. Implement inclusion and exclusion rule patterns
2. Handle edge cases in string matching (case sensitivity, notation variants)
3. Design filtering logic with clear rejection reasons
4. Build statistics tracking for filtering operations
5. Test filtering behavior comprehensively

Prerequisites

- Completion of Lessons 1-2
- Understanding of Python string operations
- Familiarity with list comprehensions
- Knowledge of logging best practices

The Problem: Uncontrolled Data Access

Why Allowlists Matter

In production systems handling sensitive or complex data:

- **Security**: Prevent access to restricted datasets

- **Performance**: Reduce processing overhead by filtering early
- **Compliance**: Enforce regulatory requirements (e.g., GDPR, industry standards)
- **Cost**: Minimize API calls and computational resources
- **Accuracy**: Ensure only validated, approved data flows through pipelines

For our Campaign 1 system:

- **Input**: 50+ possible table configurations from `element_config.json`
- **Allowed**: Only ~25 TRP and TabSummary tables
- **Excluded**: All plots and 30-second equivalent calculations
- **Reason**: Plots are for visualization (not agentic processing), 30eq metrics are derived and redundant

Part 1: Understanding the Allowlist Rules

Business Requirements (from AGENTS.md)

```
## Table allowlist
- Tables-only; include TRP tables and TabSummary tables.
- Exclude plots and exclude any `.30eq` element.
```

Rule Interpretation

Inclusion Rules (logical OR)

Rule 1: TRP Tables

- Contains `"trp"` (case-insensitive)
- Examples: `contact_sexage_trp_raw` , `contactcum_target_trp`

Rule 2: TabSummary Tables

- Contains `"tabsummary"` (case-insensitive)
- Examples: `tabsummary_table_contactreach_target_perc`

Exclusion Rules (take precedence, logical OR)

Rule 1: Plot Elements

- Contains `"plot"` (case-insensitive)
- Examples: `contact_sexage_abs_plot` , `plot_reach_target`

Rule 2: 30-Second Equivalent Metrics

- Contains `".30eq"` or `"30eq"` (any position)
- Examples: `reach_target_perc.30eq` , `table_reach_target_30eq`

Precedence: Exclusions Beat Inclusions

```
# Example table name: "contact_trp_plot"
# - Matches inclusion rule (contains "trp")
# - Matches exclusion rule (contains "plot")
# Result: REJECTED (exclusion wins)
```

Part 2: Implementation Strategy

High-Level Algorithm

```python
def filter_tables_by_allowlist(table_names: list[str]) -> dict:
    allowed = []
    rejected = []

    for table in table_names:
        # Step 1: Check exclusions FIRST
        if is_excluded(table):
            rejected.append((table, reason))
            continue

        # Step 2: Check inclusions
        if is_included(table):
            allowed.append(table)
        else:
            # Default: allow (or reject based on policy)
            allowed.append(table)

    return {"allowed": allowed, "rejected": rejected}
```

Design Decisions

1. **Check exclusions before inclusions** - fail-fast pattern
2. **Case-insensitive matching** - handle naming variations
3. **Clear rejection reasons** - aid debugging and transparency
4. **Statistics tracking** - monitor filtering effectiveness
5. **Logging** - visibility into filtering decisions

Part 3: Complete Implementation

The filter_tables_by_allowlist Function

From `lesson_01_working_code.py` (lines 246-340):

```python
import logging
from typing import Any
```

```python
logger = logging.getLogger(__name__)

def filter_tables_by_allowlist(table_names: list[str]) -> dict[str, Any]:
    """
    Filter table names based on AGENTS.md allowlist rules.

    Allowlist rules:
    - INCLUDE: TRP tables (keys containing 'trp')
    - INCLUDE: TabSummary tables (keys containing 'tabsummary')
    - EXCLUDE: Plot elements (keys containing 'plot')
    - EXCLUDE: 30-second equivalents (keys containing '30eq')

    Args:
        table_names: List of table element keys to filter

    Returns:
        Dictionary with allowed/rejected tables and statistics
    """
    logger.info(f"Filtering {len(table_names)} tables...")

    allowed_tables = []
    rejected_tables = []

    for table_name in table_names:
        table_lower = table_name.lower()  # Case-insensitive matching
        rejection_reason = None

        # Check exclusion rules FIRST (they take precedence)
        if "plot" in table_lower or ".30eq" in table_lower or "30eq" in table_lower:
            if "plot" in table_lower:
                rejection_reason = "excluded: contains 'plot'"
            else:
                rejection_reason = "excluded: contains '30eq'"

        # Check inclusion rules if not already rejected
        elif "trp" in table_lower or "tabsummary" in table_lower:
            allowed_tables.append(table_name)
            logger.debug(f"  [ALLOWED] {table_name}")
        else:
            # Other table types: allowed by default (or reject based on policy)
            # For strict mode: rejection_reason = "not in allowlist"
            allowed_tables.append(table_name)
            logger.debug(f"  [ALLOWED] {table_name}")

        # Record rejections
        if rejection_reason:
            rejected_tables.append({
                "name": table_name,
                "reason": rejection_reason,
            })
            logger.debug(f"  [REJECTED] {table_name}: {rejection_reason}")

    # Calculate statistics
    total_input = len(table_names)
    allowed_count = len(allowed_tables)
    rejected_count = len(rejected_tables)
    allowed_percentage = (
        round(100 * allowed_count / total_input, 1) if total_input > 0 else 0
    )

    result = {
        "allowed_tables": allowed_tables,
        "rejected_tables": rejected_tables,
        "allowlist_rules": [
            "INCLUDE: TRP tables (contain 'trp')",
            "INCLUDE: TabSummary tables (contain 'tabsummary')",
            "EXCLUDE: Plot elements (contain 'plot')",
            "EXCLUDE: 30eq elements (contain '30eq')",
        ],
        "statistics": {
            "total_input": total_input,
            "allowed_count": allowed_count,
            "rejected_count": rejected_count,
            "allowed_percentage": allowed_percentage,
        }
    }

    logger.info(
        f"Filtering complete: {allowed_count} allowed, {rejected_count} rejected"
    )
    return result
```

Part 4: Edge Cases and Variations

Case Sensitivity

```
# All these should match "trp" rule
table_names = [
    "contact_TRP_raw",      # Uppercase
    "Contact_Trp_Raw",      # Mixed case
    "CONTACT_trp_RAW",      # Various positions
]

# Solution: Convert to lowercase for comparison
table_lower = table_name.lower()
```

Notation Variants for 30eq

Different naming conventions exist:

```
# Variations to handle:
"standard_tabr1_table_reach_target_perc.30eq"   # Dot notation
"standard_tabr1_table_reach_target_perc_30eq"   # Underscore notation
"table_30eq_reach"                               # Position variation

# Solution: Check for both ".30eq" and "30eq" (without dot)
if ".30eq" in table_lower or "30eq" in table_lower:
    # Reject
```

Substring Matching

Be careful with partial matches:

```
# Edge case: "plotdata" contains "plot" - should it be rejected?
# Decision: Yes, our rule is simple substring match

# If you need word-boundary matching:
import re
if re.search(r'\bplot\b', table_lower):  # Matches whole word only
    # Reject
```

For our use case, **substring matching is sufficient** because:

- Table names use underscores, not concatenation
- "plot" as a substring consistently indicates visualization elements

Empty Input

```
# Handle empty list gracefully
result = filter_tables_by_allowlist([])
assert result["statistics"]["total_input"] == 0
assert result["statistics"]["allowed_percentage"] == 0
```

Part 5: Testing the Filter

Test Cases

```
def test_filter_tables_by_allowlist():
    """Comprehensive test suite for table filtering."""

    # Test 1: TRP tables are allowed
    trp_tables = [
        "standard_tabcontacts_table_contact_sexage_trp_raw",
        "standard_tabcontacts_table_contact_target_trp_raw",
        "standard_tabcontactsbu_table_contactcum_target_trp",
    ]
    result = filter_tables_by_allowlist(trp_tables)
    assert len(result["allowed_tables"]) == 3
    assert len(result["rejected_tables"]) == 0

    # Test 2: TabSummary tables are allowed
    summary_tables = [
        "standard_tabsummary_table_contactreach_target_perc",
        "standard_tabsummary_table_contactreach_sexage_abs",
    ]
    result = filter_tables_by_allowlist(summary_tables)
    assert len(result["allowed_tables"]) == 2

    # Test 3: Plot tables are rejected
    plot_tables = [
        "standard_tabcontacts_plot_contact_sexage_abs_raw",
        "standard_plot_reach_target",
    ]
    result = filter_tables_by_allowlist(plot_tables)
    assert len(result["allowed_tables"]) == 0
    assert len(result["rejected_tables"]) == 2
    assert all("plot" in r["reason"] for r in result["rejected_tables"])

    # Test 4: 30eq tables are rejected (multiple notations)
    eq_tables = [
        "standard_tabr1_table_reach_target_perc.30eq",
        "standard_tabr1_table_reach_target_perc_30eq",
    ]
    result = filter_tables_by_allowlist(eq_tables)
    assert len(result["rejected_tables"]) == 2
    assert all("30eq" in r["reason"] for r in result["rejected_tables"])

    # Test 5: Mixed list
    mixed_tables = [
        "standard_tabcontacts_table_contact_sexage_trp_raw",  # Allow (TRP)
        "standard_tabcontacts_plot_contact_sexage_abs_raw",   # Reject (plot)
        "standard_tabsummary_table_contactreach_target_perc", # Allow (summary)
        "standard_tabr1_table_reach_target_perc.30eq",        # Reject (30eq)
        "standard_tabrf_table_reach_target_abs",              # Allow (default)
    ]
    result = filter_tables_by_allowlist(mixed_tables)
    assert len(result["allowed_tables"]) == 3
    assert len(result["rejected_tables"]) == 2

    # Test 6: Case insensitivity
    case_variants = [
        "table_TRP_raw",
        "table_Trp_raw",
        "table_PLOT_data",
        "table_Plot_Data",
    ]
    result = filter_tables_by_allowlist(case_variants)
    assert len(result["allowed_tables"]) == 2  # TRP variants
    assert len(result["rejected_tables"]) == 2  # PLOT variants

    # Test 7: Statistics calculation
    assert result["statistics"]["total_input"] == 4
    assert result["statistics"]["allowed_percentage"] == 50.0

    print(" All filter tests passed!")

# Run tests
test_filter_tables_by_allowlist()
```

Integration Test with Real Data

```
from pathlib import Path
import json

def test_with_campaign1_element_config():
    """Test filtering against actual Campaign 1 element_config.json."""

    config_path = Path("Starter Kit - Agentic Models/Campaign 1") / \
                  "02_postprocessing" / "element_config.json"
    with open(config_path) as f:
        element_config = json.load(f)

    # Get all table keys
    all_tables = list(element_config.keys())
    print(f"Total tables in element_config: {len(all_tables)}")

    # Filter
    result = filter_tables_by_allowlist(all_tables)

    # Analyze results
    print(f"Allowed: {result['statistics']['allowed_count']}")
    print(f"Rejected: {result['statistics']['rejected_count']}")
    print(f"Percentage allowed: {result['statistics']['allowed_percentage']}%")

    # Show rejection reasons
    print("\nRejected tables:")
    for r in result["rejected_tables"]:
        print(f"  - {r['name']}: {r['reason']}")

    # Verify expected counts (Campaign 1 has 25 allowed tables)
    assert result['statistics']['allowed_count'] == 25

test_with_campaign1_element_config()
```

Part 6: Logging and Observability

Logging Levels

```
import logging

logger = logging.getLogger(__name__)

def filter_tables_by_allowlist(table_names: list[str]) -> dict[str, Any]:
    # INFO: High-level operation summary
    logger.info(f"Filtering {len(table_names)} tables...")

    for table_name in table_names:
        # DEBUG: Individual decision details
        if allowed:
            logger.debug(f"  [ALLOWED] {table_name}")
        else:
            logger.debug(f"  [REJECTED] {table_name}: {reason}")

    # INFO: Operation result
    logger.info(f"Filtering complete: {allowed_count} allowed, {rejected_count} rejected")
```

Debug Output Example

```
2026-01-19 10:15:23 - INFO - Filtering 5 tables...
2026-01-19 10:15:23 - DEBUG -   [ALLOWED] standard_tabcontacts_table_contact_sexage_trp_raw
```

```
2026-01-19 10:15:23 - DEBUG -   [REJECTED] standard_tabcontacts_plot_contact_sexage_abs_raw: excluded
2026-01-19 10:15:23 - DEBUG -   [ALLOWED] standard_tabsummary_table_contactreach_target_perc
2026-01-19 10:15:23 - DEBUG -   [REJECTED] standard_tabr1_table_reach_target_perc.30eq: excluded: con
2026-01-19 10:15:23 - DEBUG -   [ALLOWED] standard_tabrf_table_reach_target_abs
2026-01-19 10:15:23 - INFO - Filtering complete: 3 allowed, 2 rejected
```

Part 7: Advanced Patterns and Extensions

Pattern 1: Strict Mode (Reject Unknown)

```python
def filter_tables_strict(table_names: list[str]) -> dict[str, Any]:
    """Strict filtering: only explicitly allowed tables pass."""
    allowed_tables = []
    rejected_tables = []

    for table_name in table_names:
        table_lower = table_name.lower()
        rejection_reason = None

        # Exclusions (same as before)
        if "plot" in table_lower or "30eq" in table_lower:
            rejection_reason = "excluded: plot or 30eq"

        # Inclusions (ONLY these pass)
        elif "trp" in table_lower or "tabsummary" in table_lower:
            allowed_tables.append(table_name)

        # NEW: Reject anything not explicitly allowed
        else:
            rejection_reason = "not in allowlist (strict mode)"

        if rejection_reason:
            rejected_tables.append({"name": table_name, "reason": rejection_reason})

    return {"allowed_tables": allowed_tables, "rejected_tables": rejected_tables}
```

Pattern 2: Configurable Rules

```python
from dataclasses import dataclass

@dataclass
class AllowlistConfig:
    """Configuration for table filtering rules."""
    include_patterns: list[str]
    exclude_patterns: list[str]
    case_sensitive: bool = False
    strict_mode: bool = False

def filter_tables_configurable(
    table_names: list[str],
    config: AllowlistConfig
) -> dict[str, Any]:
    """Filter tables with configurable rules."""
    allowed_tables = []
    rejected_tables = []

    for table_name in table_names:
        table_text = table_name if config.case_sensitive else table_name.lower()
        rejection_reason = None

        # Check exclusions
        for pattern in config.exclude_patterns:
            pattern_text = pattern if config.case_sensitive else pattern.lower()
            if pattern_text in table_text:
                rejection_reason = f"excluded: matches pattern '{pattern}'"
                break

        # Check inclusions (if not excluded)
        if not rejection_reason:
            matched_inclusion = any(
                (pattern if config.case_sensitive else pattern.lower()) in table_text
                for pattern in config.include_patterns
            )

            if matched_inclusion:
                allowed_tables.append(table_name)
            elif config.strict_mode:
                rejection_reason = "not in allowlist (strict mode)"
            else:
                allowed_tables.append(table_name)  # Permissive default

        if rejection_reason:
            rejected_tables.append({"name": table_name, "reason": rejection_reason})

    return {"allowed_tables": allowed_tables, "rejected_tables": rejected_tables}

# Usage
campaign1_config = AllowlistConfig(
    include_patterns=["trp", "tabsummary"],
    exclude_patterns=["plot", "30eq", ".30eq"],
    case_sensitive=False,
    strict_mode=False,
)

result = filter_tables_configurable(table_names, campaign1_config)
```

Pattern 3: Regex-Based Filtering

```
import re

def filter_tables_regex(table_names: list[str]) -> dict[str, Any]:
    """Filter using regex patterns for precise matching."""

    # Patterns
    include_pattern = re.compile(r'(trp|tabsummary)', re.IGNORECASE)
    exclude_pattern = re.compile(r'(plot|\.?30eq)', re.IGNORECASE)

    allowed_tables = []
    rejected_tables = []
    for table_name in table_names:
        # Check exclusion regex
        if exclude_pattern.search(table_name):
            reason = f"excluded: matched pattern {exclude_pattern.pattern}"
            rejected_tables.append({"name": table_name, "reason": reason})

        # Check inclusion regex
        elif include_pattern.search(table_name):
            allowed_tables.append(table_name)

        else:
            allowed_tables.append(table_name)  # Default allow

    return {"allowed_tables": allowed_tables, "rejected_tables": rejected_tables}
```

Part 8: Common Pitfalls

Pitfall 1: Forgetting Case Insensitivity

```
# Wrong: Case-sensitive matching
if "trp" in table_name:  # Misses "TRP", "Trp"
    allowed_tables.append(table_name)

# Right: Case-insensitive
table_lower = table_name.lower()
if "trp" in table_lower:
    allowed_tables.append(table_name)
```

Pitfall 2: Wrong Rule Precedence

```
# Wrong: Inclusions before exclusions
if "trp" in table_lower:
    allowed_tables.append(table_name)
elif "plot" in table_lower:
    rejected_tables.append(...)
# Problem: "trp_plot" gets allowed!

# Right: Exclusions first
if "plot" in table_lower:
    rejected_tables.append(...)
elif "trp" in table_lower:
    allowed_tables.append(table_name)
```

Pitfall 3: Missing Edge Cases

```
# Wrong: Only checks ".30eq"
if ".30eq" in table_lower:  # Misses "table_30eq_data"
    rejected_tables.append(...)
```

```
# Right: Check both notations
if ".30eq" in table_lower or "30eq" in table_lower:
    rejected_tables.append(...)
```

Pitfall 4: Poor Error Messages

```
# Wrong: Generic message
rejected_tables.append({"name": table_name, "reason": "rejected"})

# Right: Specific reason
rejected_tables.append({
    "name": table_name,
    "reason": "excluded: contains 'plot'"
})
```

Part 9: Performance Considerations

Time Complexity

```
# Current implementation: O(n * m)
# n = number of tables
# m = average length of table name string

# For Campaign 1: ~50 tables, average name length ~60 chars
# Total: ~3,000 character comparisons - negligible

# Optimization for large datasets (1000s of tables):
def filter_tables_optimized(table_names: list[str]) -> dict[str, Any]:
    """Optimized for large datasets."""
    # Pre-compile regex (one-time cost)
    include_re = re.compile(r'(trp|tabsummary)', re.IGNORECASE)
    exclude_re = re.compile(r'(plot|\.?30eq)', re.IGNORECASE)

    allowed = []
    rejected = []

    for table_name in table_names:
        if exclude_re.search(table_name):
            rejected.append({"name": table_name, "reason": "excluded"})
        elif include_re.search(table_name):
            allowed.append(table_name)
        else:
            allowed.append(table_name)

    return {"allowed_tables": allowed, "rejected_tables": rejected}
```

Part 10: Hands-On Exercise

Exercise: Extend the Filter

Objective: Add a new exclusion rule and update statistics.

Requirements:

1. Add exclusion for tables containing `"test"` (debugging/test tables)
2. Update rejection reasons to specify which exclusion matched
3. Add a new statistic: `"rejection_reasons_count"` (count by reason type)

Starter Code:

```python
def filter_tables_extended(table_names: list[str]) -> dict[str, Any]:
    """Extended filter with test exclusion and detailed statistics."""
    allowed_tables = []
    rejected_tables = []
    rejection_reasons_count = {}  # Track counts by reason

    for table_name in table_names:
        table_lower = table_name.lower()
        rejection_reason = None

        # TODO: Add exclusion rules
        # - Check for "plot"
        # - Check for "30eq" (both notations)
        # - Check for "test"

        # TODO: Add inclusion rules
        # - Check for "trp"
        # - Check for "tabsummary"

        # TODO: Update rejection_reasons_count

        if rejection_reason:
            rejected_tables.append({
                "name": table_name,
                "reason": rejection_reason,
            })
        else:
            allowed_tables.append(table_name)

    return {
        "allowed_tables": allowed_tables,
        "rejected_tables": rejected_tables,
        "statistics": {
            "total_input": len(table_names),
            "allowed_count": len(allowed_tables),
            "rejected_count": len(rejected_tables),
            "rejection_reasons_count": rejection_reasons_count,
        }
    }
```

Solution:

▶ Click to reveal solution

Knowledge Check

Question 1: Rule Precedence

Given table name `"table_trp_plot_data"` , what is the correct result?

A) Allowed (contains "trp")
B) Rejected (contains "plot")
C) Depends on rule order in code
D) Error - conflicting rules

▶ Answer

Question 2: Case Sensitivity

Which implementation is case-insensitive?

A) `if "TRP" in table_name:`
B) `if "trp" in table_name.lower():`
C) `if table_name == "trp":`
D) `if "trp".upper() in table_name:`

▶ Answer

Question 3: 30eq Notation

Which table names should be rejected for containing "30eq"?

A) Only `"table.30eq"`
B) Only `"table_30eq"`
C) Both `"table.30eq"` and `"table_30eq"`
D) Neither (30eq is not in the exclusion list)

▶ Answer

Question 4: Statistics

If 100 tables are filtered and 75 are allowed, what is `allowed_percentage` ?

A) 75
B) 75.0
C) 0.75
D) "75%"

▶ Answer

Question 5: Empty Input

What should happen if `filter_tables_by_allowlist([])` is called?

A) Raise ValueError
B) Return None
C) Return empty result with 0 statistics
D) Return default set of tables

▶ Answer

Summary

In this lesson, you learned:

1. **Allowlist pattern** enforces business rules on data selection:
    - Inclusion rules define what's allowed
    - Exclusion rules define what's forbidden
    - Exclusions take precedence
2. **Implementation techniques**:
    - Case-insensitive string matching
    - Multiple notation handling (`.30eq` vs `30eq`)
    - Clear rejection reasons for transparency
    - Statistics tracking for monitoring
3. **Edge cases**:
    - Empty input handling
    - Case variations
    - Notation variations
    - Conflicting rules (exclusion wins)
4. **Best practices**:
    - Check exclusions before inclusions
    - Use lowercase for case-insensitive matching
    - Log decisions for debugging
    - Track statistics for visibility

Next Steps

In **Lesson 4**, you'll learn:

- Generating mock API calls from natural language
- Mapping user requests to specific data types
- Building request bodies for dataBreeders API
- Handling missing fixtures gracefully

References

- AGENTS.md - Project guidelines and allowlist rules
- Python String Methods
- Python Logging
- Python Regular Expressions

Code Reference

Complete working implementation: lesson_01_working_code.py (lines 246-340)