Uderia Platform - Tutorial Master Plan

Comprehensive Tutorial Structure & Narration Guide

Platform: Uderia (www.uderia.com)
Tutorial Platform: descript
Target Duration per Section: 2-5 minutes
Last Updated: December 7, 2025

Table of Contents

Quick Navigation

- Tutorial Philosophy
- Profile Personas Overview
- Module 1: Introduction & Quick Start
  - 1.1 Welcome to Uderia - The Platform Promise
  - 1.2 UI Orientation & Core Components
  - 1.3 First Conversation - Revenue Analysis
- Module 2: The Two-in-One Paradigm
  - 2.1 Conversational Interface → REST API Transformation
  - 2.2 Profile System - “Pro Mode” Orchestration
  - 2.3 REST API Deep Dive
  - 2.4 Airflow Integration for Batch Processing
- Module 3: Transparent Execution
  - 3.1 Live Status Updates & Execution Transparency
  - 3.2 MCP Capabilities Discovery
  - 3.3 Context Window Management
  - 3.4 Tool Customization & Preferences
  - 3.5 Execution Monitoring & Debugging
- Module 4: Strategic Intelligence
  - 4.1 Strategic Planning - The Hidden Orchestration Layer
  - 4.2 RAG Self-Learning from Champion Cases
  - 4.3 Optimization & Learning Loops
  - 4.4 Cost Tracking & Token Analytics
- Module 5: Sovereignty
  - 5.1 Multi-Provider LLM Architecture
  - 5.2 Hybrid Intelligence - Cloud + Local Models
  - 5.3 Security & Multi-User Management
- Module 6: Collaborative
  - 6.1 Marketplace Overview - The Network Effect
  - 6.2 Planner Repositories - Reusable Execution Patterns
  - 6.3 Knowledge Repositories - Shared Expertise
  - 6.4 Community Ecosystem & Ratings
- Module 7: Financial Governance
  - 7.1 Consumption Profiles & Rate Limiting
  - 7.2 Cost Analytics & Optimization Dashboard

    - 7.3 Budget Alerts & Forecasting
- Module 8: Administration
    - 8.1 User & Access Management
    - 8.2 Long-Lived Access Tokens
    - 8.3 System Configuration & Monitoring
- Module 9: Advanced Topics
    - 9.1 Flowise Integration - Low-Code Workflows
    - 9.2 Docker Deployment - Production Setup
    - 9.3 Developer Mode & Custom Integrations
    - 9.4 Installing Your Own Uderia Platform
- Module 10: User Journey Scenarios
    - 10.1 Business Analyst Journey - @ANALYST Profile
    - 10.2 Database Administrator Journey - @DBA Profile
    - 10.3 Data Quality Analyst Journey - @QUALITY Profile
    - 10.4 Compliance Officer Journey - @LOCAL Profile
    - 10.5 “Pro Mode” Power User Journey - Multi-Profile Orchestration
- Profile Ecosystem Reference
- Demo Environment Reference

Tutorial Philosophy

This tutorial series showcases Uderia’s unique value proposition: **Cloud-Level Reasoning with Zero-Trust Privacy**. Each module aligns with one of Uderia’s six core principles, ensuring comprehensive coverage of features, benefits, and differentiation.

Key Messaging Themes: - From Days to Seconds (Actionable) - From Guesswork to Clarity (Transparent) - From $$$ to ¢¢¢ (Efficient) - From Data Exposure to Data Sovereignty (Sovereignty) - From Hidden Costs to Total Visibility (Financial Governance) - From Isolated Expertise to Collective Intelligence (Collaborative)

Profile Personas Used Throughout Tutorials

A core differentiator of Uderia is the ability to **orchestrate a team of specialized AI personas** through profile switching. Throughout these tutorials, we demonstrate six key profiles that represent different user roles and use cases. Think of profiles not as configuration files, but as complete expert personas—each with their own LLM, tools, knowledge, and learned patterns.

The Profile Lineup:

1. **@ANALYST** - Business Intelligence Persona
    - **Role:** Business Analyst, Product Manager
    - **LLM:** Gemini 1.5 Flash (fast, cost-effective)
    - **Specialty:** Business insights, visualizations, executive reporting
    - **Knowledge:** Business metrics, KPIs, fitness_db business context
    - **Appears in:** Modules 1.3, 2.2, 4.1, 6.2, 10.1
2. **@DBA** - Database Administrator Persona

- **Role:** Database Admin, Data Engineer
- **LLM:** Claude Sonnet 3.5 (reliable SQL optimization)
- **Specialty:** Query optimization, schema analysis, performance tuning
- **Knowledge:** SQL best practices, Teradata optimization, schema docs
- **Appears in:** Modules 2.2, 3.2, 4.1, 8.2, 10.2

3. **@QUALITY** - Data Quality Engineer Persona
   - **Role:** Data Steward, Quality Analyst
   - **LLM:** Claude Sonnet 3.5 (thorough validation)
   - **Specialty:** Data validation, anomaly detection, compliance audits
   - **Knowledge:** Quality standards, constraints, validation rules
   - **Appears in:** Modules 2.2, 3.1, 10.3
4. **@LOCAL** - Privacy-First Persona
   - **Role:** Compliance Officer, Privacy-Conscious User
   - **LLM:** Ollama Llama 3.1 70B (on-premises, zero cloud)
   - **Specialty:** Sovereign data analysis with complete privacy
   - **Knowledge:** Same as @ANALYST but locally stored/executed
   - **Appears in:** Modules 2.2, 5.2, 5.3, 10.4
5. **@PROD** - Production Workhorse Persona
   - **Role:** Default automation profile for APIs and scheduled jobs
   - **LLM:** Claude Sonnet 3.5 (balanced reliability)
   - **Specialty:** REST API calls, Airflow DAGs, production automation
   - **Knowledge:** Cross-domain general purpose
   - **Appears in:** Modules 2.1, 2.3, 2.4, 9.1, 10.5
6. **@COST** - Budget-Conscious Persona
   - **Role:** Cost optimization, high-volume queries
   - **LLM:** Gemini 1.5 Flash (cheapest option)
   - **Specialty:** Simple lookups, batch jobs, development/testing
   - **Knowledge:** Minimal (reduces overhead)
   - **Appears in:** Modules 4.4, 7.2, 10.5 (optional)

The "Pro Mode" Concept:

Rather than being limited to one agent's perspective, Uderia users can **orchestrate an entire team of expert personas**, switching between them as naturally as consulting different colleagues. This allows one person to perform work that traditionally requires multiple specialists:

- Need SQL optimization? → Switch to **@DBA**
- Need business insights? → Switch to **@ANALYST**
- Need data validation? → Switch to **@QUALITY**
- Need privacy compliance? → Switch to **@LOCAL**
- Need to operationalize? → Switch to **@PROD**

Each profile has its own knowledge collections and learned execution patterns (planner repositories), making profile switching equivalent to accessing specialized domain expertise instantly.

Note: Detailed specifications for each profile are provided in the "Profile Ecosystem Reference" section before Module 10. They are introduced narratively in Module 2.2 and demonstrated throughout subsequent modules.

Module 1: Introduction & Quick Start

Goal: Hook viewers with immediate value demonstration and quick wins

Demo Environment Setup: - **Database:** fitness_db (Teradata) - **Schema:** 5 tables (Customers, Sales, SaleDetails, Products, ServiceTickets) - **Business Context:** Fitness equipment retail and service tracking - **MCP Tools:** base_tools, quality_tools for business user scenarios - **Profiles:** Standard profile with Teradata MCP server configured

1.1 Welcome to Uderia - The Platform Promise

Duration: 2-3 minutes
Learning Objectives: - Understand Uderia's unique positioning in the AI landscape - Recognize the six core principles - See the fundamental "before vs after" transformation

Key Topics to Cover: - [ ] The problem: Traditional AI tools force trade-offs (cloud intelligence vs data privacy) - [ ] The Uderia solution: Best of both worlds - [ ] Six core principles overview (visual diagram) - [ ] Platform overview screenshot tour

Narration Script:

```
Welcome to Uderia-the platform that finally ends the impossible choice between
cloud intelligence and data privacy.

Finally, an AI platform that doesn't force you to choose.

Uderia empowers secure, local models to perform like giants-and tames powerful
cloud models for verified compliance.

Whether on-prem or in the cloud, you get enterprise results with optimized speed
and minimal token cost.

**CLOUD-LEVEL INTELLIGENCE. ZERO-TRUST SOVEREIGNTY. EXTREME EFFICIENCY.**

This is Uderia. You. Data. Freedom.

But how do we deliver on this promise? Through six core principles that transform
how you work with enterprise data:

**CLOUD-LEVEL INTELLIGENCE** is based upon two core principles:

Cloud-level intelligence is ACTIONABLE: Go from conversational discovery to a production-ready
automated workflow in seconds. Your interactive queries can be immediately operationalized via
a REST API, eliminating the friction and redundancy of traditional data operations.
What once took data experts weeks is now at your fingertips.

And cloud-level intelligence is TRANSPARENT: Eliminate the "black box" of AI. The Uderia
Platform is built on a foundation of absolute trust, with a Live Status Window that shows
you every step of the agent's thought process. From the initial high-level plan to every
tool execution and self-correction, you have a clear, real-time view, leaving no room
```

```
for guesswork.

**ZERO-TRUST SOVEREIGNTY** is based upon two core principles:

Zero-trust sovereignty means SOVEREIGNTY: Your data, your rules, your environment. The agent
gives you the ultimate freedom to choose your data exposure strategy. Leverage the power of
hyperscaler LLMs, or run fully private models on your own infrastructure with Ollama,
keeping your data governed entirely by your rules. The agent connects to the models
you trust.

And zero-trust sovereignty is COLLABORATIVE: Transform isolated expertise into collective intel
The Intelligence Marketplace enables you to share proven execution patterns and
domain knowledge with the community, subscribe to curated collections from experts,
and fork specialized repositories for your unique needs. By leveraging community-validated
RAG collections, you reduce token costs, accelerate onboarding, and benefit from
battle-tested strategies-turning individual insights into a powerful, shared ecosystem.

**EXTREME EFFICIENCY** is based upon two core principles:

Extreme efficiency is EFFICIENT: Powered by the intelligent Fusion Optimizer, the agent
features a revolutionary multi-layered architecture for resilient and cost-effective task
execution. Through strategic and tactical planning, proactive optimization, and
autonomous self-correction, the agent ensures enterprise-grade performance and
reliability.

And extreme efficiency means FINANCIAL GOVERNANCE: Complete cost transparency and control
over your LLM spending. The agent provides real-time cost tracking, comprehensive analytics,
and detailed visibility into every token consumed. With accurate per-model pricing,
cost attribution by provider, and powerful administrative tools, you maintain full
financial oversight of your AI operations.

This isn't just another data chat tool. Uderia is a complete AI orchestration platform
that delivers enterprise results with cloud-level reasoning and zero-trust privacy.

Over the next few tutorials, you'll see exactly how Uderia transforms every aspect
of data operations-from your first conversation to production deployment at scale.

Let's dive in.
```

Screen Capture Plan: - [ ] Website hero section with tagline “Cloud-Level Reasoning. Zero-Trust Privacy.” - [ ] Six principles icons/cards appearing sequentially as mentioned - [ ] Hybrid architecture diagram (cloud planner + local executor) - [ ] Quick feature montage (API, Live Status Panel, Marketplace, Cost Dashboard) - [ ] Live platform login screen teaser

Supporting Materials: - README.md introduction - Website value proposition sections - Platform architecture diagram

---

1.2 UI Orientation & Core Components

Duration: 2-3 minutes
Learning Objectives: - Successfully log in or register - Understand the UI layout at first glance - Identify key navigation elements

Key Topics to Cover: - [ ] Registration/Login process - [ ] Platform UI overview (navigation bar, main panels) - [ ] Quick orientation: Conversations, Executions, Intelligence, Marketplace, Admin - [ ] Profile indicator and status elements

Narration Script:

```
Let's get you started with Uderia. If you're following along on the live platform
at uderia.com, you'll see the login screen. New users can register in seconds-
just an email, username, and password. Enterprise deployments integrate with your
existing SSO.

Once logged in, you're greeted by an interface designed around one goal: getting
you from question to insight as quickly as possible.

Let's orient ourselves.

**NAVIGATION:** Five key sections across the top.

**Conversations**-Your interactive workspace. Natural language queries. Real-time
agent collaboration. This is where you'll spend most of your time.

**Executions**-Mission control. Monitor tasks. Track long-running queries. Review
execution history across all automation workflows.

**Intelligence**-Where continuous improvement lives. Manage RAG collections. Review
champion cases. Track efficiency gains as the platform learns from your successes.

**Marketplace**-The community hub. Discover execution patterns. Subscribe to expert
knowledge. Contribute to collective intelligence. Network effects in action.

**Administration**-Your governance center. User management. Cost analytics. Access
tokens. System configuration. Financial oversight.

**PROFILE INDICATOR:** Top right corner.

Shows your active LLM profile-which language model and MCP tools you're using. Switch
between specialized personas with one click. Think of it as choosing which expert
consultant you're working with.

**THREE-PANEL WORKSPACE:**

**Left: Resource Panel**-Your agent's capability catalog. Available MCP tools. Prompt
templates. Data sources. Dynamically adapts based on your active profile.
```

Center: Conversation Panel-Primary interaction space. Queries. Results. Visualizations. Where the conversation happens.

Right: Live Status Panel-The transparency engine. Uderia's game-changer. Watch strategic plans unfold in real-time. See tactical decisions. Inspect raw data at every step. No black boxes. No guesswork. Complete visibility.

Everything you need. Within reach. Clean. Organized. Built for speed.

Now let's put it to work.

Screen Capture Plan: - [ ] Login screen with registration option highlighted - [ ] Successful login transition to dashboard - [ ] Navigation bar with each section highlighted as mentioned - [ ] Profile indicator zoom-in (top right corner) - [ ] Three-panel layout with annotations: Resource (left), Conversation (center), Live Status (right) - [ ] Quick hover over each panel to show dynamic content - [ ] Smooth transition to next section (query input ready)

Supporting Materials: - User Guide documentation - UI screenshots

---

1.3 Your First Conversation - From Question to Insight

Duration: 3-4 minutes
Learning Objectives: - Execute a simple query successfully - Understand query → insight → action flow - See immediate value in action

Key Topics to Cover: - [ ] Typing a natural language query - [ ] Watching the Live Status Panel (strategic plan appears) - [ ] Tool execution visualization - [ ] Receiving structured results - [ ] Understanding the response format

Narration Script:

Let's ask your first question. No SQL. No technical syntax. Just ask naturally, like you would a colleague.

We're working with a fitness equipment retail database. Type: *"What are the top 5 products by revenue this month?"*

Hit enter. Watch what happens.

LIVE STATUS PANEL ACTIVATES:

Strategic Plan appears instantly. The agent isn't blindly executing-it's thinking:

Phase 1: Identify relevant tables (Sales, Products, SaleDetails) Phase 2: Calculate revenue by product for current month Phase 3: Sort descending, return top 5

This is Uderia's transparency. No loading spinner. No wondering. You're watching a strategic blueprint unfold in real-time.

```
**TACTICAL EXECUTION:**

The agent selects its tool: `base_readQuery` (SQL execution)

Watch the SQL generate:

```sql
SELECT p.ProductName, p.Brand,
       SUM(sd.Quantity * sd.UnitPrice) as Revenue
FROM Products p
JOIN SaleDetails sd ON p.ProductID = sd.ProductID
JOIN Sales s ON sd.SaleID = s.SaleID
WHERE EXTRACT(MONTH FROM s.SaleDate) = CURRENT_MONTH
GROUP BY p.ProductName, p.Brand
ORDER BY Revenue DESC
LIMIT 5;
```

Every parameter visible. Zero guesswork.

RAW DATA RETURNS:

Click into Live Status. Inspect the exact JSON response. Complete visibility.

INTELLIGENT RENDERING:

The agent doesn't dump data. It formats results beautifully:

```
1. Treadmill Pro 3000 (FitTech) - $45,230
2. Elliptical Elite (CardioMax) - $38,900
3. Weight Bench Deluxe (IronGrip) - $32,150
4. Rowing Machine Pro (AquaFit) - $28,500
5. Spin Bike Ultra (CycleMax) - $24,100
```

Clean. Formatted. Actionable.

FOLLOW-UP:

Type: *"Show me customer demographics for buyers of that top treadmill."*

Notice what just happened. The agent remembers "that top treadmill" = Treadmill Pro 3000. It knows the table structure. Context retained.

New strategic plan:

```
Phase 1: Find customers who purchased ProductID 1047 (Treadmill Pro 3000)
Phase 2: Retrieve demographics from Customers table
Phase 3: Aggregate by location, age, visualize
```

Joining Sales → Customers automatically. You didn't specify a single table name.

THE POWER:

Question → Insight: Seconds. Insight → Follow-up: Seamless. Context: Retained.

No Python notebooks. No SQL editors. No context switching.
```

Just you, your questions, and an agent that shows its work.

THE KICKER:

Every query you just ran? Already available via REST API. Your conversational discovery is already operationalized. We'll explore this in Module 2.

ONE MORE THING:

See that profile indicator (top right)? You're using @ANALYST right now—business analyst profile.

But imagine switching to: - **@DBA** - SQL optimization expert - **@QUALITY** - Data validation specialist
- **@LOCAL** - Privacy-first, zero cloud exposure

One person orchestrating multiple expert personas. Switch like consulting different colleagues.

This is "Pro Mode." We'll explore it in depth soon.

WELCOME TO UDERIA:

Intelligence that's actionable, transparent, and built for speed.

You're not just chatting with AI. You're orchestrating specialized expertise.

Let's keep going.

```
**Screen Capture Plan:**
- [ ] Query typed into input: "What are the top 5 products by revenue this quarter?"
- [ ] Live Status Panel: Strategic plan appears with phases
- [ ] Live Status Panel: Tactical section shows tool selection and SQL generation
- [ ] Live Status Panel: Raw data response visible (expandable JSON)
- [ ] Conversation Panel: Results render as formatted table with actual data
- [ ] Follow-up query typed: "Why did Product X decline compared to last quarter?"
- [ ] New strategic plan appears showing comparative analysis
- [ ] Results with trend analysis and potential chart visualization
- [ ] Quick flash of REST API documentation showing same query structure
- [ ] End with satisfied user looking at insights

**Supporting Materials:**
- fitness_db schema documentation (5 tables: Customers, Sales, SaleDetails, Products, ServiceT
- MCP base_tools configuration (Teradata connector)
- Sample queries: revenue analysis, customer demographics, product trends
- Expected output samples with realistic fitness retail data
- MCP quality_tools for data validation demonstrations

---

## Module 2: ACTIONABLE - From Discovery to Production
**Goal:** Demonstrate seamless operationalization from conversation to automation

### 2.1 The Two-in-One Philosophy: Conversation → API
```

```
**Duration:** 3-4 minutes
**Learning Objectives:**
- Understand the unique "discovery is production" paradigm
- See how a conversational query becomes an API call
- Grasp the value of eliminating handoffs

**Key Topics to Cover:**
- [ ] Traditional workflow: Analyst → Data Engineer → API Developer (days/weeks)
- [ ] Uderia workflow: One conversation → Immediate API (seconds)
- [ ] REST API endpoint overview
- [ ] Session management concept
- [ ] Bearer token authentication basics

**Narration Script:**
```

Let's talk about what separates Uderia from every other data tool.

THE OLD WORLD:

Traditional workflow for ONE analytics question:

```
Business Analyst asks: "What are our top revenue products?"
  ↓ (2 days)
Data Analyst writes SQL in notebook
  ↓ (3 days)
Data Engineer rebuilds it in pipeline
  ↓ (5 days)
API Developer wraps endpoint
  ↓ (2 days)
DevOps deploys to production

Total: ~2 weeks, 4 handoffs, context loss
```

This friction kills innovation.

THE UDERIA WAY:

We just asked: *"What are the top 5 products by revenue this month?"*

Got answer in seconds. That was conversational discovery.

Here's the revolution:

That EXACT query is already available via REST API.

- No rebuilding
- No translation
- No handoffs
- No deployment

WATCH:

Postman → `/v1/query/submit` endpoint

```
POST https://tda.uderia.com/api/v1/query/submit
Authorization: Bearer YOUR_TOKEN

{
  "query": "What are the top 5 products by revenue this month?",
  "session_id": "your-session-id"
}
```

Same question. Same strategic plan. Same tool execution. Same results.

Or cURL:

```
curl -X POST https://tda.uderia.com/api/v1/query/submit \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"query": "What are the top 5 products by revenue this month?", "session_id": "abc123"}'
```

THE PARADIGM:

Discovery = Production.

Conversation = Automation.

No engineering team. No deployment cycle. Zero friction.

THE MULTIPLIER:

Now imagine:

1. @DBA profile → Query optimization
2. @ANALYST profile → Business visualization
3. @QUALITY profile → Data validation

All three conversations → All three APIs → Instantly.

You're not automating one query. You're operationalizing an entire team of expert personas.

THE IMPACT:

Old world: - Weeks to production - Multiple specialists required - Context lost in handoffs

Uderia: - Seconds to production - One person orchestrating profiles - Context preserved

From insight to automation: One conversation.

This is actionable intelligence.

Let's see how it works.

```
**Screen Capture Plan:**
- [ ] Traditional workflow diagram: Analyst → Engineer → API Dev → DevOps (days/weeks)
- [ ] Uderia workflow: One conversation → Immediate API (seconds)
- [ ] Live query execution showing `base_readQuery` tool call
- [ ] Switch to Postman: same query via `/v1/query/submit` endpoint
```

```
- [ ] Show cURL command with actual request/response
- [ ] Side-by-side comparison: UI results vs API results (identical)
- [ ] REST API documentation page

**Supporting Materials:**
- REST API documentation (query endpoints)
- Postman collection with pre-configured requests
- cURL command examples
- Workflow comparison diagram
- MCP tools reference (base_readQuery)

---

### 2.2 Profile System: Complete AI Personas
**Duration:** 4-5 minutes
**Learning Objectives:**
- Understand profiles as complete AI personas (LLM + Tools + Knowledge + Patterns)
- See how RAG Knowledge Collections integrate into profiles
- Learn how Planner Repositories provide learned execution patterns
- Master profile switching as persona switching

**Key Topics to Cover:**
- [ ] What is a profile? (Four-component persona architecture)
- [ ] Component 1: LLM Provider configuration
- [ ] Component 2: MCP Tools/Prompts (capabilities)
- [ ] Component 3: RAG Knowledge Collections (domain expertise)
- [ ] Component 4: Planner Repositories (execution patterns)
- [ ] Profile tags and switching (@DBA, @ANALYST, @LOCAL)

**Narration Script:**
```

Now let's talk about profiles—but not as simple configuration files. Profiles are something far more powerful: they're complete AI personas with memory, tools, knowledge, and learned behaviors.

When you switch profiles in Uderia, you're not just changing which LLM you're using. You're switching to a completely different intelligent agent with its own expertise, capabilities, and experience.

Let me show you what I mean.

THE FOUR PILLARS OF A PROFILE:

A profile bundles four critical components into a cohesive persona:

1. LLM Provider - Which language model powers the reasoning Think: Claude Opus for complex analysis, Gemini Flash for cost efficiency, Ollama for complete privacy.

2. MCP Tools & Prompts - Which capabilities are available Think: Database tools (base_readQuery, base_tableList), Quality tools (qlty_validateData), Admin tools (dba_databaseVersion), Platform tools (TDA_FinalReport, TDA_Charting).

3. RAG Knowledge Collections - What domain expertise the agent has access to Think: SQL optimization guides, database documentation, data quality standards, business logic rules, company-

specific knowledge.

4. Planner Repositories - Which execution patterns the agent has mastered Think: Champion cases from successful queries—proven strategies for revenue analysis, customer segmentation, performance optimization, all learned from past successes.

Together, these four components create a complete persona. Let me show you with a real example.

PERSONA EXAMPLE: @DBA Profile

Navigate to Setup → Profiles. Let's look at the "Database Administrator" profile with tag @DBA.

Component 1: LLM Provider - Model: Claude Sonnet 3.5 (reliable, balanced performance) - Cost tier: Mid-range (~$3 per 1M input tokens)

Component 2: MCP Tools - Teradata MCP Server enabled - Tools available: - base_databaseList (list all databases) - base_tableList (list tables in a database) - base_tableDDL (get table structure) - base_readQuery (execute SQL) - dba_databaseVersion (system version info) - qlty_profileColumn (data profiling) - qlty_validateData (quality checks)

Component 3: RAG Knowledge Collections Click "Knowledge" tab—three collections attached: 1. "Teradata SQL Optimization Guide" (23 documents) 2. "fitness_db Schema Documentation" (5 documents) 3. "DBA Best Practices" (12 documents)

3. "Database Administration Runbooks" (12 documents)
    - Troubleshooting procedures
    - Maintenance scripts
    - Performance tuning guides

When you ask this profile a question, it doesn't just have access to database tools— it has access to your organization's accumulated DBA knowledge.

Ask: "What's the best way to optimize a query joining Sales and SaleDetails for large date ranges?"

The agent retrieves relevant content from your SQL Optimization Guide: "For large date ranges, use partition elimination with explicit date predicates. Consider creating a covering index on SaleDate + frequently joined columns."

This isn't generic LLM knowledge. This is YOUR organization's expertise, embedded into the profile.

Component 4: Planner Repositories Now click the "Planners" tab.

You see the "DBA_Champion_Cases" repository with 47 successful execution patterns: - "Monthly Sales Aggregation" (champion case, 94/100 efficiency) - "Customer Purchase History Lookup" (champion case, 91/100 efficiency) - "Product Revenue Analysis with Trends" (champion case, 96/100 efficiency)

Each case contains: - The original query that succeeded - The strategic plan that was used - The exact tool execution sequence (e.g., base_readQuery with optimized SQL) - Token usage and cost data - Efficiency metrics

When you ask the @DBA profile a new question, it searches these repositories for similar past successes. If it finds one, it uses that proven pattern as a few-shot example to guide the new execution.

Ask: “Show me top revenue products by category this quarter.”

The Live Status Panel shows:

```
  RAG-RETRIEVED CONTEXT:

Similar patterns found in DBA_Champion_Cases:
1. "Product Revenue Analysis with Trends" (similarity: 0.92)
   - Strategy: Single aggregation with RANK() window function
   - Tools: base_readQuery → TDA_FinalReport
   - SQL pattern: SUM(Quantity * UnitPrice) GROUP BY with PARTITION BY
   - Efficiency: 96/100 (champion case)
```

The agent learned from past success. It’s using the proven champion pattern, not reinventing the approach.

This is profile-bound learning. The @DBA persona gets smarter every time a DBA uses it.

PERSONA COMPARISON: @ANALYST Profile

Now switch to the “Business Analyst” profile (@ANALYST). Watch what changes.

LLM Provider: Gemini 1.5 Flash (fast, cheap—$0.075 per 1M input tokens)

MCP Tools: Same Teradata tools, but ALSO: - TDA_Charting (visualization generation) - TDA_LLMTask (natural language analysis)

RAG Knowledge: Completely different collections: 1. “Business Metrics Definitions” (15 documents) - KPI calculation formulas - Reporting standards - Business terminology

2. “fitness_db Business Context” (8 documents)
    - Product categories explained
    - Customer segmentation rules
    - Sales cycle patterns

Planner Repositories: Different patterns: - “Monthly_Executive_Reports” (23 cases focused on summarization) - “Customer_Segmentation_Analysis” (17 cases with visualization) - “Trend_Analysis_Patterns” (31 cases with time-series queries)

Same platform. Same database. Completely different persona.

Ask the @ANALYST profile: “Show me top revenue products by category this quarter.”

The strategic plan is different: - Phase 1: Query data with business-friendly column aliases - Phase 2: Calculate percentage contribution by category - Phase 3: Generate bar chart visualization - Phase 4: Summarize insights in executive language

It uses TDA_Charting to create a visual. It frames results in business terms, not technical output. Because this persona has different knowledge, different patterns, and different priorities.

PROFILE SWITCHING = PERSONA SWITCHING:

This is the profound insight: switching profiles isn’t about changing infrastructure. It’s about switching intelligent personas.

Need database optimization? Switch to @DBA. - Tools: Database admin tools - Knowledge: SQL optimization, schema docs - Patterns: Performance-tuned queries - Behavior: Technical, optimization-focused

Need business insights? Switch to @ANALYST. - Tools: Charting, summarization - Knowledge: Business metrics, KPI definitions - Patterns: Visualization-heavy reports - Behavior: Executive-friendly, insight-focused

Need complete privacy? Switch to @LOCAL. - LLM: Ollama (on-premises, zero cloud) - Tools: Same capabilities, local execution - Knowledge: Same collections, private retrieval - Patterns: Same strategies, sovereign infrastructure

One platform. Multiple specialized personas. Each with their own expertise.

CREATING A NEW PERSONA:

Let's create a custom profile from scratch.

Setup → Profiles → New Profile

Name: "Data Quality Auditor" Tag: @QUALITY

Step 1: Choose LLM Provider: Claude Sonnet 3.5 (reliable for analysis)

Step 2: Select MCP Server Teradata MCP → This gives us database access

Step 3: Attach Knowledge Collections Click "Add Knowledge Collection" - Select: "Data Quality Standards" (your org's validation rules) - Select: "fitness_db Constraints Documentation" (expected value ranges)

Step 4: Attach Planner Repositories Click "Add Planner Repository" - Select: "Quality_Validation_Patterns" (champion cases for data validation queries)

Step 5: Configure Tool Preferences Enable specific tools: - qlty_profileColumn (profiling) - qlty_validateData (validation) - base_readQuery (data sampling)

Save profile.

Now @QUALITY is a complete persona: - Reasoning: Claude Sonnet - Capabilities: Quality tools + database access - Expertise: Your org's quality standards - Experience: Proven validation patterns

Ask @QUALITY: "Validate the Customers table for data quality issues."

The agent: 1. Retrieves validation rules from your "Data Quality Standards" collection 2. Uses proven patterns from "Quality_Validation_Patterns" repository 3. Executes qlty_profileColumn and qlty_validateData tools 4. Reports findings against YOUR organization's standards

This isn't generic data profiling. This is a specialized quality auditor persona with your company's rules and proven validation workflows.

MARKETPLACE INTEGRATION:

Here's where it gets powerful for organizations.

Go to Marketplace → Knowledge Repositories.

Browse community-shared collections: - “Teradata SQL Best Practices” (4,200 subscribers) - “Retail Analytics Patterns” (1,800 subscribers) - “Customer Churn Analysis Techniques” (950 subscribers)

Find one relevant: “Retail Analytics Patterns”

Click “Subscribe” → Select your @ANALYST profile → Confirm.

That collection is now attached to your @ANALYST persona. Instant expertise expansion.

Or browse Marketplace → Planner Repositories: - “E-commerce Revenue Optimization” (3,100 subscribers, 127 champion cases) - “Customer Lifetime Value Analysis” (2,400 subscribers, 89 cases)

Subscribe “E-commerce Revenue Optimization” to your @ANALYST profile.

Now your @ANALYST persona has 127 additional proven patterns from the community’s best executions.

Your profiles don’t just grow from your own usage—they grow from collective intelligence.

ORCHESTRATING A TEAM OF EXPERTS:

Here’s the ultimate power move. You’re not working with ONE agent. You’re orchestrating a TEAM of specialized experts—and you can switch between them instantly.

Watch this workflow:

Scenario: Complex Business Intelligence Project

You need to analyze Q4 revenue performance, but the workflow requires multiple types of expertise. In a traditional organization, this would involve: - Database Admin to optimize queries - Data Quality Engineer to validate data - Business Analyst to generate insights - Executive to review and approve

That’s four people. Four handoffs. Days of coordination.

With Uderia? You ARE all four experts. Watch.

Step 1: Start with @DBA - Database Optimization

Type: “@DBA What’s the most efficient way to query Q4 revenue across all product categories?”

The @DBA persona: - Uses SQL optimization knowledge from your Teradata guides - Applies proven champion patterns for revenue aggregation - Generates optimal SQL with proper indexing strategy - Returns: Optimized query structure

Result: “Use partitioned query with covering index on SaleDate, ProductType. Estimated execution time: 1.2s for 2.3M rows.”

Step 2: Switch to @QUALITY - Data Validation

Type: “@QUALITY Validate the data quality for Q4 sales data before analysis.”

The @QUALITY persona: - Uses your organization’s data quality standards - Executes qlty_profileColumn on Sales table - Applies validation rules from your quality knowledge base - Returns: Data quality report

Result: “Q4 data validated: 99.7% completeness, 2 outliers flagged in SaleDetails (likely data entry errors), recommend excluding SaleID 45023, 45089.”

Step 3: Switch to @ANALYST - Business Intelligence

Type: "@ANALYST Generate Q4 revenue analysis by category with trends and visualizations."

The @ANALYST persona: - Uses business metrics definitions for proper KPI calculation - Applies executive reporting patterns from champion cases - Generates TDA_Charting visualizations - Frames insights in business language

Result: Beautiful charts + "Cardio equipment down 18% YoY due to market saturation. Accessories up 34% driven by home fitness trend. Recommend diversification strategy."

Step 4: Switch to @LOCAL - Sensitive Deep Dive

Type: "@LOCAL Show me individual customer purchase patterns for our top 10 customers."

The @LOCAL persona: - Uses Ollama (local model, zero cloud exposure) - Same MCP tools, but execution stays on-premises - Customer PII never leaves your infrastructure - Returns: Detailed customer analysis with complete privacy

Result: Customer-level insights without compromising sovereignty.

Four personas. Four specializations. One seamless workflow. Under 5 minutes.

This is what we call "Pro Mode"—you're not limited to one agent's perspective. You orchestrate an entire team of specialized experts, each with their own knowledge base and proven execution patterns, switching between them as naturally as asking a colleague for their input.

No coordination overhead. No context lost in handoffs. No delays.

Just instant access to specialized intelligence, perfectly suited for each phase of work.

THE PROFILE PARADIGM SHIFT:

Forget "configuration management." Profiles are intelligent personas:

- @DBA knows SQL optimization and has proven DBA workflows
- @ANALYST speaks business language and has visualization patterns
- @QUALITY enforces your org's standards and has validation playbooks
- @LOCAL preserves sovereignty while maintaining same capabilities

Switch profiles. Switch personas. Orchestrate experts. Work in Pro Mode.

And every profile gets smarter: - RAG captures successful queries as champion cases - Knowledge collections grow with new documentation - Marketplace subscriptions add community wisdom - Planner repositories expand with every execution

This is profile-bound continuous improvement. Your personas evolve.

COLLABORATIVE INTELLIGENCE AT SCALE:

In traditional organizations, expertise is siloed: - The DBA knows database optimization but not business context - The analyst knows metrics but not data quality rules - The quality engineer knows validation but not SQL tuning

In Uderia, YOU have access to ALL specialized personas, each with complete domain expertise, and you can orchestrate them together in a single workflow.

This isn't just faster. This is a fundamentally different way of working.

One person. Multiple expert personas. Instant switching. Zero friction.

From individual contributor to orchestrator of specialized intelligence.

From "I need to ask the DBA" to "Let me switch to @DBA persona."

From days of coordination to seconds of profile switching.

This is Uderia's profile system. Not just settings—complete AI personas with memory, tools, knowledge, and experience.

Infrastructure as personas. Intelligence as a package. Expertise as a profile.

And YOU as the orchestrator. God mode activated.

```
**Screen Capture Plan:**
- [ ] Setup → Profiles landing page showing multiple personas
- [ ] @DBA Profile detail view - Four tabs visible: Config, Tools, Knowledge, Planners
- [ ] Config tab: Claude Sonnet 3.5 selected
- [ ] Tools tab: List of MCP tools (base_*, dba_*, qlty_*)
- [ ] Knowledge tab: Three collections attached with document counts
  - Teradata SQL Optimization Guide (23 docs)
  - fitness_db Schema Documentation (5 docs)
  - Database Administration Runbooks (12 docs)
- [ ] Click into one collection: show document list with titles
- [ ] Planners tab: DBA_Champion_Cases repository (47 cases)
- [ ] Case detail view: "Product Revenue Analysis" champion case
  - Original query
  - Strategic plan
  - Tool execution trace with base_readQuery
  - Efficiency score: 96/100
- [ ] Query with @DBA: "Optimize query joining Sales and SaleDetails"
- [ ] Live Status Panel: RAG context shows retrieved SQL optimization knowledge
- [ ] Switch to @ANALYST profile - watch UI update
- [ ] @ANALYST Tools tab: Additional TDA_Charting, TDA_LLMTask
- [ ] @ANALYST Knowledge tab: Different collections (Business Metrics, Business Context)
- [ ] @ANALYST Planners tab: Different repositories (Executive Reports, Segmentation)
- [ ] Same query with @ANALYST: Different strategic plan (includes visualization phase)
- [ ] Results include chart generation
- [ ] Create new profile: "Data Quality Auditor" @QUALITY
- [ ] Step through four-component setup
- [ ] Marketplace → Knowledge Repositories browser
- [ ] "Retail Analytics Patterns" collection with subscriber count
- [ ] Subscribe to @ANALYST profile
- [ ] Marketplace → Planner Repositories browser
- [ ] "E-commerce Revenue Optimization" with 127 cases
- [ ] Subscribe to @ANALYST profile
- [ ] @ANALYST profile now shows expanded Knowledge + Planners counts
```

```
- [ ] Side-by-side comparison: @DBA vs @ANALYST vs @QUALITY personas

**Supporting Materials:**
- Profile architecture diagram (four-component model)
- RAG Knowledge Collections documentation
- Planner Repositories guide
- MCP server configuration guide
- Marketplace subscription workflow
- Profile tag reference (@DBA, @ANALYST, @QUALITY, @LOCAL, @PROD, @COST, @OPUS)
- Classification mode comparison (Light vs Full)
- Example persona configurations (DBA, Analyst, Quality Auditor, Data Scientist)
- Champion case efficiency scoring methodology
- Knowledge collection upload guide
- Planner repository case capture process

---

### 2.3 REST API Power: Operationalize Everything
**Duration:** 4-5 minutes
**Learning Objectives:**
- Master REST API capabilities
- Understand async task pattern
- Learn endpoint categories

**Key Topics to Cover:**
- [ ] API endpoint categories (session, query, task, config, RAG, analytics)
- [ ] Async submit + poll pattern
- [ ] Session management (create, reuse, delete)
- [ ] Long-lived access tokens
- [ ] Profile override via API

**Narration Script:**
```

Now let's see the REST API that makes everything programmable.

SIX ENDPOINT CATEGORIES:

1. **Session Management** - Create/delete conversation contexts
2. **Query Execution** - Submit/poll/retrieve
3. **Task Management** - Cancel, check status
4. **Configuration** - Switch profiles, control MCP
5. **RAG Operations** - Manage knowledge collections
6. **Analytics** - Track costs, tokens, efficiency

AUTHENTICATION:

Admin Panel → Generate long-lived access token (90 days or permanent).

Copy once. Never shown again. Store securely.

WORKFLOW:

Step 1: Create Session

```
POST /api/v1/sessions/create
{
  "session_name": "Revenue Analysis",
  "profile_override": "@PROD"
}

→ {"session_id": "abc-123"}
```

Isolated conversation context. All queries share memory.

Step 2: Submit Query (Async)

```
POST /api/v1/query/submit
{
  "query": "What are top 5 products by revenue this month?",
  "session_id": "abc-123"
}

→ {"task_id": "task_789"}
```

No immediate results. Agent works in background.

Step 3: Poll Status

```
GET /api/v1/query/status/task_789

→ {"status": "running", "progress": "Phase 1: Database query"}

(poll again)

→ {"status": "completed", "result_ready": true}
```

Step 4: Retrieve Results

```
GET /api/v1/query/result/task_789

→ {
  "query": "...",
  "answer": "...",
  "tool_calls": [
    {"tool": "base_readQuery", "args": {...}, "result": [...]}
  ],
  "tokens": {"input": 1250, "output": 320},
  "cost": 0.0042
}
```

Everything you'd see in UI—as JSON.

Answer + Tool executions + Token counts + Costs.

Complete transparency via API.

Step 5: Follow-Up (Context Preserved)

```
POST /api/v1/query/submit
{
  "query": "Show customer demographics for that top product",
  "session_id": "abc-123"
}
```

Agent remembers previous query. Context maintained across API calls.

PROFILE OVERRIDE:

```
POST /api/v1/query/submit
{
  "query": "Complex analytical question",
  "session_id": "abc-123",
  "profile_override": "@OPUS"
}
```

This query runs on Claude Opus. Next query returns to default profile.

TASK CANCELLATION:

```
POST /api/v1/tasks/cancel/task_789
```

Gracefully stops long-running execution.

PRODUCTION USE CASES:

- Batch processing (Airflow)
- Chatbot integrations (Flowise)
- Custom dashboards (live insights)
- CI/CD validation queries
- Scheduled overnight reports

Your conversational intelligence—fully programmable.

Production-grade AI orchestration.

```
**Screen Capture Plan:**
- [ ] API documentation landing page with six categories
- [ ] Admin panel: Generate long-lived access token
- [ ] Postman collection: Session creation request/response
- [ ] Postman: Query submission (async) → task_id returned
- [ ] Postman: Status polling (running → completed)
- [ ] Postman: Result retrieval with full JSON response
- [ ] Show tool_calls array with `base_readQuery` execution details
- [ ] Follow-up query demonstrating context preservation
- [ ] Profile override example (@OPUS)
- [ ] Task cancellation request
```

```
**Supporting Materials:**
- Complete REST API documentation
- Postman collection (pre-configured)
- Authentication guide (JWT vs access tokens)
- Async pattern best practices
- Error handling reference
- Rate limiting documentation

**Supporting Materials:**
- REST API complete guide
- Postman collection
- Python/JavaScript SDK examples

---

### 2.4 Airflow Integration: Production Automation
**Duration:** 3-4 minutes
**Learning Objectives:**
- Understand batch automation use case
- Learn DAG structure for Uderia integration
- See production-ready patterns

**Key Topics to Cover:**
- [ ] Apache Airflow integration overview
- [ ] Example DAG walkthrough
- [ ] Session reuse pattern
- [ ] Profile override for specialized workloads
- [ ] Async polling pattern for long-running queries

**Narration Script:**
```

Let's put the REST API into production with Apache Airflow—industry-standard workflow orchestration.

Uderia ships with production-ready DAG examples. Deploy immediately.

AIRFLOW UI: "TDA Execute Questions" DAG - Runs analytical queries daily at 6 AM.

CODE STRUCTURE: `tda_00_execute_questions.py`

1. SESSION REUSE PATTERN

Create ONE session for all queries:

```python
def create_tda_session(**context):
    response = requests.post(
        f"{TDA_BASE_URL}/api/v1/sessions/create",
        headers={"Authorization": f"Bearer {TDA_ACCESS_TOKEN}"},
        json={"session_name": f"Airflow Batch {context['ds']}"}
    )
    session_id = response.json()["session_id"]
```

```
    context['ti'].xcom_push(key='tda_session_id', value=session_id)
```

Session ID stored in XCom. All tasks share context. Query 2 references Query 1—conversational follow-ups in batch mode.

2. ASYNC SUBMIT + POLL

```
def submit_query(query_text, session_id):
    # Submit
    response = requests.post(
        f"{TDA_BASE_URL}/api/v1/query/submit",
        json={
            "query": query_text,
            "session_id": session_id,
            "profile_override": "@PROD"
        }
    )
    task_id = response.json()["task_id"]

    # Poll until complete
    while True:
        status = requests.get(
            f"{TDA_BASE_URL}/api/v1/query/status/{task_id}"
        ).json()

        if status["status"] == "completed":
            break
        elif status["status"] == "failed":
            raise Exception(f"Query failed: {status['error']}")

        time.sleep(5)

    # Retrieve results
    return requests.get(
        f"{TDA_BASE_URL}/api/v1/query/result/{task_id}"
    ).json()
```

No timeouts. No blocking. Graceful long-running query handling.

3. PROFILE OVERRIDE PER TASK

Morning batch: `@COST` (Gemini Flash, cheap) Executive reports: `@OPUS` (Claude Opus, accurate)

Same infrastructure. Different profiles per workload.

4. ERROR HANDLING + RETRIES

```
task = PythonOperator(
    task_id='query_revenue_analysis',
    python_callable=submit_query,
    op_kwargs={'query_text': 'What are monthly revenue trends?'},
    retries=3,
```

```
    retry_delay=timedelta(minutes=2)
)
```

Airflow auto-retries failures. MCP hiccup? LLM timeout? Handled.

5. SCHEDULING

```
dag = DAG(
    'tda_execute_questions',
    schedule_interval='0 6 * * *',  # Daily 6 AM
    catchup=False
)
```

Cron scheduling. Manual trigger. API trigger. Upstream data trigger. Full flexibility.

6. RESULTS DISTRIBUTION

```
def store_results(result, **context):
    s3.put_object(
        Bucket='analytics-reports',
        Key=f"daily_revenue_{context['ds']}.json",
        Body=json.dumps(result)
    )
```

Uderia: Intelligence. Airflow: Distribution.

WATCH IT RUN:

```
Task 1: Create session  (XCom: session_id)
Task 2: Query "Top revenue products"  (base_readQuery → fitness_db)
Task 3: Query "Customer demographics"  (context from Task 2)
Task 4: Store to S3
```

Four tasks. One conversational workflow. Production-grade.

THE SHIFT:

Old: Notebook explorations → Manual engineering → Separate deployment

Uderia: Conversation → API → Airflow orchestration

Same agent. Same queries. Production reliability.

Conversational AI meets enterprise workflows.

```
**Screen Capture Plan:**
- [ ] Airflow UI: DAG graph view showing task dependencies
- [ ] Code editor: `tda_00_execute_questions.py` walkthrough
- [ ] Session creation task code
- [ ] Async submit + poll code pattern
- [ ] Profile override configuration
- [ ] Error handling and retry configuration
- [ ] Schedule interval setting (cron)
- [ ] Manual DAG trigger
```

```
- [ ] Live task execution: green checks appearing
- [ ] Task logs showing API calls and responses
- [ ] XCom variables showing session_id propagation
- [ ] Results storage task completing

**Supporting Materials:**
- docs/Airflow complete documentation
- Example DAG files (tda_00_execute_questions.py)
- Airflow setup guide
- Best practices document
- Error handling patterns
- Production deployment checklist

**Supporting Materials:**
- docs/Airflow documentation
- Example DAG files
- Production deployment guide

---

## Module 3: TRANSPARENT - Eliminate the Black Box
**Goal:** Build trust through complete visibility into agent reasoning

### 3.1 Live Status Panel: Watch the Agent Think
**Duration:** 3-4 minutes
**Learning Objectives:**
- Understand real-time agent reasoning visibility
- Learn to read strategic plans
- Recognize tactical decisions and tool selections

**Key Topics to Cover:**
- [ ] Live Status Panel components (Strategic Plan, Tactical Execution, Raw Data)
- [ ] Streaming updates via Server-Sent Events
- [ ] Phase-by-phase plan breakdown
- [ ] Tool selection rationale display
- [ ] Self-correction events

**Narration Script:**
```

Welcome to the feature that changes AI trust—the Live Status Panel.

THE BLACK BOX PROBLEM:

Most AI tools: - You ask → You wait → You get answer - What happened between? Unknown. - Accurate data? Hallucination? Guesswork.

Uderia eliminates guesswork. Radical transparency.

SCENARIO: Customer Support Director

Complaints piling up. Need data.

Ask: *"Compare service ticket resolution times across product categories and identify quality issues."*

Complex query: - Join ServiceTickets + Products - Calculate time deltas - Group by category - Infer quality patterns

Hit enter. Watch right panel.

STRATEGIC PLAN APPEARS:

```
Phase 1: Identify relevant tables (ServiceTickets, Products)
Phase 2: Calculate average resolution times by ProductType
Phase 3: Identify high-ticket products with slow resolution
Phase 4: Generate comparative analysis report
```

Not a loading spinner. A strategic blueprint.

Agent shows reasoning BEFORE executing. Validate approach before burning tokens.

TACTICAL EXECUTION:

Phase 1:

```
Tool: base_tableList
Arguments: {"database": "fitness_db"}
Rationale: "Verify ServiceTickets and Products tables exist"
```

Exact tool. Exact arguments. Exact reason. Zero ambiguity.

Raw Data (click to expand):


```
{
  "tables": ["Customers", "Products", "Sales", "SaleDetails", "ServiceTickets"],
  "database": "fitness_db"
}
```


Complete visibility into agent's data sources.

Phase 2:

```
Tool: base_readQuery
SQL:
  SELECT p.ProductType,
         AVG(CAST(st.ResolutionDate - st.TicketDate AS INT)) as AvgResolutionDays
  FROM fitness_db.ServiceTickets st
  JOIN fitness_db.Products p ON st.ProductID = p.ProductID
  WHERE st.ResolutionDate IS NOT NULL
  GROUP BY p.ProductType
  ORDER BY AvgResolutionDays DESC
Rationale: "Calculate average resolution time by product type"
```

Exact SQL visible. Spot logic issues immediately. No blind trust.

Raw Results:


```
[
  {"ProductType": "Cardio", "AvgResolutionDays": 8.3},
```

```
  {"ProductType": "Strength", "AvgResolutionDays": 5.1},
  {"ProductType": "Accessories", "AvgResolutionDays": 2.7}
]
```

Phase 3:

```
Tool: TDA_LLMTask
Task: "Analyze which categories show quality issues based on ticket volume + resolution time"
Data: [resolution data + ticket counts]
Rationale: "Need LLM reasoning for pattern identification beyond metrics"
```

Delegated analytical reasoning. Decision made explicitly. Not hidden.

SELF-CORRECTION (The Trust Builder):

Agent tries wrong table:

```
Tool: base_readQuery
SQL: ...FROM ProductReturns...
  Error: "Table fitness_db.ProductReturns does not exist"
```

RECOVERY:

Live Status Panel shows:

```
  Error detected. Replanning.
Strategy: Use ServiceTickets as quality proxy (ProductReturns unavailable)
New approach: Query ServiceTickets grouped by ProductID for complaint patterns
```

```
Tool: base_readQuery (corrected)
SQL: ...FROM ServiceTickets GROUP BY ProductID...
  Success
```

WHAT JUST HAPPENED:

Agent failed → Recognized failure → Replanned → Succeeded

All visible. All transparent.

CONTRAST:

Traditional AI: Generic error or hallucinated answer Uderia: Think → Execute → Detect → Self-correct (all visible)

THE IMPACT:

Live Status Panel = Radical visibility

Every decision. Every tool call. Every data source. Every recovery.

No black boxes. No guesswork.

Complete trust through complete transparency.

```
**Screen Capture Plan:**
- [ ] Complex query input: "Compare service ticket resolution times..."
```

```
- [ ] Strategic Plan appears with 4 phases listed
- [ ] Tactical section: Phase 1 executing with `base_tableList` tool
- [ ] Raw Data section: Click to expand JSON response
- [ ] Tactical section: Phase 2 executing with `base_readQuery` showing full SQL
- [ ] Raw Data: Query results JSON display
- [ ] Tactical section: Phase 3 with `TDA_LLMTask` delegation
- [ ] Error scenario: Wrong table name → Error message appears
- [ ] Recovery strategy displayed in Live Status
- [ ] Corrected query execution succeeds
- [ ] Final results rendering in Conversation Panel

**Supporting Materials:**
- Live Status Panel documentation
- Complex query examples
- Self-correction scenarios
- Error handling guide
- Tool execution logs

**Supporting Materials:**
- Live Status Panel documentation
- Example complex queries
- Self-correction examples

---

### 3.2 Capabilities Discovery: Know What's Possible
**Duration:** 2-3 minutes
**Learning Objectives:**
- Explore available MCP tools
- Understand prompt library organization
- Learn resource enumeration

**Key Topics to Cover:**
- [ ] Capabilities Panel overview
- [ ] MCP Tools tab (automatic discovery)
- [ ] Prompts tab (categorized library)
- [ ] Resources tab (data source visibility)
- [ ] Dynamic updates on configuration changes

**Narration Script:**
```

Trust requires knowing what the agent can do.

Capabilities Panel = Complete visibility into tools, prompts, resources.

LEFT PANEL: Three tabs

1. TOOLS TAB

Complete catalog of MCP tools (current profile).

Organized by category:

Database Operations: - `base_databaseList` - List all databases - `base_tableList` - List tables in database - `base_tableDDL` - Get table DDL structure - `base_readQuery` - Execute SELECT queries

Data Quality: - `qlty_dataProfile` - Profile data quality metrics - `qlty_findAnomalies` - Detect statistical anomalies - `qlty_validateConstraints` - Check constraint violations

System Administration: - `dba_databaseVersion` - Get system version - `dba_systemHealth` - Check resource utilization - `dba_userActivity` - Monitor user sessions

Each tool shows: 1. Exact name agent uses 2. Clear description 3. Arguments (required vs optional - hover for details)

Not documentation. Real-time inventory of current capabilities.

2. PROMPTS TAB

Pre-built complex workflows. Invoke like specialized team members.

Business Intelligence:

- `base_databaseBusinessDesc` - Business context for database
  - Args: database_name (required), detail_level (optional)
  - Returns: Purpose, entities, use cases
- `base_tableBusinessDesc` - Business meaning of table
  - Args: database_name, table_name (required)
  - Returns: Purpose, key columns, business rules

Data Quality:

- `qlty_databaseQuality` - Comprehensive quality assessment
  - Args: database_name (required), include_profiling (optional)
  - Returns: Scores, issues, recommendations

Administration:

- `dba_databaseHealthAssessment` - Full system health report
  - Args: include_recommendations (optional)
  - Returns: Resource usage, performance, alerts
- `dba_userActivityAnalysis` - User behavior patterns
  - Args: time_range (optional, default: 24h)
  - Returns: Active users, query volumes, resource usage

Instead of complex multi-step queries, invoke prompts that know the workflow.

3. RESOURCES TAB

Data sources + knowledge repositories:

- Database connections (e.g., fitness_db on Teradata)
- API endpoints (if configured)
- Knowledge collections (uploaded docs)
- External tools (if integrated)

DYNAMIC CAPABILITY UPDATES:

Add new capability:

Setup → MCP Servers → Add New Server

Example: PostgreSQL for cross-database analysis

```
Name: "Analytics PostgreSQL"
Host: analytics-db.company.com
Port: 5432
Path: /mcp
```

Save. Add to profile.

Capabilities Panel refreshes (auto or manual).

New tools appear: - `postgres_queryExecute` - `postgres_schemaIntrospect` - `postgres_tableSample`

Agent immediately knows. No restart. Dynamic discovery.

Now ask: *"Compare revenue trends in Teradata production vs PostgreSQL analytics models."*

Agent sees both: - `base_readQuery` (Teradata) - `postgres_queryExecute` (PostgreSQL)

Orchestrates cross-database analysis automatically.

CAPABILITY FILTERING:

Disable expensive tool:

Setup → Capabilities Management → Tools Find `base_tableDDL` → Toggle OFF

Capabilities Panel: Tool turns gray ("Disabled")

Ask: *"Show me Products table structure."*

Agent uses alternatives: - `qlty_dataProfile` - INFORMATION_SCHEMA query

`base_tableDDL` unavailable → Agent adapts.

Re-enable → Back in action.

THE POWER:

Dynamic capability management. Real-time control. No code.

Capabilities Panel = Window into agent potential.

Every tool. Every prompt. Every resource. Visible. Managed.

Know your agent. Trust your agent. Control your agent.

```
**Screen Capture Plan:**
- [ ] Capabilities Panel (left side) with three tabs
- [ ] Tools tab: Expand "Database Operations" category showing base_* tools
- [ ] Tool hover showing arguments and descriptions
- [ ] Expand "Data Quality" category showing qlty_* tools
- [ ] Expand "System Administration" showing dba_* tools
- [ ] Switch to Prompts tab
```

```
- [ ] Expand "Business Intelligence" showing base_databaseBusinessDesc, base_tableBusinessDesc
- [ ] Prompt detail view showing arguments (required vs optional)
- [ ] Expand "Data Quality" showing qlty_databaseQuality
- [ ] Expand "Administration" showing dba_* prompts
- [ ] Switch to Resources tab showing database connections
- [ ] Setup → MCP Servers → Add New Server (PostgreSQL example)
- [ ] Capabilities Panel refresh showing new postgres_* tools
- [ ] Capability filtering: Disable base_tableDDL
- [ ] Tool turns gray in Capabilities Panel
- [ ] Query execution using alternative tool
- [ ] Re-enable tool demonstration

**Supporting Materials:**
- MCP Tools complete reference
- Prompt library documentation
- Dynamic capability management guide
- MCP server configuration examples
- Tool filtering best practices

**Supporting Materials:**
- MCP Tools documentation
- Prompt library reference
- Capability discovery guide

---

### 3.3 Context Management: Control Agent Memory
**Duration:** 3-4 minutes
**Learning Objectives:**
- Master turn activation/deactivation
- Understand context purge vs selective memory
- Learn query replay functionality

**Key Topics to Cover:**
- [ ] Turn-level context control (checkboxes)
- [ ] Full Context vs Turn Summaries modes
- [ ] Context purge for memory reset
- [ ] Query replay for alternative approaches
- [ ] Context indicator real-time status

**Narration Script:**
```

Most underappreciated AI feature: Context management—controlling what the agent remembers.

Uderia: Surgical precision over agent memory.

TURN HISTORY PANEL:

Session History (below conversation). Every query-response = “turn” with checkbox.

Current conversation:

```
Turn 1:   "What are the top 5 products by revenue?"
Turn 2:   "Show customer demographics for those products"
Turn 3:   "Which states have the highest sales?"
Turn 4:   "What's the average ticket resolution time?"
```

All checked = All active in context.

SELECTIVE MEMORY:

Uncheck Turn 1.

Context indicator updates: “3 turns active”

Ask: *“What are the top selling products this week?”*

Agent treats as fresh question. Turn 1 deactivated = Not referenced.

Turns 2-4 still available if needed.

WHY THIS MATTERS:

Exploration mistake: - Asked about “Product X” - Meant “Product Y”

Traditional AI: Carries mistake through entire conversation

Uderia: Uncheck wrong turn → Never happened → Clean continuation

CONTEXT MODES:

Top bar toggle: “Full Context” vs “Turn Summaries”

Full Context (default): - Every active turn’s complete text → LLM - High token usage - Maximum accuracy

Turn Summaries: - Only summaries sent - Lower token cost
- Slightly less detail

Switch to Turn Summaries. Ask: *“What patterns do you see across our conversation?”*

Agent receives:

```
- Turn 1 summary: "Top products by revenue"
- Turn 2 summary: "Customer demographics request"
- Turn 3 summary: "Geographic sales distribution"
- Turn 4 summary: "Service ticket resolution times"
```

Reasoning maintained. 70% fewer tokens.

Long exploratory sessions (dozens of questions): Turn Summaries = Significant cost savings.

CONTEXT PURGE:

Fresh start WITHOUT creating new session (preserves history + tool results).

Click “Purge Context” → Confirm

All checkboxes uncheck. Context indicator: “0 turns active”

Agent has zero conversational memory. Every query = Brand new.

History still visible. Reactivate turns selectively.

QUERY REPLAY:

Turn 2 gave interesting results. Want different angle?

Three-dot menu (Turn 2) → “Replay Query”

Original question appears: *“Show customer demographics for those products.”*

Modify it: *“Show customer demographics AND purchasing patterns for those products.”*

Hit enter.

Agent executes with Turn 2 context (Turn 1 still active) + enhanced question.

Becomes Turn 5.

Alternative path explored. Original results preserved.

CONTEXT INDICATOR:

Top right - real-time status:

```
"4/5 turns active (2,340 tokens)"
```

Tells you: - 4 of 5 turns providing context - Current context size: 2,340 tokens - Exact token count sent to LLM

ADVANCED USE CASE:

Complex analysis: 20 turns

```
Turns 1-5: Data exploration (messy)
Turns 6-15: Deep analysis (focused)
Turns 16-20: Final synthesis (polished)
```

Goal: Clean summary without exploratory noise.

Strategy: - Deactivate Turns 1-5 - Keep Turns 6-20 active

Ask: *“Summarize our key findings.”*

Agent generates summary from: - Deep analysis (6-15) - Final synthesis (16-20)

Exploratory phase excluded. Context clean.

THE POWER:

Surgical control over agent memory.

Context = Tool (not limitation)

No other platform: This precision.

Uderia: Context management = First-class feature.

Real-world workflows: What agent remembers = As important as what it can do.

Memory precision + Cost control + Exploratory freedom.

All in your hands.

Screen Capture Plan:
- [ ] Session History panel showing 4 turns with checkboxes (all checked)
- [ ] Context indicator showing "4 turns active"
- [ ] Uncheck Turn 1 → Context indicator updates to "3 turns active"
- [ ] New query asked with Turn 1 deactivated
- [ ] Context mode toggle: Switch from "Full Context" to "Turn Summaries"
- [ ] Token count comparison (Full Context: 4,200 tokens vs Turn Summaries: 1,300 tokens)
- [ ] Context Purge button → Confirmation dialog
- [ ] All checkboxes unchecked after purge
- [ ] Context indicator showing "0 turns active"
- [ ] Turn 2 three-dot menu → "Replay Query"
- [ ] Modified query in input box
- [ ] New Turn 5 executes with enhanced question
- [ ] Complex scenario: 20 turns displayed, selectively activate Turns 6-20
- [ ] Summary query generates clean results

Supporting Materials:
- Context management best practices
- Token optimization guide
- Turn Summaries vs Full Context comparison
- Advanced exploration workflows
- Cost optimization through context control

Supporting Materials:
- Context management guide
- Advanced context strategies
- Use case examples

3.4 System Customization: Shape Agent Behavior
Duration: 3-4 minutes
Learning Objectives:
- Edit system prompts for custom behavior
- Test baseline LLM without tools
- Manage capability enable/disable

Key Topics to Cover:
- [ ] System Prompt Editor access
- [ ] Per-model instruction customization
- [ ] Save and reset capabilities
- [ ] Direct Model Chat for baseline testing
- [ ] Dynamic capability management (tools/prompts)

Narration Script:

Transparency = Seeing what agent does. Customization = Controlling HOW it behaves.

Uderia: Deep customization. No proprietary walls.

SYSTEM PROMPT EDITOR:

Setup → Advanced → System Prompts

Actual system instructions. Guide agent behavior.

Organized by provider:

```
- MASTER_SYSTEM_PROMPT (Universal)
- GOOGLE_MASTER_SYSTEM_PROMPT (Gemini optimizations)
- OLLAMA_MASTER_SYSTEM_PROMPT (Local model adaptations)
```

Click “Edit” on MASTER_SYSTEM_PROMPT:

```
You are an enterprise data intelligence agent with tools and knowledge.

Your role:
1. Decompose queries into strategic phases
2. Select optimal tools per phase
3. Execute with precision, self-correct on errors
4. Provide structured, actionable insights

Planning:
- Break complex requests into clear phases
- Identify required tools before execution
- Consider data dependencies
- Anticipate errors

[...more...]
```

This is the agent’s brain. Editable.

CUSTOMIZE REASONING:

Add verbosity:

```
When selecting tools, always explain:
- Why this tool is best
- What alternatives you considered
- What you expect returned
```

Save. Ask: *“What are the top revenue products?”*

Before:

```
Tool selected: base_readQuery
```

After:

```
Tool selected: base_readQuery
Rationale: Direct SQL execution optimal for revenue aggregation.
Alternatives: base_tableList (insufficient-structure only),
```

```
              qlty_dataProfile (unnecessary-not quality query)
Expected: Table with ProductName, Revenue (sorted desc)
```

You customized agent communication by editing system prompt.

PROVIDER-SPECIFIC PROMPTS:

Gemini ≠ Claude ≠ GPT-4. Optimizations differ.

GOOGLE_MASTER_SYSTEM_PROMPT:

```
Function calling: Structure arguments as clear JSON
Gemini strength: Parallel tool execution
Identify opportunities for simultaneous tool calls
```

Provider-specific tuning = Optimal performance per model.

SAVE AND RESET:

```
Save to Overrides: Stores in prompt_overrides/ (persists across updates)
Reset to Default: Revert to baseline
```

DIRECT MODEL CHAT:

Test raw LLM. No tools. No agent logic.

Setup → Advanced → Direct Model Chat

Ask: *"What are the top revenue products?"*

LLM Response:

```
"I don't have database access. Provide data or run query."
```

Baseline = No tooling.

Switch back to agent mode. Same question:

Agent Response:

```
Actual data from fitness_db via base_readQuery:
1. Treadmill Pro 3000 - $45,230
2. Elliptical Elite - $38,900
...
```

Comparison shows orchestration layer value: - Tool access - Context management - Execution planning

DYNAMIC CAPABILITY MANAGEMENT:

Setup → Capabilities → Tools

```
base_databaseList (Enabled)
base_tableList (Enabled)
base_tableDDL (Enabled)
base_readQuery (Enabled)
```

Experiment: Disable `base_readQuery`

Toggle OFF → Confirm

```
base_readQuery (Disabled)
```

Ask: *“Top 5 products by revenue?”*

Live Status Panel:

```
ERROR: base_readQuery unavailable
Replanning: Use base_tableList + TDA_LLMTask for schema insights
Query fails or produces workaround
```

Re-enable → Toggle ON

Same question → □ Succeeds instantly

USE CASES: - Test agent without specific tools - Restrict expensive tools (cost control) - Disable problematic tools (debugging) - Phase in new tools gradually

No restart. No code edits. Dynamic.

PROMPT MANAGEMENT:

Setup → Capabilities → Prompts

Disable `base_databaseBusinessDesc`

Ask: *“Business overview of fitness_db?”*

Agent falls back: `base_tableList` + `base_tableDDL` (manual investigation)

Re-enable → Efficient pre-built workflow returns

THE SHIFT:

Most AI platforms: Proprietary secrets. Take what they give.

Uderia: Full keys. - Edit system prompts - Test raw models - Enable/disable capabilities dynamically

Not just transparency. CONTROL.

Your agent. Your rules. Your customization.

Trust through visibility. Power through customization.

```
**Screen Capture Plan:**
- [ ] Setup → Advanced → System Prompts menu
- [ ] System Prompt Editor showing MASTER_SYSTEM_PROMPT
- [ ] Edit mode: Adding custom reasoning verbosity instructions
- [ ] Save changes
- [ ] Query execution showing enhanced tactical reasoning output
- [ ] Provider-specific prompts: GOOGLE_MASTER_SYSTEM_PROMPT example
- [ ] "Save to Overrides" and "Reset to Default" buttons
- [ ] Setup → Advanced → Direct Model Chat
- [ ] Direct chat query (no tools) → Generic LLM response
- [ ] Switch back to agent mode → Same query with actual data results
- [ ] Setup → Capabilities → Tools management
```

```
- [ ] Toggle base_readQuery to Disabled
- [ ] Query execution failing/workaround due to disabled tool
- [ ] Live Status Panel showing "base_readQuery unavailable" message
- [ ] Re-enable base_readQuery
- [ ] Query succeeds with proper tool
- [ ] Setup → Capabilities → Prompts management
- [ ] Disable base_databaseBusinessDesc prompt
- [ ] Query using fallback approach
- [ ] Re-enable prompt

**Supporting Materials:**
- System prompt customization guide
- Provider-specific optimization examples
- Direct Model Chat use cases
- Capability management best practices
- Tool filtering strategies
- Prompt override documentation

**Supporting Materials:**
- System prompt examples
- Customization best practices
- Developer mode guide

---

### 3.5 Execution Monitoring: Track Every Task
**Duration:** 2-3 minutes
**Learning Objectives:**
- Monitor cross-source workload tracking
- Review execution logs and tool history
- Learn task control operations

**Key Topics to Cover:**
- [ ] Executions View dashboard
- [ ] Real-time task list (running, completed, failed)
- [ ] Detailed execution logs
- [ ] Tool invocation history with arguments
- [ ] Task control (cancel, retry)

**Narration Script:**
```

Transparency doesn’t stop at current conversation.

Uderia: Complete view of ALL agent activity—past, present, running.

Welcome to **Executions View** = Mission Control.

Main Navigation → “Executions”

EXECUTIONS DASHBOARD:

Mission control for all agent tasks: - UI conversations - REST API calls - Airflow DAGs - Flowise workflows

Three sections:

1. ACTIVE TASKS (Real-Time)

Currently running:

```
Task: task_7f3a2b  Status: RUNNING  Progress: 65%
Query: "Analyze quarterly revenue trends"
Session: Revenue Analysis Q4  Started: 2m ago
Phase: Data aggregation (Phase 2 of 3)
Tools: base_tableList, base_readQuery
[View Live Status] [Cancel]
```

Click “View Live Status” → Modal opens

Watch execution real-time (even if submitted via API, not UI):

```
- Strategic plan: Phase 1    Phase 2    Phase 3
- Current: base_readQuery (complex aggregation SQL)
- Tokens: 3,200 in / 1,800 out (live)
- Cost: $0.023 (estimated)
```

2. COMPLETED TASKS (Historical Log)

Execution history:

```
  Task: task_6a8d1c  COMPLETED  Duration: 12s
Query: "Top 5 products by revenue?"
Session: Morning Batch  Finished: 5m ago
Tools: base_readQuery, TDA_FinalReport
Tokens: 1,250 in / 340 out  Cost: $0.008
[View Execution Log] [Replay Query]
```

Click “View Execution Log” → Forensic detail:

```
Task: task_6a8d1c
Query: "Top 5 products by revenue?"
Session: Morning Batch (abc-123)
Profile: @PROD (Claude Sonnet 3.5 + Teradata MCP)
Started: 09:15:23 UTC
Completed: 09:15:35 UTC
Duration: 12.4s

--- STRATEGIC PLAN ---
Phase 1: Revenue aggregation query
Phase 2: Final report formatting
```

```
--- EXECUTION TRACE ---

[09:15:23] Planning
  LLM: Strategic Planner (Claude Sonnet 3.5)
  Tokens: 1,100 in / 85 out
  Cost: $0.0035
  Result: 2-phase plan

[09:15:25] Phase 1
  Tool: base_readQuery
  SQL: SELECT p.ProductName, SUM(sd.Quantity * sd.UnitPrice) as Revenue
       FROM Products p JOIN SaleDetails sd ...
  Time: 3.2s
  Result: 5 rows
  Preview: [{"ProductName": "Treadmill Pro 3000", "Revenue": 45230}, ...]

[09:15:29] Phase 2
  Tool: TDA_FinalReport
  Format: table_with_summary
  LLM: Report Formatter
  Tokens: 150 in / 255 out
  Cost: $0.0045
  Result: Formatted HTML

[09:15:35]  Completed

TOTAL: 1,250 in / 340 out | $0.008
```

Forensic detail: - Exact timing - Every tool + arguments - Every LLM call + tokens - SQL executed - Data returned - Cost breakdown

Something wrong? Know exactly where and why.

3. FAILED TASKS (Error Investigation)

```
   Task: task_2c5e9f  FAILED  Duration: 8s
  Query: "Compare sales across regions"
  Session: Regional Analysis  Failed: 1h ago
  Error: Table 'fitness_db.Regions' not found
  [View Error Log] [Retry with Corrections]
```

Click “View Error Log”:

```
[10:42:15] Phase 1
  Tool: base_readQuery
  SQL: SELECT * FROM fitness_db.Regions ...
    ERROR: Table does not exist
    Recovery: Replan using Customers.State
    Recovery FAILED: Insufficient data
```

```
[10:42:23] Task Failed
  Reason: Unable to complete with available tables
  Suggestion: Create Regions table OR use City/State from Customers
```

Agent tried self-correction. Recovery failed. You know exactly what to fix.

Click “Retry with Corrections” → Modify query: *“Compare sales by state using customer location.”*

TASK CANCELLATION:

Runaway query?

Active Tasks → Find task → “Cancel Task” → Confirm

```
   Task: task_9b3f1a  CANCELLED  Duration: 45s
  Query: "Full DB quality scan"
  Session: System Audit  Cancelled: Now
  Reason: Manual cancellation
  Progress: 30% (Phase 1 of 4 complete)
```

Graceful shutdown. Resources freed. Partial results saved.

CROSS-SOURCE VISIBILITY:

ALL sources in one place:

```
- UI conversations
- REST API calls (Postman, cURL, apps)
- Airflow DAGs (batch jobs)
- Flowise workflows (chatbots)
```

Unified execution log.

FILTER BY: - Session ID - Profile
- Status (running/completed/failed/cancelled) - Date range - User

OPERATIONAL INTELLIGENCE:

Not just debugging. Operational insights:

```
Sort by cost     → Most expensive queries?
Filter by failed → Which sessions failing most?
Filter by pattern→ Avg execution time for revenue queries?
Aggregate usage  → Which MCP tools used most?
```

Agent activity → Operational data.

THE POWER:

Every task. Every tool. Every token. Every error.

Complete transparency across entire AI operations.

Production-grade monitoring. Trust through visibility. Control through insight.

```
**Screen Capture Plan:**
- [ ] Executions View navigation (main menu)
- [ ] Dashboard with three sections: Active, Completed, Failed
- [ ] Active task card showing progress bar and current phase
- [ ] Click "View Live Status" → Modal with real-time execution
- [ ] Strategic plan showing Phase 1  , Phase 2 Running, Phase 3 Pending
- [ ] Tool execution details (base_readQuery in progress)
- [ ] Token count and cost tracking (live updates)
- [ ] Completed tasks list with multiple entries
- [ ] Click "View Execution Log" on completed task
- [ ] Detailed execution trace with timestamps
- [ ] Tool invocations with full arguments (SQL queries visible)
- [ ] Token and cost breakdown per phase
- [ ] Failed task card with error message
- [ ] Click "View Error Log" → Detailed failure analysis
- [ ] Recovery attempt details
- [ ] "Retry with Corrections" button
- [ ] Active task cancellation: Click "Cancel Task" → Confirmation
- [ ] Task status changes to CANCELLED
- [ ] Filter options: Session, Profile, Status, Date range
- [ ] Cross-source task list: UI, API, Airflow labels
- [ ] Sort by cost (most expensive queries at top)

**Supporting Materials:**
- Execution monitoring guide
- Task lifecycle documentation
- Error investigation workflows
- Operational analytics examples
- Cross-source tracking reference
- Performance optimization guide

**Supporting Materials:**
- Execution monitoring guide
- Task lifecycle documentation
- Error debugging examples

---

## Module 4: EFFICIENT - The Fusion Optimizer
**Goal:** Showcase the revolutionary optimization engine and self-learning capabilities

### 4.1 Strategic & Tactical Planning Architecture
**Duration:** 4-5 minutes
**Learning Objectives:**
- Understand multi-layered planning process
- See strategic meta-plans in action
```

```
- Grasp recursive delegation concept

**Key Topics to Cover:**
- [ ] Strategic Planner (high-level meta-plan)
- [ ] Tactical Execution (best next action)
- [ ] Recursive delegation for complex problems
- [ ] Phase-by-phase breakdown visualization
- [ ] Plan vs execution distinction

**Narration Script:**
```

Welcome to Uderia's heart—the **Fusion Optimizer**.

LLM wrapper → Enterprise AI orchestration engine. This is the transform.

MOST AI TOOLS:

```
User asks → Model responds → Repeat

No planning. No optimization. No intelligence in execution.
```

FUSION OPTIMIZER:

```
Strategic planning + Tactical execution + Continuous optimization

Multi-layered architecture. Enterprise-grade intelligence.
```

SCENARIO: Q4 Board Meeting Prep

CEO wants: Why revenue missed + What actions to take.

Traditional setup requires:

```
1. BI Analyst     → Pull sales data
2. Data Engineer  → Correlate service data
3. Business Analyst → Synthesize insights
4. Executive      → Frame recommendations

Multiple specialists. Days of coordination.
```

Uderia:

```
One intelligent workflow. All perspectives orchestrated.
```

Profile: @ANALYST (business knowledge + reporting patterns + visualization tools)

Ask: *"Analyze fitness equipment business: identify declining product categories, investigate service patterns, recommend improvements."*

Watch Live Status Panel.

STRATEGIC PLANNING LAYER:

Agent doesn't just execute. First: Generates meta-plan (strategic blueprint):

```
STRATEGIC META-PLAN:
```

```
Phase 1: Identify declining product categories
  Requires: Sales, SaleDetails tables
  Output: Categories with negative growth
  Dependencies: None (execute immediately)

Phase 2: Analyze service ticket patterns
  Requires: ServiceTickets + Phase 1 results
  Output: Ticket volume, resolution times, issues
  Dependencies: Phase 1

Phase 3: Correlate sales decline + service issues
  Requires: Phase 1 + Phase 2 results
  Output: Root cause analysis
  Dependencies: Phases 1, 2

Phase 4: Generate actionable recommendations
  Requires: All analysis
  Output: Prioritized strategies
  Dependencies: All phases
```

Strategic thinking. Entire workflow mapped BEFORE executing single tool.

TACTICAL EXECUTION LAYER:

Phase 1 execution begins.

Strategic planner → Tactical executor: “Complete Phase 1”

Tactical layer asks: “Best next action?”

Tactical Decision:

```
Goal: Identify declining categories
Action: SQL query comparing Q4 vs Q3 sales by ProductType
Tool: base_readQuery
Rationale: Single query captures current + previous quarter,
           calculates growth %, identifies decline efficiently
```

SQL Generated:

```sql
SELECT p.ProductType,
       SUM(CASE WHEN QUARTER(s.SaleDate) = CURRENT_QUARTER
           THEN sd.Quantity * sd.UnitPrice ELSE 0 END) as CurrentQ,
       SUM(CASE WHEN QUARTER(s.SaleDate) = CURRENT_QUARTER-1
           THEN sd.Quantity * sd.UnitPrice ELSE 0 END) as PrevQ,
       ((CurrentQ - PrevQ) / PrevQ * 100) as GrowthPct
FROM Products p
JOIN SaleDetails sd ON p.ProductID = sd.ProductID
JOIN Sales s ON sd.SaleID = s.SaleID
WHERE s.SaleDate >= '2024-07-01'
GROUP BY p.ProductType
ORDER BY GrowthPct ASC;
```

Tactical precision. Optimal tool + approach for this step.

Results:


```
[
  {"ProductType": "Cardio", "CurrentQ": 125000, "PrevQ": 180000, "GrowthPct": -30.5},
  {"ProductType": "Accessories", "CurrentQ": 45000, "PrevQ": 52000, "GrowthPct": -13.5},
  {"ProductType": "Strength", "CurrentQ": 200000, "PrevQ": 190000, "GrowthPct": 5.3}
]
```


 Phase 1 complete. Cardio: -30.5% | Accessories: -13.5%

RECURSIVE DELEGATION:

Phase 2: “Analyze service ticket patterns for declining products”

Complex phase requires:

```
1. Get tickets for Cardio + Accessories
2. Calculate avg resolution times
3. Categorize issue types
4. Identify patterns
```

Strategic planner doesn’t handle monolithically.

Delegates to NEW subordinate planner instance.

Recursive sub-plan appears:

```
SUB-PLAN for Phase 2:
  2.1: Retrieve product IDs for Cardio and Accessories categories
  2.2: Query ServiceTickets for those product IDs
  2.3: Calculate resolution time statistics
  2.4: Extract and categorize issue descriptions
```

This sub-planner operates independently, executes its sub-phases, and returns consolidated results to the parent planner.

The parent doesn’t micromanage. It delegated a complex sub-task to a specialized instance, which broke it down further and solved it.

This is recursive intelligence.

Sub-Plan 2.1 executes:

```
Tool: base_readQuery
SQL: SELECT ProductID FROM fitness_db.Products WHERE ProductType IN ('Cardio', 'Accessories')
Result: 47 product IDs identified
```

Sub-Plan 2.2 executes:

```
Tool: base_readQuery
SQL: SELECT ProductID, COUNT(*) as TicketCount,
            AVG(CAST(ResolutionDate - TicketDate AS INT)) as AvgResolutionDays,
            IssueDescription
     FROM fitness_db.ServiceTickets
     WHERE ProductID IN (list of 47 IDs)
```

```
    GROUP BY ProductID
Result: Service patterns retrieved
```

Sub-Plan 2.3 executes:

```
Tool: TDA_LLMTask
Task: "Analyze resolution time distribution and identify outliers"
Result: Cardio products average 8.3 days to resolve vs 2.7 for Accessories
```

Sub-Plan 2.4 executes:

```
Tool: TDA_LLMTask
Task: "Categorize issue descriptions into common themes"
Result: Top issues for Cardio: Motor failures (45%), Belt wear (30%), Electronic issues (25%)
```

Sub-plan complete. Results return to parent Phase 2. Phase 2 marks complete.

HIERARCHICAL PROBLEM SOLVING:

Phase 3 begins: “Correlate sales decline with service issues.”

This phase uses results from BOTH Phase 1 and Phase 2. The strategic planner has those results in context.

Tactical execution:

```
Tool: TDA_LLMTask
Task: "Analyze correlation between 30% Cardio sales decline and 8.3-day average
       service resolution time with 45% motor failure rate. Determine causation likelihood."
Result: Strong correlation. Customer reviews and repeat purchase rates show service
        experience directly impacts sales. Motor failures driving negative sentiment.
```

Phase 3 complete.

Phase 4 begins: “Generate actionable recommendations.”

```
Tool: TDA_FinalReport
Arguments: All prior phase results
Task: Synthesize insights into prioritized recommendations
Result:
  1. URGENT: Address motor quality issues in Cardio line (root cause)
  2. HIGH: Reduce service resolution time target to <5 days (customer experience)
  3. MEDIUM: Implement proactive outreach for products >6 months old (prevention)
```

Final answer delivered to the user.

WHAT HAPPENED:

Complex question decomposed into:

```
 4-phase strategic meta-plan
 Tactical tool selections per phase
 Recursive sub-plan for Phase 2
 Data gathering → Analysis → Correlation → Synthesis
 Actionable recommendations
```

All autonomous. All visible. All optimized.

THE DISTINCTION:

```
Chatbot: User asks → Model responds
Uderia:  Strategic planning → Tactical execution → Recursive delegation
```

Enterprise AI orchestration engine.

FUSION OPTIMIZER =

```
Strategic planning + Tactical execution
Recursive delegation + Hierarchical problem-solving
```

WHY IT MATTERS:

Uderia: Enterprise complexity □ Other tools: Simple Q&A only

Intelligence in architecture. Not just model.

```
**Screen Capture Plan:**
- [ ] Complex multi-part query input
- [ ] Strategic Meta-Plan appears with 4 phases and dependencies mapped
- [ ] Phase 1 tactical execution: base_readQuery with sales comparison SQL
- [ ] Phase 1 results: Declining product categories identified
- [ ] Phase 2 begins: "Analyze service patterns"
- [ ] RECURSIVE delegation indicator appears
- [ ] Sub-plan appears: 2.1, 2.2, 2.3, 2.4 sub-phases
- [ ] Sub-plan execution: Multiple base_readQuery and TDA_LLMTask calls
- [ ] Sub-plan results consolidated and returned to parent
- [ ] Phase 2 complete checkmark
- [ ] Phase 3 execution: Correlation analysis using Phase 1 + Phase 2 data
- [ ] Phase 4 execution: TDA_FinalReport with synthesized recommendations
- [ ] Final formatted report in Conversation Panel
- [ ] Execution summary: 4 phases, 8 tool calls, 1 recursive delegation

**Supporting Materials:**
- Fusion Optimizer architecture diagram
- Strategic vs Tactical planning comparison
- Recursive delegation examples
- Complex query patterns
- Phase dependency visualization

**Supporting Materials:**
- Fusion Optimizer architecture diagram
- Complex query examples
- Planning algorithm documentation

---
```

```
### 4.2 RAG Self-Learning System: Continuous Improvement
**Duration:** 4-5 minutes
**Learning Objectives:**
- Understand automatic case capture
- Learn efficiency analysis and champion selection
- See few-shot learning injection in action

**Key Topics to Cover:**
- [ ] Automatic case archiving after successful sessions
- [ ] Efficiency scoring (token reduction, fast paths)
- [ ] Champion strategy selection
- [ ] Few-shot learning injection at planning time
- [ ] Asynchronous processing (zero latency)

**Narration Script:**
```

Fusion Optimizer: Powerful.

What makes it revolutionary: **Learns from every success.**

Welcome to self-improving RAG system.

MOST AI AGENTS:

```
Execute in isolation
Don't remember what worked
Don't learn from success
Reinvent wheel every time
```

UDERIA RAG:

```
Every successful interaction → Organizational knowledge
Agent gets smarter with every query
```

AUTOMATIC CASE CAPTURE:

Intelligence → RAG Collections

“Planner Repository” = Knowledge base of successful execution patterns

Every completed session: - Analyzed automatically - Archived as reusable pattern

Click case: “Revenue Analysis - Top Products by Category”

```
CASE: Revenue Analysis - Top Products by Category

Query: "Top 5 products by revenue in each category?"

Strategic Plan:
  Phase 1: Revenue aggregation with category grouping
  Phase 2: Format results

Tool Trace:
  1. base_readQuery (1.8s)
```

```
     SQL: SELECT ProductType, ProductName, Brand,
                 SUM(Quantity * UnitPrice) as Revenue,
                 RANK() OVER (PARTITION BY ProductType
                              ORDER BY Revenue DESC) as Rank
          FROM Products p
          JOIN SaleDetails sd ON p.ProductID = sd.ProductID
          WHERE Rank <= 5
          GROUP BY ProductType, ProductName, Brand
     Result: 15 rows (5 per category)

  2. TDA_FinalReport
     Format: Grouped table
     Result: Clean output

Tokens:
  Planning: 1,200 in / 95 out
  Execution: 850 in / 120 out
  Total: 2,265

Cost: $0.0112

Efficiency Score: 94/100
    Optimal tool selection
    Minimal tokens
    Single-pass
    No errors/retries

Champion Case: YES  (lowest tokens for pattern)
```

Entire workflow → Reusable pattern.

EFFICIENCY ANALYSIS:

Score: 94/100. RAG analyzes every case:

```
Token Efficiency:    Minimal tokens?
Execution Efficiency: Single-pass?
Tool Selection:      Optimal choices?
Strategic Planning:  Direct path?
```

Cases ranked. Most efficient = **Champion Case** □

List shows champion badges.

CHAMPION STRATEGY SELECTION:

Let's say we have three cases for "top revenue queries":

1. Revenue by Product (Champion □) - 2,265 tokens, 1.8s, score 94
2. Revenue by Category - 3,100 tokens, 3.2s, score 78
3. Revenue with Customer Analysis - 5,800 tokens, 7.5s, score 65

Case 1 is the champion—most efficient for straightforward revenue queries.

When a new similar query comes in, the RAG system will inject Case 1 as a few-shot example to guide the strategic planner.

FEW-SHOT LEARNING INJECTION:

New similar query: *"Show me top revenue products by brand this month."*

Live Status Panel → Strategic planning shows:

RAG-RETRIEVED CONTEXT:

```
Similar Past Cases:

1. Revenue Analysis - Top Products by Category (similarity: 0.89)
   Strategy: Single aggregation + grouping
   Tools: base_readQuery → TDA_FinalReport
   Tokens: 2,265 | Cost: $0.0112 | Duration: 1.8s
   Key: RANK() window function for top-N per group

2. Monthly Revenue Trends (similarity: 0.76)
   Strategy: Time-based filtering + SUM
   Tools: base_readQuery → TDA_FinalReport
   Tokens: 1,980 | Cost: $0.0095 | Duration: 1.5s
   Key: EXTRACT(MONTH) for current month filter
```

Strategic planner learns from champions:

```
Case 1 → Use RANK() for top-N per group
Case 2 → Use EXTRACT(MONTH) for monthly filter
Combine both patterns
```

Strategic plan generated:

```
Phase 1: Execute revenue query by brand for current month using optimized SQL pattern
  Tool: base_readQuery
  SQL: SELECT p.Brand, p.ProductName,
              SUM(sd.Quantity * sd.UnitPrice) as Revenue,
              RANK() OVER (PARTITION BY p.Brand ORDER BY Revenue DESC) as Rank
       FROM fitness_db.Products p
       JOIN fitness_db.SaleDetails sd ON p.ProductID = sd.ProductID
       JOIN fitness_db.Sales s ON sd.SaleID = s.SaleID
       WHERE EXTRACT(MONTH FROM s.SaleDate) = EXTRACT(MONTH FROM CURRENT_DATE)
         AND Rank <= 5
       GROUP BY p.Brand, p.ProductName

Phase 2: Format results
  Tool: TDA_FinalReport
```

Look at that—the agent learned the RANK() pattern from past success. It didn't waste time exploring alternative approaches. It went straight to the proven champion strategy.

Execution completes.

Token usage: 2,180 total (85 fewer than the first time we solved this pattern) Cost: $0.0106 (saved $0.0006) Duration: 1.6s (0.2s faster)

CONTINUOUS IMPROVEMENT:

New execution analyzed:

```
Succeeded?              YES
More efficient?         YES (fewer tokens, faster)
Become new champion?    YES
```

RAG auto-promotes to champion.

Next similar query → Benefits from even-more-optimized approach.

THE EVOLUTION:

```
Agent learning: Continuous
Getting faster: Automatic
Getting cheaper: Automatic
```

ASYNCHRONOUS PROCESSING:

Does RAG slow responses?

No. Watch timeline:

```
[00:00.000] User submits query
[00:00.050] RAG retrieval starts (parallel)
[00:00.100] Strategic planner invoked
[00:00.350] RAG context injected
[00:00.400] Strategic plan finalized
[00:00.450] Tactical execution begins
[00:02.100] Results → User
------- USER HAS ANSWER -------
[00:02.150] Async: Case archiving (background)
[00:03.500] Async: Efficiency analysis (background)
[00:03.600] Async: Case stored (background)
```

RAG retrieval: During planning (<300ms) Case archiving: After user gets answer

Zero user-facing latency.

PER-USER COST SAVINGS ATTRIBUTION:

Go to Intelligence → Efficiency Metrics.

You'll see a dashboard:

```
YOUR RAG EFFICIENCY GAINS:

Total Queries: 247
RAG-Guided Queries: 183 (74%)

Token Savings:
  Without RAG (projected): 892,000 tokens
```

```
  With RAG (actual): 645,000 tokens
  Savings: 247,000 tokens (28% reduction)

Cost Savings:
  Projected cost: $52.30
  Actual cost: $37.60
  Savings: $14.70 (28% reduction)

Time Savings:
  Average execution time without RAG: 4.2s
  Average execution time with RAG: 2.8s
  Per-query savings: 1.4s (33% faster)
```

The RAG system isn't just making the agent smarter—it's making it dramatically more cost-effective. And these savings compound as your repository grows.

KNOWLEDGE TRANSFER & PERSONA-BOUND LEARNING:

Here's the organizational benefit—and it gets even more powerful with profiles.

When you use the @DBA profile to optimize a complex join query, that champion case gets stored in the DBA persona's planner repository. The next time ANYONE switches to @DBA, they inherit that optimized pattern. Your database expertise becomes the DBA persona's expertise.

When your colleague uses @ANALYST to create an executive dashboard, that visualization pattern gets captured. Now the @ANALYST persona has learned a new reporting style. Every user of that profile benefits.

When the quality team uses @QUALITY to validate data, those validation workflows become part of the Quality persona's playbook.

This is collective intelligence at work—not just across users, but across specialized personas. Each profile evolves independently: - @DBA gets better at SQL optimization - @ANALYST gets better at business insights
- @QUALITY gets better at data validation - @LOCAL maintains same patterns with sovereign execution

You're not just improving one agent. You're building a library of specialized expert personas, each learning from every user's successes in their domain.

Switch profiles. Switch to the persona that learned from your team's best work. Orchestrate expertise that compounds over time.

CROSS-TIER KNOWLEDGE TRANSFER: Train on Premium, Execute on Budget

Here's the ultimate cost optimization strategy: use expensive intelligence to teach, then use cheap execution to repeat.

Let's demonstrate with a complex analytical challenge that would normally require a premium model.

Phase 1: Learning with @OPUS (Claude Opus - Premium)

Switch to @OPUS profile. This is the most expensive, most capable model—$15 per 1M input tokens. But we're going to use it strategically.

Ask a genuinely complex question:

“Analyze the relationship between customer demographics (age, location), product purchase patterns, service ticket frequency, and revenue contribution to identify high-value customer segments and predict churn risk.”

This requires sophisticated reasoning: - Multi-table joins (Customers, Sales, SaleDetails, Products, ServiceTickets) - Statistical analysis (correlation, clustering) - Pattern recognition (churn indicators) - Segmentation logic (defining high-value cohorts)

Watch @OPUS work. The Live Status Panel shows:

```
Profile: @OPUS (Claude Opus)
Cost Tier: Premium ($15/$75 per 1M tokens)

Strategic Plan (6 phases):
  Phase 1: Aggregate customer purchase history
  Phase 2: Join service ticket patterns
  Phase 3: Calculate customer lifetime value
  Phase 4: Perform demographic clustering
  Phase 5: Identify churn risk indicators
  Phase 6: Generate segmentation report

Tool Executions:
  - base_readQuery (3 complex multi-table joins)
  - TDA_LLMTask (statistical analysis)
  - TDA_LLMTask (clustering algorithm)
  - TDA_FinalReport (executive summary)

Execution:
  Tokens: 12,400 (input) + 2,300 (output) = 14,700 total
  Cost: $0.3585
  Duration: 8.7s
  Result: HIGH QUALITY - Sophisticated segmentation with 5 distinct customer cohorts
```

Expensive? Yes. But brilliant. Opus identified patterns a simpler model would miss.

Champion Case Captured:

The RAG system analyzes this execution:

```
Case: "Customer Segmentation with Churn Prediction"
Profile: @OPUS
Efficiency Score: 91/100
  - Complex reasoning:   (required premium model)
  - Optimal tool selection:
  - Minimal retries:
  - High-quality output:

Champion Case: YES
Pattern Captured:
  - Strategic approach: Multi-phase with clustering
```

```
  - Tool sequence: Aggregation → Join → Calculate → Analyze → Report
  - SQL patterns: Complex CTEs with window functions
  - Analysis techniques: Demographic clustering, churn indicators

Stored in: DBA_Champion_Cases + ANALYST_Champion_Cases
(This pattern is valuable for both database optimization and business analysis)
```

We just paid $0.36 to solve this problem brilliantly. But that investment is about to pay massive dividends.

Phase 2: Execution with @COST (Gemini Flash - Budget)

Now switch to @COST profile. This is the cheapest model available—$0.075 per 1M input tokens. That's 200x cheaper than Opus for input, 125x cheaper for output.

Ask a similar question (not identical, but same pattern):

"Identify high-value customer segments for our Strength equipment line and predict which segments are at churn risk."

Watch the magic happen. The Live Status Panel shows:

```
Profile: @COST (Gemini 1.5 Flash)
Cost Tier: Budget ($0.075/$0.30 per 1M tokens)

  RAG-RETRIEVED CONTEXT:

Champion Case Found: "Customer Segmentation with Churn Prediction"
  - Similarity: 0.94 (excellent match)
  - Source Profile: @OPUS
  - Original Cost: $0.3585
  - Strategy: Multi-phase clustering with churn analysis
  - Tools: base_readQuery (aggregations) → TDA_LLMTask (clustering) → TDA_FinalReport
  - SQL Pattern: CTEs with window functions for lifetime value calculation
  - Analysis Pattern: Demographic clustering → Service pattern correlation → Churn scoring

Applying learned strategy...

Strategic Plan (6 phases - INHERITED FROM OPUS):
  Phase 1: Aggregate customer purchase history (Strength products filter)
  Phase 2: Join service ticket patterns
  Phase 3: Calculate customer lifetime value
  Phase 4: Perform demographic clustering
  Phase 5: Identify churn risk indicators
  Phase 6: Generate segmentation report

Tool Executions (SAME SEQUENCE AS OPUS):
  - base_readQuery (using Opus-optimized SQL patterns)
  - TDA_LLMTask (following Opus analysis structure)
  - TDA_LLMTask (applying Opus clustering approach)
  - TDA_FinalReport (matching Opus report format)
```

```
Execution:
  Tokens: 8,200 (input) + 1,850 (output) = 10,050 total
  Cost: $0.0123
  Duration: 5.1s
  Result: HIGH QUALITY - Sophisticated segmentation with 4 distinct customer cohorts
```

Look at what just happened:

Cost Comparison: - @OPUS: $0.3585 (to learn the pattern) - @COST: $0.0123 (to execute the learned pattern) - **Savings: $0.3462 per query (96.5% cost reduction)**

Quality Comparison: - @OPUS: 5 customer cohorts, sophisticated clustering - @COST: 4 customer cohorts, nearly identical insights - **Quality preservation: ~95% of Opus quality at 3.5% of Opus cost**

The ROI Calculation:

If you run this type of analysis monthly: - Without RAG: 12 queries × $0.3585 = $4.30/year (all @OPUS) - With RAG Transfer: 1 query × $0.3585 + 11 queries × $0.0123 = $0.4938/year - **Annual Savings: $3.80 (88.5% cost reduction)**

If you run it weekly: - Without RAG: 52 queries × $0.3585 = $18.64/year - With RAG Transfer: 1 query × $0.3585 + 51 queries × $0.0123 = $0.9858/year
- **Annual Savings: $17.65 (94.7% cost reduction)**

The Strategy:

1. Use premium models (@OPUS, @SONNET) to solve complex problems ONCE
2. RAG captures the champion strategy automatically
3. Use budget models (@COST, @GEMINI_FLASH) for subsequent similar queries
4. Budget models inherit premium intelligence via few-shot learning
5. Pay premium prices for intelligence, budget prices for execution

Cross-Profile Learning at Scale:

This works across ANY profile combination:

- **@OPUS → @ANALYST:** Learn complex business analysis, execute cheaply for routine reports
- **@SONNET → @DBA:** Learn SQL optimization, execute with local Ollama for free
- **@OPUS → @QUALITY:** Learn sophisticated anomaly detection, execute routinely at low cost
- **Premium Cloud → @LOCAL:** Learn in cloud, execute sovereignly on-premises

The expensive model teaches. The cheap model learns. Your organization gets premium intelligence at budget prices.

Organizational Knowledge Compounding:

As your team uses premium profiles occasionally for hard problems, the RAG repositories fill with high-quality champion cases. Over time, your budget profiles become incredibly capable—not because the underlying LLM improved, but because they're standing on the shoulders of premium model intelligence.

This is cross-tier knowledge transfer. This is how you optimize costs without sacrificing quality.

Train with genius. Execute with efficiency. Compound organizational intelligence.

THE TRANSFORMATION:

```
Individual success      → Organizational knowledge
One-time execution      → Reusable patterns
Static AI               → Self-improving intelligence

Train with genius       → Execute with efficiency
Premium model teaches   → Budget model learns
Expensive intelligence  → Cheap execution
```

RAG-POWERED FUSION OPTIMIZER:

Agent that gets smarter every time you use it.

Continuous improvement. Organizational learning. Cross-tier knowledge transfer.

This is intelligence that compounds.

```
**Screen Capture Plan:**
- [ ] Intelligence → RAG Collections navigation
- [ ] Planner Repository view with list of cases
- [ ] Case detail: "Revenue Analysis - Top Products by Category"
- [ ] Strategic plan display
- [ ] Tool execution trace with SQL queries
- [ ] Token usage breakdown (planning + execution)
- [ ] Efficiency Score: 94/100 with breakdown
- [ ] Champion badge ( ) indicator
- [ ] List view showing three cases with champion ranking
- [ ] New query: "Top revenue products by brand this month"
- [ ] Live Status Panel: "RAG-RETRIEVED CONTEXT" section appears
- [ ] Similar cases displayed with similarity scores
- [ ] Strategic plan using learned patterns (RANK + EXTRACT)
- [ ] Execution completes with improved metrics
- [ ] New case promoted to champion status
- [ ] Timeline visualization: RAG retrieval parallel to planning
- [ ] Intelligence → Efficiency Metrics dashboard
- [ ] Token savings: 247,000 saved (28% reduction)
- [ ] Cost savings: $14.70 saved
- [ ] Time savings: 1.4s per query faster
- [ ] **CROSS-TIER DEMO:** Profile switch to @OPUS
- [ ] Complex segmentation query execution with @OPUS
- [ ] Token usage: 14,700 tokens, Cost: $0.3585
- [ ] Champion case capture with "Source Profile: @OPUS" tag
- [ ] Profile switch to @COST (Gemini Flash)
- [ ] Similar segmentation query
- [ ] Live Status: RAG retrieval shows @OPUS champion case
```

```
- [ ] "Applying learned strategy from @OPUS..."
- [ ] Execution with @COST: 10,050 tokens, Cost: $0.0123
- [ ] Side-by-side comparison: $0.3585 (Opus) vs $0.0123 (Flash)
- [ ] Savings calculation: 96.5% cost reduction
- [ ] ROI calculator: Annual savings projection
- [ ] Cross-profile learning matrix showing all combinations

**Supporting Materials:**
- RAG system architecture documentation
- Efficiency scoring methodology
- Champion case selection algorithm
- Few-shot learning examples
- Asynchronous processing flow diagram
- Cost savings calculator
- Cross-tier learning transfer diagram
- Premium vs Budget model pricing comparison
- ROI calculation methodology

---

### 4.3 Proactive Optimization: Fast Paths & Hydration
**Duration:** 3-4 minutes
**Learning Objectives:**
- Learn plan hydration mechanism
- Understand tactical fast path
- See specialized orchestrators in action

**Key Topics to Cover:**
- [ ] Plan hydration (reusing previous turn data)
- [ ] Tactical fast path (skip LLM for simple operations)
- [ ] Specialized orchestrators (date range, comparative analysis)
- [ ] Context distillation for large datasets
- [ ] Optimization impact on speed and cost

**Narration Script:**
```

Fusion Optimizer: Plans + Learns + **Hunts for optimization opportunities**

Proactive optimization. Before execution. During execution.

Make Uderia fast. Make Uderia cost-effective.

TECHNIQUE 1: PLAN HYDRATION

Query: *“Top 5 products by revenue this month?”*

Execution:

```
Strategic Plan: Single phase
Tool: base_readQuery
Result: 5 products + revenue
```

```
Tokens: 1,800
Duration: 2.1s
```

Results displayed.

Immediate follow-up: *“What about the top 10?”*

Live Status Panel → Watch carefully:

```
  PLAN HYDRATION DETECTED

New: "Top 10?"
Analysis: Expanded version of previous query

Previous results cached:
  Query: Top 5 products by revenue
  Tool: base_readQuery
  Results: 5 products
  Age: 3s (fresh)

Strategy: HYDRATE with cached data
    Skip Phase 1 planning (reuse strategy)
    Skip redundant call (modify LIMIT 5 → 10)
    Execute optimized query

Savings: ~900 tokens, ~$0.005, ~1.2s
```

Agent recognizes iterative refinement. Reuses context vs starting over.

Result:

```
Tokens: 950 (850 fewer)
Duration: 0.9s (1.2s faster)
Cost: $0.0045 ($0.005 saved)
```

Common triggers: - “What about X instead?” (parameter change) - “Show me more” (pagination) - “Add Y” (incremental) - “Break down by Z” (granularity)

TECHNIQUE 2: TACTICAL FAST PATH

Query: *“What’s the current date?”*

Live Status Panel:

No strategic planning. No LLM call.

```
  TACTICAL FAST PATH

Query: "Current date?"
Complexity: TRIVIAL (single deterministic tool)

Strategy: BYPASS strategic planning
    Skip planner LLM
    Direct tool invocation
```

```
  Tool: TDA_CurrentDate
  Args: {}

Savings: ~1,500 tokens, ~$0.008, ~0.8s
```

Instant recognition: Trivial query. Single tool. No args. No planning.

Direct execution:

```
Tool: TDA_CurrentDate
Result: "2025-12-07"
Time: 0.2s
```

No strategic plan. No tactical LLM. Pure efficiency.

Triggers: - Single deterministic tool - Arguments inferrable - No multi-step logic

Eliminate ~1,500 tokens per fast-path query. Massive savings for conversational UIs.

TECHNIQUE 3: SPECIALIZED ORCHESTRATORS

Query: *"Show daily sales for past week."*

The agent detects a pattern:

```
  OPTIMIZATION: DATE RANGE ORCHESTRATOR

Query: "daily sales for the past week"
Pattern detected: Time-series iteration over date range

Standard approach:
  - Plan each day as separate phase (7 phases)
  - Execute 7 individual queries
  - Tokens: ~12,000 | Duration: ~15s

Optimized approach:
  - Specialized Date Range Orchestrator
  - Single multi-day aggregation query
  - Post-processing for daily breakdown
  - Tokens: ~2,800 | Duration: ~3.5s

Applying optimization...
```

The orchestrator generates:

```sql
SELECT CAST(SaleDate AS DATE) as Day,
       SUM(TotalAmount) as DailySales
FROM fitness_db.Sales
WHERE SaleDate >= CURRENT_DATE - 7
GROUP BY CAST(SaleDate AS DATE)
ORDER BY Day
```

One query. Seven days. Clean results.

Tokens: 2,800 (9,200 saved vs naive approach) Duration: 3.2s (11.8s saved)

Other specialized orchestrators:

Comparative Analysis Orchestrator: Query: “Compare Claude vs GPT-4 on this question: [complex query]”

Detects: Multi-model comparison pattern Optimization: Executes query on both models in parallel, aggregates results Savings: ~40% faster than sequential execution

Multi-Table Join Orchestrator:
Query: “Join customers, sales, and products for full order analysis”

Detects: Complex multi-table join pattern Optimization: Single optimized JOIN query instead of multiple tool calls Savings: ~60% token reduction

TECHNIQUE 4: CONTEXT DISTILLATION

Ask: “List all products in the database.”

The query returns 500 products. That’s a lot of data.

Now ask a follow-up: “Which of those are Cardio equipment?”

Normally, the agent would send all 500 product records back to the LLM for analysis. That’s ~50,000 tokens of context.

Watch the optimization:

```
  OPTIMIZATION: CONTEXT DISTILLATION

Previous tool result: 500 products (48,000 tokens raw)
New query requires: Subset filtering by ProductType='Cardio'

Strategy: DISTILL large dataset before LLM processing
  - Apply deterministic filter: ProductType='Cardio'
  - Reduced dataset: 87 products
  - Context tokens: 48,000 → 4,200 (91% reduction)

Passing distilled context to LLM for response generation...
```

The agent performed a deterministic pre-filter, dramatically reducing context size before involving the LLM.

Tokens used: 5,800 total (42,200 saved) Cost: $0.028 ($2.10 saved on this single query)

Context distillation applies when: - Large tool results (>10,000 tokens) - Follow-up query requires subset or aggregation - Deterministic filtering possible without LLM reasoning

TECHNIQUE 5: MULTI-TOOL PARALLEL EXECUTION

Ask: “Give me the database version, table list, and current system date.”

The agent detects three independent requests:

```
  OPTIMIZATION: PARALLEL TOOL EXECUTION

Query decomposition:
  1. Database version → dba_databaseVersion (independent)
```

```
  2. Table list → base_tableList (independent)
  3. Current date → TDA_CurrentDate (independent)

No dependencies detected. Executing in parallel...

Parallel execution:
  [Thread 1] dba_databaseVersion → 0.8s
  [Thread 2] base_tableList → 1.2s
  [Thread 3] TDA_CurrentDate → 0.1s

  Total duration: 1.2s (longest thread)

Sequential execution would take: 2.1s
Time saved: 0.9s (43% faster)
```

When multiple tools have no data dependencies, the Fusion Optimizer executes them concurrently.

OPTIMIZATION IMPACT:

Intelligence → Optimization Metrics (30 days)

```
Plan Hydration:
  Applied: 89x | Tokens: 78K saved | Cost: $4.20 saved

Tactical Fast Path:
  Applied: 134x | Tokens: 195K saved | Cost: $10.40 saved

Specialized Orchestrators:
  Applied: 23x | Tokens: 142K saved | Cost: $7.60 saved

Context Distillation:
  Applied: 12x | Tokens: 380K saved | Cost: $18.90 saved

Parallel Execution:
  Applied: 34x | Time: 41s saved

--- TOTAL IMPACT ---
Tokens: 795,000 saved (32% of usage)
Cost: $41.10 saved (34% of spend)
Time: 556s saved (9.3 minutes)
```

Automatic. Transparent. Continuous.

No configuration. No enabling. Built into Fusion Optimizer core.

THE SHIFT:

```
Dollars  → Cents
Slow     → Fast
Wasteful → Efficient
```

Proactive optimization. Works smarter, not harder.

Screen Capture Plan:
- [ ] Query: "Top 5 products by revenue"
- [ ] Results displayed with token/cost metrics
- [ ] Follow-up: "What about the top 10?"
- [ ] Live Status Panel: " PLAN HYDRATION DETECTED" message
- [ ] Optimization strategy explanation displayed
- [ ] Hydrated execution with reduced tokens (950 vs 1,800)
- [ ] Query: "What's the current date?"
- [ ] Live Status Panel: " TACTICAL FAST PATH" message
- [ ] Direct tool execution (no strategic planning)
- [ ] Result in 0.2s with token comparison
- [ ] Query: "Show daily sales for the past week"
- [ ] Live Status Panel: " DATE RANGE ORCHESTRATOR" detected
- [ ] Optimization comparison: 12,000 tokens naive vs 2,800 optimized
- [ ] Single aggregated SQL query executed
- [ ] Query: "List all products" → 500 results
- [ ] Follow-up: "Which are Cardio equipment?"
- [ ] Live Status Panel: " CONTEXT DISTILLATION" message
- [ ] Context reduction: 48,000 → 4,200 tokens (91% reduction)
- [ ] Query: "Give me version, tables, and date"
- [ ] Live Status Panel: " PARALLEL TOOL EXECUTION" message
- [ ] Three threads executing simultaneously
- [ ] Duration comparison: 1.2s parallel vs 2.1s sequential
- [ ] Intelligence → Optimization Metrics dashboard
- [ ] 30-day summary with savings breakdown
- [ ] Total impact: 795,000 tokens saved, $41.10 saved

Supporting Materials:
- Optimization techniques documentation
- Plan hydration pattern recognition guide
- Fast path eligibility criteria
- Orchestrator catalog
- Context distillation algorithms
- Parallel execution dependency analyzer
- Performance benchmarks

Supporting Materials:
- Optimization techniques documentation
- Performance benchmarks
- Orchestrator examples

4.4 Cost Tracking: See Every Token, Every Cent
Duration: 3-4 minutes
Learning Objectives:

```
- Monitor per-turn token usage
- Understand cost-to-token mapping
- Learn cost optimization strategies

**Key Topics to Cover:**
- [ ] Per-turn token breakdown (input/output)
- [ ] Provider-specific pricing
- [ ] Historical token trends
- [ ] Optimization insights
- [ ] RAG efficiency savings attribution

**Narration Script:**
```

Optimizations powerful. But how know they're saving money?

Uderia: **Complete cost transparency. Every level.**

Token usage: Mystery → Actionable data.

TURN-LEVEL COST VISIBILITY:

Session History panel. Any turn.

Below each query-response:

```
Turn 3: "Top revenue products?"

  TOKENS:
   Input:  1,850
   Output:   340
   Total:  2,190

  COST:
   Model: Claude Sonnet 3.5
   Input:  $0.0055 (1,850 × $0.003/1K)
   Output: $0.0051 (340 × $0.015/1K)
   Total:  $0.0106

  EFFICIENCY:
   RAG-guided: YES
   Optimization: Plan hydration
   Baseline: 3,200 tokens
   Saved: 1,010 tokens (32%)
   Cost saved: $0.0034
```

Exact consumption. Exact cost. Not estimates. Actual.

PROVIDER PRICING:

Uderia tracks all models accurately:

```
Claude (Anthropic):
  Opus:      $15/$75 per 1M
```

```
  Sonnet 3.5: $3/$15 per 1M
  Haiku:     $0.25/$1.25 per 1M

GPT-4 (OpenAI):
  GPT-4o:      $2.50/$10 per 1M
  GPT-4o-mini: $0.15/$0.60 per 1M

Gemini (Google):
  1.5 Pro ( 128K): $1.25/$5 per 1M
  1.5 Pro (>128K): $2.50/$10 per 1M
  1.5 Flash:      $0.075/$0.30 per 1M

AWS Bedrock:
  Various models + inference profiles
  Custom pricing tracked

Ollama (Local):
  $0.00 (local execution)
```

Every interaction: Actual model + Actual pricing tier.

SESSION-LEVEL COST SUMMARY:

At the top of every session, you see cumulative costs:

```
  SESSION SUMMARY: "Q4 Revenue Analysis"

Turns completed: 12
Total tokens: 28,450 (input: 19,200 | output: 9,250)
Total cost: $0.2134

Average per turn:
  Tokens: 2,371
  Cost: $0.0178

Most expensive turn: Turn 7 - "Complex multi-table analysis" ($0.0456)
Most efficient turn: Turn 3 - "Simple product lookup" ($0.0032)

Optimization impact:
  Baseline (no optimization): $0.3287
  Actual (with optimization): $0.2134
  Savings: $0.1153 (35%)
```

You can see exactly where costs are concentrated and where optimizations delivered value.

HISTORICAL TOKEN TRENDS:

Go to Intelligence → Token Analytics.

You'll see charts showing usage over time:

30-Day Token Usage:

```
    Tokens (thousands)
    ↑
50
40
30
20
10
                        →
     Day 1    5    10   15   20   25   30
```

You can spot trends: - Usage spike on Day 23 (what happened?) - Declining baseline after Day 15 (optimizations kicking in?) - Consistent daily average ~25K tokens

Cost Trends:

```
    Cost ($)
    ↑
2.0
1.5
1.0
0.5
                        →
     Day 1    5    10   15   20   25   30

Total 30-day spend: $34.20
Average daily: $1.14
Projected monthly: $35.04
```

This helps with budgeting and cost forecasting.

PROVIDER COST COMPARISON:

Now here's where it gets strategic. Go to Intelligence → Provider Comparison.

You can run the same workload on different models and see actual cost differences:

```
WORKLOAD: "Standard Revenue Analysis" (10 typical queries)


  Provider/Model            Tokens     Cost      Duration

  Claude Opus               24,500     $0.487    8.2s
  Claude Sonnet 3.5         24,500     $0.097    7.8s
  GPT-4o                    26,100     $0.131    9.1s
  Gemini 1.5 Pro            25,800     $0.077    6.9s
  Gemini 1.5 Flash          25,800     $0.010    4.2s
  Ollama (Llama 3.1 70B)    28,200     $0.000    12.3s


Insights:
- Gemini Flash: Lowest cost, fastest (4.2s), great for batch jobs
- Claude Sonnet 3.5: Best balance of cost ($0.097) and quality
```

```
- Claude Opus: 5x more expensive than Sonnet for similar results
- Ollama: Zero cost but slowest (acceptable for non-urgent queries)

Recommendation: Use Gemini Flash for routine queries (@COST profile),
                Claude Sonnet for complex analysis (@PROD profile),
                Ollama for sensitive data (@LOCAL profile)
```

This data-driven comparison helps you optimize your profile strategy.

RAG EFFICIENCY SAVINGS:

Back to Intelligence → RAG Efficiency.

The RAG system tracks how much it saves you:

```
RAG COST IMPACT ANALYSIS:

Total queries: 247
RAG-guided queries: 183 (74%)

Without RAG (projected baseline):
  Average tokens per query: 3,612
  Total tokens: 892,164
  Total cost: $52.30

With RAG (actual usage):
  Average tokens per query: 2,611
  Total tokens: 645,013
  Total cost: $37.60

RAG SAVINGS:
  Tokens saved: 247,151 (28%)
  Cost saved: $14.70 (28%)

Cost per query:
  Without RAG: $0.212
  With RAG: $0.152
  Savings per query: $0.060

Projected annual savings (at current usage): $876
```

The RAG system pays for itself through continuous optimization.

OPTIMIZATION INSIGHTS:

Every cost display includes optimization context:

```
  COST OPTIMIZATION TIPS:

1. Turn 7 used Claude Opus ($0.0456) for a simple lookup.
   → Could use Gemini Flash ($0.0018) for 96% cost reduction
```

```
2. 34% of queries are suitable for fast-path execution.
   → Enable tactical fast path for ~$12/month savings

3. Session "Daily Reports" runs at peak hours with Sonnet.
   → Switch to Gemini Flash off-peak for 90% cost reduction

4. 12 queries loaded >10K context unnecessarily.
   → Enable context distillation for ~$8/month savings
```

Actionable recommendations based on your actual usage patterns.

COST ALERTS:

You can set spending alerts:

```
 COST ALERT CONFIGURATION:

 Daily spending exceeds $5
 Session cost exceeds $1
 Single query exceeds $0.50
 Monthly projected spend exceeds $150

Alert method: Email + Dashboard notification
```

If a query unexpectedly balloons in cost, you'll know immediately.

EXPORT AND REPORTING:

All cost data is exportable:

```
EXPORT OPTIONS:
- CSV (for Excel analysis)
- JSON (for API integration)
- PDF (for stakeholder reports)

Filters:
- Date range
- User (multi-user environments)
- Session
- Model/Provider
- Cost threshold
```

Generate monthly reports for finance. Track ROI. Justify infrastructure costs.

THE BOTTOM LINE:

From individual tokens to organizational budgets, Uderia gives you complete visibility:

- Turn-level: See every token, every cent
- Session-level: Track cumulative costs per workflow
- Historical: Identify trends and forecast spending
- Comparative: Choose optimal models based on real data
- RAG-attributed: Measure continuous improvement ROI
- Optimization-aware: Understand what's saving money

No hidden costs. No surprises. Complete financial transparency.

Because in enterprise AI, trust requires knowing exactly what you're paying for.

Every token. Every penny. Every optimization. All visible. All actionable.

This is financial governance for the AI age.

```
**Screen Capture Plan:**
- [ ] Session History with turn-level cost displays
- [ ] Turn 3 expanded: Token breakdown (input/output)
- [ ] Cost breakdown with model-specific pricing
- [ ] Efficiency section showing RAG guidance and optimization savings
- [ ] Provider pricing reference table (Claude, GPT, Gemini, Bedrock, Ollama)
- [ ] Session summary at top: 12 turns, $0.2134 total
- [ ] Average per turn, most expensive turn, most efficient turn
- [ ] Optimization impact: $0.3287 baseline vs $0.2134 actual
- [ ] Intelligence → Token Analytics dashboard
- [ ] 30-day token usage chart with trend line
- [ ] 30-day cost trend chart
- [ ] Projected monthly spend calculation
- [ ] Intelligence → Provider Comparison
- [ ] Workload comparison table across 6 providers
- [ ] Cost range: $0.487 (Opus) to $0.000 (Ollama)
- [ ] Recommendation box with strategic guidance
- [ ] Intelligence → RAG Efficiency dashboard
- [ ] RAG cost impact: $52.30 baseline vs $37.60 actual
- [ ] Savings: $14.70 (28%), projected annual: $876
- [ ] Optimization insights panel with 4 actionable tips
- [ ] Cost alert configuration interface
- [ ] Export options: CSV, JSON, PDF with filters

**Supporting Materials:**
- Cost tracking documentation
- Provider pricing reference (updated monthly)
- Cost optimization strategies guide
- ROI calculation methodology
- Budget forecasting tools
- Cost alert setup guide
- Financial reporting templates

**Supporting Materials:**
- Token tracking documentation
- Cost optimization guide
- Pricing model reference

---
```

```
## Module 5: SOVEREIGNTY - Your Data, Your Rules
**Goal:** Demonstrate ultimate flexibility in data exposure and infrastructure choice

### 5.1 Multi-Provider LLM Support: Freedom to Choose
**Duration:** 3-4 minutes
**Learning Objectives:**
- Explore supported LLM providers
- Learn provider switching mechanics
- Understand use cases for different providers

**Key Topics to Cover:**
- [ ] Cloud hyperscalers (Google, Anthropic, OpenAI, Azure)
- [ ] AWS Bedrock support
- [ ] Friendli.AI integration
- [ ] Ollama for local execution
- [ ] Dynamic provider switching
- [ ] Live model refresh

**Narration Script:**
```

Most AI platforms lock you into one provider. Choose Claude or GPT-4 or Gemini—you're stuck.

Vendor lock-in. No flexibility. No negotiating leverage.

Uderia breaks this constraint. True multi-provider freedom.

THE PROVIDER LANDSCAPE:

Setup → LLM Providers. Uderia's ecosystem:

```
PROVIDER ECOSYSTEM:

  Cloud Hyperscalers:
    • Anthropic (Claude Opus, Sonnet, Haiku)
    • OpenAI (GPT-4o, GPT-4o-mini)
    • Google (Gemini 1.5 Pro, Flash)
    • Azure OpenAI (Enterprise models)

  Managed Inference:
    • AWS Bedrock (foundation models + inference profiles)
    • Friendli.AI (optimized inference)

  Sovereign Execution:
    • Ollama (Llama 3.1, Mistral, Qwen - fully local)
```

Every provider. Every model. One platform. Your choice.

ADDING A NEW PROVIDER:

Add AWS Bedrock. Click "Add Provider" → "AWS Bedrock."

```
BEDROCK CONFIGURATION:
```

```
  Region: us-east-1
  Access Key ID: [encrypted at rest]
  Secret Access Key: [SHA256 hashing + salt]
  Inference Profile: [optional cross-region routing]

Save → Bedrock available
```

MODEL SELECTION:

Setup → Profiles → Edit @PROD.

LLM Provider dropdown:

```
AVAILABLE PROVIDERS:
• Claude Sonnet 3.5 (Anthropic)
• GPT-4o (OpenAI)
• Gemini 1.5 Pro (Google)
• Claude Sonnet 3.5 v2 (AWS Bedrock)
• Llama 3.1 70B (Ollama - Local)
```

One dropdown. Six infrastructure options.

Select “Claude Sonnet 3.5 v2 (AWS Bedrock)” → Save. **DYNAMIC PROVIDER SWITCHING:**

Flexibility in action. Query with @PROD (AWS Bedrock):

```
QUERY: "Q4 revenue trends by product category?"

@PROD Execution:
  Provider: AWS Bedrock
  Model: Claude Sonnet 3.5 v2
  Region: us-east-1
  Cost: $0.0112
```

Switch to @ANALYST (Google Gemini Flash). Same query:

```
@ANALYST Execution:
  Provider: Google AI
  Model: Gemini 1.5 Flash
  Cost: $0.0018 (6x cheaper)
```

Same question. Different provider. Different cost.

You’re not locked in. You’re orchestrating strategically. Execution: - Provider: Google AI - Model: Gemini 1.5 Flash - Cost: $0.0018

Same question. Same Uderia platform. Different provider. Different cost. Different infrastructure characteristics (Gemini is 6x cheaper, but less capable on complex reasoning).

You’re not locked in. You’re orchestrating providers strategically.

USE CASES FOR DIFFERENT PROVIDERS:

```
STRATEGIC PROVIDER SELECTION:

Anthropic Direct → Latest Claude features, fastest updates
```

```
AWS Bedrock → Enterprise compliance, cross-region routing
Google AI → Cost optimization, fast inference, multimodal
OpenAI → GPT-4 reasoning, specific requirements
Ollama Local → Complete sovereignty, zero cloud cost
```

LIVE MODEL REFRESH:

Providers release new models constantly. Uderia supports live refresh.

Setup → LLM Providers → "Refresh Models."

Uderia queries the provider API. Updates dropdown immediately.

No platform restart. No config editing. Just click.

Anthropic released Claude Opus 4? Refresh → Available. OpenAI released GPT-5? Refresh → Ready. **COST ARBITRAGE:**

Strategic advantage: price competition between providers.

```
PRICING COMPARISON (per 1M tokens):

Model                          Input / Output

Claude Sonnet 3.5 (Anthropic)  $3 / $15
Claude Sonnet 3.5 v2 (Bedrock) $3 / $15 (volume discounts)
Gemini 1.5 Pro (Google)        $1.25 / $5 (comparable quality)
```

Test identical queries across providers. Measure quality. Choose optimal cost-to-quality ratios. **PROVIDER FAILOVER:**

Bonus: provider outage? Switch profiles and continue.

Anthropic API down? Switch @PROD (Anthropic) → @BACKUP (AWS Bedrock Claude).

Same model. Different infrastructure. Zero downtime.

Multi-provider sovereignty. Not vendor lock-in. Not single-point-of-failure.

From "stuck with one vendor" to "orchestrating the entire AI ecosystem."

Your data. Your infrastructure. Your choice.

From "stuck with one vendor" to "orchestrating the entire AI ecosystem."

Your data. Your infrastructure. Your choice.

```
**Screen Capture Plan:**
- [ ] Setup → LLM Providers landing page showing 6+ providers
- [ ] "Add Provider" button → AWS Bedrock configuration form
- [ ] Credential entry (Access Key, Secret Key) with encryption indicator
- [ ] Save provider → Success confirmation
- [ ] Setup → Profiles → Edit @PROD profile
- [ ] LLM Provider dropdown showing 6 options across providers
- [ ] Select "Claude Sonnet 3.5 v2 (AWS Bedrock)"
```

```
- [ ] Query execution showing "Provider: AWS Bedrock" in metadata
- [ ] Switch to @ANALYST profile (Gemini Flash)
- [ ] Same query execution showing "Provider: Google AI" in metadata
- [ ] Cost comparison: $0.0112 (Bedrock) vs $0.0018 (Gemini)
- [ ] "Refresh Models" button demonstration
- [ ] New models appearing in dropdown after refresh
- [ ] Provider comparison table with pricing across all options

**Supporting Materials:**
- Multi-provider documentation
- Provider setup guides per platform (AWS, Google, Anthropic, etc.)
- Credential encryption security documentation
- Cost comparison calculator
- Provider failover strategies
- Model refresh documentation

---

### 5.2 Hybrid Intelligence: Cloud Power + Local Privacy
**Duration:** 4-5 minutes
**Learning Objectives:**
- Understand hybrid architecture
- See cloud planning + local execution
- Learn champion case injection for local models

**Key Topics to Cover:**
- [ ] Hybrid architecture concept (strategic planner vs executor)
- [ ] Cloud LLM for planning intelligence
- [ ] Local Ollama for private execution
- [ ] Champion case injection to boost local model performance
- [ ] Complete data sovereignty

**Narration Script:**
```

The impossible choice every organization faces: cloud intelligence or data sovereignty.

Pick one.

Uderia says: Why not both?

Hybrid Intelligence—cloud reasoning with zero-trust privacy.

THE IMPOSSIBLE CHOICE:

```
TRADITIONAL APPROACH:

Option A: Cloud Models (Claude, GPT-4)
  Intelligence:   Brilliant
  Data:   Sent to external APIs
  Sovereignty:   Sacrificed
```

```
Option B: Local Models (Ollama)
  Intelligence:  Mediocre
  Data:  Kept private
  Sovereignty:  Maintained
```

Binary choice. Intelligence OR privacy. Never both.

UDERIA'S HYBRID ARCHITECTURE:

Uderia splits the AI agent into two layers:

```
LAYER 1: Strategic Planner (Cloud)
  Runs: Claude, GPT-4, Gemini
  Receives: Question + tool names + schemas
  Does NOT receive: Actual data, PII, results
  Generates: Strategic plan, SQL templates
  Sends: Intelligence back to your infrastructure

LAYER 2: Tactical Executor (Local/Sovereign)
  Runs: Ollama on YOUR infrastructure
  Receives: Strategic plan + MCP tools + actual data
  Executes: SQL against YOUR database
  Data: Never leaves your environment
  Returns: Results to you (never to cloud)
```

Cloud does the thinking. Local does the data access.

DEMONSTRATION: @LOCAL PROFILE WITH HYBRID MODE

Switch to @LOCAL profile.

```
PROFILE CONFIGURATION:

Strategic Planner: Claude Sonnet 3.5 (Cloud - Anthropic)
Tactical Executor: Llama 3.1 70B (Local - Ollama)
MCP Tools: Local Teradata connector
Mode: Hybrid Intelligence
```

SCENARIO: Analyzing Customer Churn with PII

Ask: "Identify customers who stopped purchasing in Q4, show their names, emails, addresses, and analyze common patterns for churn risk."

This requires: - Sensitive PII (names, emails, addresses) - Multi-table joins (Customers, Sales) - Complex pattern analysis

Live Status Panel shows: - Sensitive PII (names, emails, addresses) - Multi-table joins (Customers, Sales) - Complex pattern analysis (churn indicators)

Watch what happens in the Live Status Panel:

```
  STRATEGIC PLANNING (Cloud - Claude Sonnet 3.5):

Query received: "Identify customers who stopped purchasing in Q4..."
```

```
Data sent to cloud:
  - Query text:
  - Available tools: [base_readQuery, base_tableList, TDA_LLMTask]
  - Schema info: Customers(CustomerID, FirstName, LastName, Email, Address...)
                 Sales(SaleID, CustomerID, SaleDate...)
  - Actual customer data:   (NEVER SENT)

Strategic Plan Generated:
  Phase 1: Query customers with no Q4 purchases
    Tool: base_readQuery
    SQL Template: SELECT c.CustomerID, c.FirstName, c.LastName, c.Email, c.Address
                  FROM Customers c
                  LEFT JOIN Sales s ON c.CustomerID = s.CustomerID
                    AND EXTRACT(QUARTER FROM s.SaleDate) = 4
                  WHERE s.SaleID IS NULL

  Phase 2: Analyze patterns using TDA_LLMTask
    Input: Churned customer demographics
    Analysis: Identify common characteristics

Plan sent back to local environment.
```

```
CLOUD RECEIVED:
  Your question (not sensitive)
  Schema metadata (table/column names)
  Tool names (generic capabilities)

CLOUD NEVER RECEIVED:
  Customer names, emails, addresses
  Actual sales data
  Any PII or confidential information
```

Cloud generated brilliant strategic plan without ever touching your data.

```
  TACTICAL EXECUTION (Local - Ollama Llama 3.1 70B):

Strategic plan received from cloud.
Executing locally with full data access...

Phase 1 Execution:
  Tool: base_readQuery (LOCAL MCP connector)
  SQL: [Executing against local Teradata database]
  Results: 47 customers identified

  Sample data (STAYS LOCAL):
    - John Smith, john.smith@email.com, 123 Main St, Boston MA
    - Sarah Johnson, sarah.j@email.com, 456 Oak Ave, Seattle WA
    [...45 more customers with full PII]
```

```
Phase 2 Execution:
  Tool: TDA_LLMTask (LOCAL Ollama execution)
  Input: Churned customer demographics
  Analysis:
    - 68% are located in urban areas
    - Average age: 34-42 demographic
    - Common trait: High service ticket volume (avg 3.2 tickets per customer)
    - Purchase pattern: Previously bought Cardio equipment

  Churn indicator: Strong correlation with unresolved service issues

Final Report Generated (LOCAL):
  Churn risk strongly tied to poor customer service experience.
  Recommendation: Proactive outreach to customers with >2 open service tickets.

DATA FLOW:
1. Your question → Cloud (strategic planning)
2. Strategic plan ← Cloud (execution strategy)
3. Data access → Local only (PII never leaves)
4. Analysis → Local Ollama (cloud strategy)
5. Results → You (privacy maintained)

THE RESULT:
  Cloud-level strategic intelligence
  Complete data sovereignty
  Regulatory compliance (GDPR, HIPAA, SOC2)
  Zero compromise on capability
```

PERFORMANCE COMPARISON:

```
APPROACH COMPARISON:

Approach 1: Fully Cloud (Traditional)
  Quality: 95/100 (excellent)
  Privacy:   (PII exposed to third party)
  Compliance:   (violates data residency)
  Cost: $0.045

Approach 2: Fully Local (Traditional)
  Quality: 72/100 (struggles with planning)
  Privacy:   (completely local)
  Compliance:   (data never leaves)
  Cost: $0.00 (local execution)

Approach 3: Uderia Hybrid Intelligence
  Quality: 92/100 (cloud strategy + local execution)
  Privacy:   (data never leaves premises)
  Compliance:   (regulatory requirements met)
  Cost: $0.012 (planning only, execution free)
```

92% of full-cloud quality with 100% data sovereignty. Best of both worlds.

CHAMPION CASE INJECTION BOOST:

Here's the multiplier effect. Remember RAG champion cases from Module 4.2? **CHAMPION CASE INJECTION BOOST:**

Multiplier effect. RAG champion cases from Module 4.2?

Hybrid Intelligence with @LOCAL: local Ollama gets boosted by champion cases learned from cloud.

```
LEARNING SEQUENCE:
1. Cloud Claude generates brilliant plan
2. RAG captures as champion case
3. Next similar query: Cloud plan + RAG champion
4. Local Ollama receives: Strategy + past success
5. Result: Local performs at near-cloud quality
```

Local model isn't just executing. It's learning from cloud intelligence via RAG—without exposing your data.

THE BREAKTHROUGH:

```
COMPARISON MATRIX:

                    Intelligence  Privacy    Cost
Traditional Cloud                           $$$
Traditional Local                            $
Uderia Hybrid                               $$
```

From impossible choice to having it all.

Cloud-level reasoning. Zero-trust privacy. No compromise.

Your data stays home. Your intelligence reaches the cloud. Hybrid sovereignty.

```
**Screen Capture Plan:**
- [ ] Hybrid architecture diagram: Cloud Planner + Local Executor with data flow arrows
- [ ] @LOCAL profile configuration showing dual LLM setup
- [ ] Strategic Planner: Claude Sonnet 3.5 (Cloud)
- [ ] Tactical Executor: Ollama Llama 3.1 (Local)
- [ ] Customer churn query input with PII requirements
- [ ] Live Status Panel: Strategic Planning section (Cloud)
- [ ] Data sent to cloud: Schema  , Actual data
- [ ] Strategic plan with SQL template generated
- [ ] Live Status Panel: Tactical Execution section (Local)
- [ ] Local SQL execution with actual PII data
- [ ] Customer records with names, emails, addresses visible (LOCAL ONLY)
- [ ] Local Ollama analysis execution
- [ ] Results with churn patterns and recommendations
- [ ] Network traffic monitor showing: Outbound (plan request), Inbound (strategy), No data up
```

```
- [ ] Performance comparison table: Cloud vs Local vs Hybrid
- [ ] Quality scores: 95 vs 72 vs 92
- [ ] Privacy indicators:  vs  vs
- [ ] Cost comparison: $0.045 vs $0.00 vs $0.012
- [ ] Champion case boost diagram showing RAG injection to local model

**Supporting Materials:**
- Hybrid intelligence architecture documentation
- Data flow diagrams (with security annotations)
- Ollama setup guide for local execution
- Compliance certification documentation (GDPR, HIPAA, SOC2)
- Performance benchmark comparisons
- Champion case injection technical details

---

### 5.3 Security & Multi-User Isolation
**Duration:** 3-4 minutes
**Learning Objectives:**
- Understand JWT authentication
- Learn user isolation mechanisms
- Explore role-based access control

**Key Topics to Cover:**
- [ ] JWT-based authentication (24-hour expiry)
- [ ] User-specific session directories
- [ ] Database-level UUID isolation
- [ ] Role-based access (User, Developer, Admin)
- [ ] Encrypted credential storage
- [ ] Multi-user simultaneous support

**Narration Script:**
```

Data sovereignty isn't just where your data lives. It's who can access it.

Uderia: bank-grade security isolation for enterprise multi-user environments.

JWT AUTHENTICATION: SHORT-LIVED TOKENS

Login to Uderia? Not a permanent session cookie. JSON Web Token (JWT) with 24-hour expiration.

```
LOGIN SEQUENCE:
1. Enter username + password
2. Server validates (bcrypt-hashed, salted)
3. Server issues JWT (HS256 signed)
4. Token contains: user_id, username, role, expiration
5. Stored: httpOnly cookie (secure)

EVERY API REQUEST:
```

```
  Signature authentic?
  Not expired?
  Valid account status?

After 24 hours → Token expires → Re-authenticate
```

Security by design. Stolen token? Worthless after 24 hours.

USER-SPECIFIC SESSION ISOLATION:

Multi-user isolation. File system level.

```
SESSION DIRECTORIES:

tda_sessions/
   user_alice_82b9231b.../
       session_001.json
       session_002.json
   user_bob_f3a1c5d7.../
       session_001.json
   user_carol_a9b8c7d6.../
        session_001.json
```

UUID suffix? Security isolation.

Alice cannot access Bob's sessions. File system enforces user-specific paths.

```
API REQUEST: GET /api/sessions/user_bob_session_001
JWT TOKEN: user_id=alice
SERVER: 403 Forbidden (user_id mismatch)
```

Session data isolated: file system level AND application level.

DATABASE-LEVEL UUID ISOLATION: DATABASE-LEVEL UUID ISOLATION:

Database layer: PostgreSQL with UUID-based user identification.

Not sequential integers. UUIDs. Non-enumerable.

```
BAD (Sequential IDs):
SELECT * FROM consumption_logs WHERE user_id = 1001;
Attacker: "Try 1002, 1003, 1004..." (enumeration attack)

GOOD (UUID):
SELECT * FROM consumption_logs
WHERE user_id = '82b9231b-08b1-4b1e-bcdf-9762db21f90b';
Attacker: "Try... um... 82b9231b...90c?" (impossible)
```

Every table uses UUID foreign keys:

```
users(user_id UUID)
sessions(session_id UUID, user_id UUID)
consumption_logs(log_id UUID, user_id UUID)
rag_cases(case_id UUID, user_id UUID)
```

Even with SQL injection (doesn't exist—parameterized queries), enumerating users is cryptographically infeasible. **ROLE-BASED ACCESS CONTROL (RBAC):**

ROLE-BASED ACCESS CONTROL (RBAC):

Three roles with escalating privileges:

```
ROLE: User (Standard)
  Execute queries (UI + API)
  Manage own sessions
  Access own RAG collections
  View own consumption logs

ROLE: Developer
  All User permissions +
  Developer tools
  Custom MCP servers
  Debug mode + advanced logging

ROLE: Admin
  All Developer permissions +
  User management (create, delete, suspend)
  System configuration (LLM providers, MCP)
  Global consumption analytics
  Long-lived access tokens
  Audit log access
```

Role enforcement at every API endpoint:

```python
@app.route('/api/admin/users', methods=['GET'])
@require_role('Admin')
def list_all_users():
    # Only executes if JWT has role='Admin'
    return get_all_users()
```

Access admin endpoints without admin role? **403 Forbidden.**

When you configure LLM providers (Anthropic API keys, AWS credentials, etc.), where do those secrets go?

NOT in plaintext. Not in environment variables (easily leaked). In encrypted database columns.

Storage sequence: 1. User enters API key: `sk-ant-api03-xxxxxxxxxxx` 2. Server encrypts with AES-256: `U2FsdGVkX1+abcdef...` (cipher text) 3. Server stores cipher text in database 4. When needed: Server decrypts on-the-fly for API calls 5. Cipher text never leaves the server

Even if someone dumps the database, they get encrypted gibberish—useless without the encryption key (stored separately in secure environment variables).

This is how Uderia handles sensitive credentials: - LLM provider API keys: Encrypted - AWS access keys: Encrypted - Database connection strings: Encrypted - OAuth tokens: Encrypted

MULTI-USER SIMULTANEOUS SUPPORT:

Shared multi-user platform. Alice, Bob, Carol—all logged in simultaneously. No interference.

How?

```
SESSION ISOLATION:
Alice: tda_sessions/user_alice_.../session_001.json
Bob: tda_sessions/user_bob_.../session_001.json
No overlap. No cross-contamination.

MEMORY ISOLATION:
• Python processes handle requests independently
• No shared state between users
• Each API call validates JWT + user_id

DATABASE ISOLATION:
• All queries filtered by user_id UUID
• PostgreSQL row-level security policies
• Application layer blocks cross-user queries
```

AUDIT LOGGING:

Compliance and security monitoring. Audit logs:

```
AUDIT LOG ENTRIES:

[10:32:15] alice (82b9...) | LOGIN | IP: 192.168.1.45 | Success
[10:33:01] alice (82b9...) | QUERY_EXECUTE | session_001 | $0.0045
[10:45:22] bob (f3a1...) | LOGIN | IP: 192.168.1.67 | Success
[10:46:10] bob (f3a1...) | ACCESS_ADMIN_PANEL | DENIED (role=User)
[11:02:33] admin (a9b8...) | USER_CREATE | Target: carol | Role: User
```

Every sensitive action logged. Admins review:

```
  Failed login attempts (brute force detection)
  Unauthorized access attempts (security monitoring)
  User creation/deletion (accountability)
  Credential changes (suspicious activity)
```

THE SECURITY GUARANTEE:

```
UDERIA MULTI-USER ARCHITECTURE:

Authentication → JWT (24h expiration) + bcrypt hashing
Isolation → User-specific paths + UUID database keys
Authorization → RBAC (User, Developer, Admin)
Encryption → AES-256 (credentials) + SHA256 (tokens)
Auditability → Complete logging (compliance + forensics)
```

From “shared platform” to “bank-grade multi-tenant security.”

Your data. Your users. Your control. Complete isolation. Total security. From “shared platform” to “bank-grade multi-tenant security.”

Your data. Your users. Your control. Complete isolation. Total security.

Screen Capture Plan:
- [ ] Login page with credential entry
- [ ] JWT token structure visualization (decoded payload showing user_id, role, exp)
- [ ] File system view: tda_sessions/ directory with multiple user folders
- [ ] User folder showing UUID suffix for isolation
- [ ] API request attempt: Alice trying to access Bob's session → 403 Forbidden
- [ ] Database schema showing UUID columns (users, sessions, consumption_logs)
- [ ] UUID vs Sequential ID comparison diagram
- [ ] Admin panel: User Management interface
- [ ] Three users listed: alice (User), bob (Developer), admin (Admin)
- [ ] Role assignment dropdown
- [ ] Permissions matrix table showing role capabilities
- [ ] Developer trying to access admin endpoint → 403 Forbidden
- [ ] Setup → LLM Providers: API key entry
- [ ] Encryption indicator ( ) next to saved credential
- [ ] Database view showing encrypted credential (cipher text)
- [ ] Multi-user simultaneous query execution (split screen)
- [ ] Alice running revenue query (her session)
- [ ] Bob running customer query (his session)
- [ ] No interference between executions
- [ ] Admin → Audit Logs viewer
- [ ] Log entries showing: Logins, Query executions, Access denials
- [ ] Failed access attempt highlighted for security review

Supporting Materials:
- Security architecture documentation
- JWT authentication flow diagram
- Multi-user isolation technical specifications
- RBAC permissions matrix
- Credential encryption technical details (AES-256)
- Audit logging guide
- Compliance certifications (SOC2, ISO27001 readiness)
- Multi-tenant security best practices

Module 6: COLLABORATIVE - Intelligence Marketplace
Goal: Transform individual expertise into collective organizational knowledge

6.1 Marketplace Overview: Collective Intelligence
Duration: 3-4 minutes
Learning Objectives:
- Understand marketplace value proposition
- Learn dual repository architecture (Planner vs Knowledge)
- See community ecosystem vision

```
**Key Topics to Cover:**
- [ ] From isolated expertise to collective intelligence
- [ ] Planner Repositories ( execution patterns)
- [ ] Knowledge Repositories ( domain knowledge)
- [ ] Visual separation and organization
- [ ] Network effects and cost reduction

**Narration Script:**
```

Every organization faces the same problem.

Your data analyst discovers a brilliant query pattern for customer churn analysis. Three days of iteration. $45 in LLM tokens. Perfect results.

Next week, another analyst asks the same question. Starts from zero. Another three days. Another $45.

Reinventing the wheel.

Uderia's Marketplace breaks this cycle. Collective Intelligence—organizational knowledge becomes reusable assets.

THE MARKETPLACE VISION:

Marketplace icon → Uderia's dual repository architecture:

```
TWO TYPES OF INTELLIGENCE:

  PLANNER REPOSITORIES (Left Tab):
    • Execution patterns ("how to do it")
    • RAG champion cases from successful queries
    • Strategic plans, tool sequences, optimizations
    • Think: "Recipes for accomplishing tasks"

    Examples:
      Customer Churn Analysis Patterns (23 cases)
      Revenue Forecasting Templates (17 cases)
      Database Performance Playbooks (31 cases)

  KNOWLEDGE REPOSITORIES (Right Tab):
    • Domain expertise ("what you need to know")
    • Uploaded documents: Specs, guides, documentation
    • Chunked and embedded for RAG retrieval
    • Think: "Reference library your AI can read"

    Examples:
      GDPR Compliance Guidelines (47 chunks)
      SQL Optimization Best Practices (89 chunks)
      Teradata System Documentation (312 chunks)
```

VISUAL ORGANIZATION:

Collection cards show:

```
COLLECTION CARD:

Customer Churn Analysis Patterns
  Planner | Created by @ANALYST

23 items | 847 uses | 4.8

"Pre-validated patterns for identifying at-risk customers
using purchase history and service ticket correlation."

Tags: #churn #customer-analytics #predictive
```

Sort by: Most Popular | Top Rated | Recently Added | Most Relevant **THE VALUE PROPOSITION:**

Quantifying collective intelligence:

```
WITHOUT MARKETPLACE (Traditional):
Analyst A → Develops churn query → 3 days, $45
Analyst B → Same need, starts from scratch → 3 days, $45
Analyst C → Same need, starts from scratch → 3 days, $45

Total: 9 days, $135, 3x redundant work

WITH MARKETPLACE (Uderia):
Analyst A → Develops query → 3 days, $45 → Publishes
Analyst B → Searches → Finds pattern → Deploys → 5 min, $0.12
Analyst C → Same search → Same deploy → 5 min, $0.12

Total: 3 days + 10 min, $45.24 (67% savings), zero redundancy
```

Analyst A's effort → Organizational asset. B and C benefit immediately.ce - Analyst B: Searches marketplace → Finds "Customer Churn Patterns" → Deploys to profile → 5 minutes, $0.12 cost - Analyst C: Same search → Same deployment → 5 minutes, $0.12 cost

Total: 3 days + 10 minutes, $45.24 cost (67% savings), zero redundancy

Analyst A's effort becomes organizational asset. Analysts B and C benefit immediately.

NETWORK EFFECTS:

The more people contribute, the more valuable the marketplace:

```
MARKETPLACE GROWTH:

After 10 analysts:
  10 planner collections (200+ champion cases)
  10 knowledge collections (500+ document chunks)
  Every new analyst → Instant collective expertise

After 100 users:
  100+ collections (every common business question)
  Thousands of validated patterns
```

```
  New analysts productive on day one
  Zero ramp-up time for standard analyses
```

Value grows exponentially. Classic network effect.

COST REDUCTION AT SCALE:

Marketplace impact projection:

```
ORGANIZATION: 50 analysts

WITHOUT MARKETPLACE:
• Each analyst: 10 common queries/quarter
• Avg cost per query: $30 (LLM tokens + time)
• Total quarterly: 50 × 10 × $30 = $15,000
• Redundancy rate: ~70% (already solved elsewhere)

WITH MARKETPLACE:
• 30% developed from scratch: $4,500
• 70% reused from marketplace: $210 (deploy only)
• Total quarterly: $4,710
• Savings: $10,290 (69% reduction)
```

DISCOVERY & SEARCH:

Finding the right collection? Smart search.

Search: “How do I identify customers likely to churn?”

```
SEARCH RESULTS:

1. Customer Churn Analysis Patterns
     Planner | 23 cases | 4.8
   "Churn prediction using purchase gaps"

2. Customer Retention Best Practices
     Knowledge | 67 chunks | 4.6
   "Domain knowledge from CRM strategy documents"

3. Service Ticket Risk Indicators
     Planner | 14 cases | 4.7
**PROFILES AS MARKETPLACE CONSUMERS:**

@ANALYST profile from Module 2.2:
```

@ANALYST PROFILE: ├─ LLM: Gemini Flash (cost-optimized) ├─ Tools: TDA_Charting (visualizations) ├─ Deployed Planners: │ ├─ Customer Churn Patterns │ └─ Revenue Forecasting Templates └─ Subscribed Knowledge: ├─ Sales Strategy Playbooks └─ Product Documentation

```
Profiles aren't just LLM configs. They're marketplaces of deployed intelligence.

Activate @ANALYST → LLM + deployed patterns + subscribed knowledge.
```

```
One profile. Dozens of collections. Hundreds of expert patterns. Instant capability.ce.

When you activate @ANALYST, you're activating:
- The LLM (Gemini Flash)
**THE COLLABORATIVE SHIFT:**
```

TRADITIONAL AI: □ Each user isolated □ Every query starts from zero □ Knowledge dies when user leaves □ No organizational learning

UDERIA MARKETPLACE: □ Community of contributors □ Every query builds on past success □ Knowledge persists as organizational asset □ Continuous collective improvement

```
From "everyone for themselves" to "standing on each other's shoulders."

From isolated expertise to collective intelligence. From reinventing wheels to deploying prover

The Marketplace isn't a feature. It's a fundamental shift in how organizations learn.

From isolated expertise to collective intelligence. From reinventing wheels to deploying
proven patterns. From costly redundancy to efficient reuse.

The Marketplace isn't a feature. It's a fundamental shift in how organizations learn.
```

Screen Capture Plan: - [ ] Marketplace landing page with dual-tab interface - [ ] Planner Repositories tab (left) with collection cards - [ ] Knowledge Repositories tab (right) with collection cards - [ ] Collection card detail view showing: - Name, type badge (□ or □), author - Stats (23 items | 847 uses | 4.8□) - Description and tags - [ ] Sort options dropdown (Popular, Top Rated, Recent, Relevant) - [ ] Value proposition comparison chart: - Without Marketplace: 9 days, $135 - With Marketplace: 3.17 days, $45.24 (67% savings) - [ ] Network effects visualization: Value growth curve as user count increases - [ ] Cost reduction projection table for 50-analyst organization - [ ] Search bar with query: “identify customers likely to churn” - [ ] Search results showing 3 relevant collections with semantic matching - [ ] @ANALYST profile view showing deployed collections: - Planner: Customer Churn Patterns, Revenue Forecasting - Knowledge: Sales Strategy, Product Docs - [ ] Community statistics panel: - Total collections: 127 - Total contributors: 43 - Total deployments: 1,284 - Avg cost savings per reuse: $28.50

Supporting Materials: - Marketplace architecture documentation - Value proposition case studies - Network effects analysis - ROI calculator for marketplace adoption - Discovery and search algorithm documentation

---

6.2 Planner Repositories: Share Execution Patterns

Duration: 3-4 minutes
Learning Objectives: - Create templates from successful cases - Browse and discover planner collections - Deploy patterns to your repository

Key Topics to Cover: - [ ] “Create Template” from RAG case - [ ] Template metadata (name,

description, tags) - [ ] Browsing planner collections - [ ] “Deploy to My Repository” workflow - [ ] Pattern reuse and cost savings

Narration Script:

You just executed the perfect query.

20 minutes of refinement. Three tool iterations. $2.40 in LLM tokens. Flawless result.

Now what? Let it disappear into conversation history? Make someone else spend another 20 minut

No. Capture it. Templatize it. Share it.

Welcome to Planner Repositories.

WHAT ARE PLANNER REPOSITORIES?

Collections of execution patterns-RAG champion cases from successful queries shared across you

THINK OF THEM AS: • Cookbooks for AI agents (step-by-step recipes) • Playbooks for analysts (proven strategies) • Accelerators for new users (deploy expert patterns instantly)

EVERY PLANNER REPOSITORY CONTAINS: ├— Champion case metadata (query pattern, context, tools) ├— Strategic plan (high-level approach) ├— Tool execution sequence (tactical steps) └— Expected outcomes (what success looks like)

Not generic templates. Actual successful executions captured from real work.
CREATING A TEMPLATE FROM RAG CASE:

You just finished this query:

"Analyze Q4 revenue by product category, identify top 3 performers, and create a trend chart c

EXECUTION DETAILS: Tools: base_readQuery → TDA_LLMTask → TDA_Charting Cost: $1.85 Duration: 34 seconds Result: Perfect insights with professional chart

Worth saving. See "Create Template" button? Click it.orth saving. Look at the conversation tur
see the "Create Template" button?
Click it.
TEMPLATE CREATION FORM:

Uderia auto-populates:

TEMPLATE:

Name: “Quarterly Revenue Analysis with Trend Comparison”

Description: “Analyzes revenue by product category for specified quarter, identifies top performers, generates comparative trend chart.”

Category: Financial Analysis Tags: #revenue #quarterly-analysis #trending #visualization

Captured Elements: ├─ Query Pattern: “Analyze {time_period} revenue by {dimension}…” ├─ Tool Sequence: base_readQuery → TDA_LLMTask → TDA_Charting ├─ Context: Access to Sales and SaleDetails tables └─ Expected Cost: ~$1.50-$2.00 per execution

Privacy Level: ○ Public (anyone in org can discover) ○ Unlisted (link only) ○ Private (only you)

```
Choose **Public**. Click "Save Template."

20 minutes of work → Reusable organizational asset.
You just transformed 20 minutes of work into a reusable organizational asset.

**BROWSING PLANNER COLLECTIONS:**

Marketplace → Planner Repositories. Organizational knowledge base:
```

FEATURED COLLECTIONS:

1. Customer Insights Playbook | □ 23 patterns | 4.8□ Author: @ANALYST “Customer behavior analysis, churn prediction, segmentation” Top patterns: ├─ Customer Churn Risk Identification (142 uses) ├─ High-Value Customer Segmentation (98 uses) └─ Purchase Frequency Analysis (87 uses)
2. Database Performance Optimization | □ 31 patterns | 4.9□ Author: @DBA “Query optimization, index analysis, troubleshooting” Top patterns: ├─ Slow Query Diagnosis (234 uses) ├─ Index Recommendation Strategy (156 uses) └─ Table Statistics Refresh (89 uses)
3. Sales Analytics Accelerators | □ 17 patterns | 4.7□ Author: @ANALYST + @PROD “Revenue analysis, forecasting, product performance” Includes YOUR template (12 uses already!)

```
Click "Customer Insights Playbook" to explore.

**COLLECTION DETAIL VIEW:**

23 champion case patterns organized by category.
```

PATTERN PREVIEW:

Customer Churn Risk Identification

Query Template: “Identify customers with {condition} indicating churn risk in {timeframe}”

Tools: base_readQuery, base_tableList, TDA_LLMTask Avg Cost: $0.85 Deployments: 142 Rating: 4.9□

Preview: “Analyzes purchase gaps, service ticket volume, engagement drops to identify at-risk customers. Prioritized list with risk scores.”

```
Proven pattern-142 successful uses. Not experimenting. Deploying certified expertise.

**DEPLOYING A COLLECTION:**
```

```
**DEPLOYING A COLLECTION:**

Click "Deploy to My Repository."
```

DEPLOYMENT OPTIONS:

Target Profile: @ANALYST Deployment Mode: ○ Subscribe (reference-based, auto-updates) ○ Fork (independent copy, you control updates)

Subscribe means: □ Instant access to all 23 patterns □ New patterns added → You receive automatically □ Patterns improved → You benefit immediately □ No storage duplication

```
Click "Deploy." Done.

Activate @ANALYST → Intelligence View → Planner Repository:
```

YOUR REPOSITORY: ├— Your templates: (1 pattern) ├— Subscribed: Customer Insights Playbook (23 patterns) └— Total available: 24 patterns

```
@ANALYST profile just became 24x more capable. Instantly.

Let's test it. Ask a new query:

"Which customers haven't purchased in 60 days and have multiple service tickets?"

Watch the RAG system:
```

□ RAG RETRIEVAL:

Searching planner repository for relevant patterns…

Match found: Collection: Customer Insights Playbook Pattern: “Customer Churn Risk Identification” Similarity: 0.94 (excellent match)

Retrieved champion case: Query Pattern: “Identify customers with {no recent purchases} and {high service volume}” Strategic Approach: 1. Query Sales table for customers with SaleDate < (TODAY - 60 days) 2. Join with ServiceTickets table 3. Filter WHERE ticket_count >= 2 4. Analyze demographic patterns Tool Sequence: base_readQuery → TDA_LLMTask Expected Cost: ~$0.85

Injecting champion case into context…

```
Your query matched a deployed pattern. The LLM receives:
- Your question
- A proven strategy for similar questions
- Expected tool sequence
- Cost estimate

Execution:
- Uses champion case guidance
- Adapts to your specific query (60 days, multiple tickets)
- Executes with proven approach
```

- Cost: $0.91 (within expected range)
- Result: Perfect

THE MULTIPLICATION EFFECT:

COMPARISON:

Traditional Approach: ├─ You ask churn question ├─ AI starts from scratch, experiments ├─ Cost: $2.50 (trial and error) ├─ Time: 45 seconds (multiple iterations) └─ Quality: 7/10 (decent but not optimal)

With Deployed Planner Repository: ├─ You ask churn question ├─ RAG finds "Customer Churn Risk Identification" ├─ AI executes proven strategy immediately ├─ Cost: $0.91 (64% savings) ├─ Time: 18 seconds (single execution) └─ Quality: 9.5/10 (expert-level, first try)

Not just saving cost. Executing at expert level immediately.

You're not just saving cost. You're executing at expert level immediately.

CONTRIBUTION CYCLE:

ORGANIZATIONAL LEARNING AT AI SPEED:

1. You execute 10 queries this week
2. 3 are novel, high-quality results
3. You click "Create Template" for all 3
4. You publish to "Sales Analytics Accelerators"
5. 50 colleagues have that collection deployed
6. Tomorrow → All 50 benefit from your work
7. They add their own successful patterns
8. Next week → You benefit from their contributions

Every success → Collective knowledge. Every pattern deployment → Multiplies impact.

ROI OF PLANNER REPOSITORIES:

ORGANIZATION: 30 analysts, 1000 queries/month

WITHOUT PLANNER REPOSITORIES: • Avg cost per query: $2.80 (trial-and-error) • Monthly cost: $2,800 • Redundant development: ~40% waste

WITH PLANNER REPOSITORIES: • Pattern reuse rate: 60% (600 queries) • Pattern-matched cost: $1.10 (60% savings) • Novel query cost: $2.80 (same) • Monthly cost: (600 × $1.10) + (400 × $2.80) = $1,780 • Savings: $1,020/month (36% reduction)

Planner Repositories eliminate redundancy and multiply expertise.

From "starting from scratch" to "standing on organizational giants' shoulders."

Capture success. Templatize excellence. Deploy expertise. Multiply impact.

Screen Capture Plan: - [ ] Successful query conversation with “Create Template” button visible - [ ] Click “Create Template” → Form auto-populated - [ ] Template creation form showing: - Name: “Quarterly Revenue Analysis with Trend Comparison” - Description (multi-line) - Category: Financial Analysis - Tags: #revenue #quarterly-analysis #trending - Captured elements preview (query pattern, tool sequence) - Privacy level radio buttons (Public selected) - [ ] Click “Save Template” → Success confirmation - [ ] Navigate to Marketplace → Planner Repositories tab - [ ] Collection cards showing: - Customer Insights Playbook (23 patterns, 4.8□) - Database Performance Optimization (31 patterns, 4.9□) - Sales Analytics Accelerators (17 patterns, 4.7□) - [ ] Click “Customer Insights Playbook” → Collection detail view - [ ] Pattern list showing 23 patterns organized by category - [ ] Pattern preview card: - “Customer Churn Risk Identification” - Query template, tools, cost, deployments (142), rating (4.9□) - [ ] “Deploy to My Repository” button - [ ] Deployment options modal: - Target Profile: @ANALYST dropdown - Mode: Subscribe vs Fork radio buttons - [ ] Click “Deploy” → Success confirmation - [ ] Switch to @ANALYST profile - [ ] Intelligence View → Planner Repository tab - [ ] Repository list showing: - Your templates (1) - Subscribed: Customer Insights Playbook (23) - Total: 24 patterns available - [ ] New query: “Which customers haven’t purchased in 60 days…” - [ ] Live Status Panel → RAG Retrieval section showing: - Pattern match: “Customer Churn Risk Identification” (0.94 similarity) - Champion case details injected - [ ] Query execution with pattern guidance - [ ] Cost comparison: $0.91 vs expected $0.85 (within range) - [ ] ROI calculator showing 36% monthly savings for 30-analyst org

Supporting Materials: - Planner repository architecture documentation - Template creation best practices guide - Pattern quality guidelines - Deployment workflow technical specifications - ROI calculator for planner repository adoption - Contribution cycle and organizational learning analysis

---

6.3 Knowledge Repositories: Share Domain Expertise

Duration: 3-4 minutes
Learning Objectives: - Upload documents to knowledge collections - Publish knowledge for community access - Deploy knowledge to your environment

Key Topics to Cover: - [ ] Creating knowledge collections - [ ] Document upload (PDF, TXT, DOCX, MD) - [ ] Chunking strategies - [ ] Publishing to marketplace - [ ] Subscribing to expert knowledge

Narration Script:

```
Planner Repositories capture execution patterns-the "how to do it."

But what about domain expertise? Product specifications? Compliance regulations? Technical docu

Knowledge Repositories. The second pillar of Uderia's marketplace-shareable, searchable, AI-rea

**WHAT ARE KNOWLEDGE REPOSITORIES?**

Collections of uploaded documents-chunked, embedded, indexed for RAG retrieval. Your AI agents
```

THINK OF THEM AS: • Reference libraries your AI reads instantly • Institutional memory that never forgets • Expert knowledge accessible to every user

DIFFERENCE FROM PLANNER REPOSITORIES: Planners → Execution patterns (“how to analyze churn”) Knowledge → Domain facts (“what is churn, why, benchmarks”)

```
Different architecture. Different purpose. Both essential.

**CREATING A KNOWLEDGE COLLECTION:**

Marketplace → Knowledge Repositories → "Create Collection."
```

COLLECTION SETUP:

Name: “Fitness Equipment Product Catalog”

Description: “Complete product specifications, pricing history, warranty terms, technical details for all fitness equipment.”

Category: Product Knowledge Tags: #products #specifications #warranty #pricing

Visibility: ○ Public (all users in org) ○ Unlisted (link only) ○ Private (only you)

```
Choose **Public**-benefits entire sales and support team.

Click "Create Collection." Empty collection ready for documents.

**DOCUMENT UPLOAD:**
**DOCUMENT UPLOAD:**

Click "Upload Documents."
```

SUPPORTED FORMATS: PDF | TXT | DOCX | MD | CSV (experimental)

YOUR DOCUMENTS: 1. product_catalog_2024.pdf (47 pages, product specs) 2. warranty_policies.docx (12 pages, terms) 3. pricing_history_q4.txt (pricing changes)

VALIDATION: □ product_catalog_2024.pdf (2.3 MB, 47 pages) □ warranty_policies.docx (487 KB, 12 pages) □ pricing_history_q4.txt (23 KB, plain text)

```
All accepted. Click "Process Documents."
**CHUNKING CONFIGURATION:**

Before AI-searchable, documents need chunking-split large documents into smaller, semantically

Why?
- LLMs have context limits (can't ingest 47-page PDF in one shot)
- Retrieval precision improves (find exact paragraph, not entire document)
- Embedding efficiency (smaller chunks = better semantic matching)
```

CHUNKING STRATEGIES:

1. Fixed Size (Default) Size: 1000 chars | Overlap: 200 chars Best for: General documents, mixed content

2. Semantic Paragraphs Size: ~800 chars (variable) | By paragraph boundaries Best for: Well-structured documents
3. Hybrid (Recommended) Semantic paragraphs + max 1200 char limit | Overlap: 150 Best for: Mixed documents

```
Choose **Hybrid**. Click "Start Chunking."

**PROCESSING STATUS:**
```

 DOCUMENT PROCESSING:

Doc 1: product_catalog_2024.pdf ├─ Extracting… (47 pages, 34,521 chars) ├─ Chunking… (89 chunks) ├─ Embedding… (text-embedding-ada-002) └─ Storing… | COMPLETE (12s)

Doc 2: warranty_policies.docx ├─ Extracting… (12 pages, 8,943 chars) ├─ Chunking… (23 chunks) └─ Complete | 4s

Doc 3: pricing_history_q4.txt ├─ Reading… (1,876 chars) ├─ Chunking… (5 chunks) └─ Complete | 2s

 SUMMARY: Total: 3 docs | 117 chunks | 45,340 chars Avg chunk: 387 chars | Processing: 18s Status: READY FOR USE

```
Collection live. 117 searchable chunks. Fully indexed. Ready for RAG retrieval.

**PUBLISHING TO MARKETPLACE:**

Now make it discoverable. Click "Publish to Marketplace."

**Publication Settings:**

**Visibility:** Public (already set)

**Featured Tags:** #products #specifications #warranty #pricing (already set)

**PUBLISHING TO MARKETPLACE:**

Make it discoverable. Click "Publish to Marketplace."
```

PUBLICATION SETTINGS:

Visibility: Public Tags: #products #specifications #warranty #pricing

Purpose: “Enables sales and support teams to answer customer questions about product specs, warranty, pricing with AI accuracy.”

Recommended For: • Sales Representatives (product questions) • Customer Support (warranty inquiries) • Marketing Team (pricing strategies)

```
Click "Publish." Done.
```

```
Knowledge Repository now live in marketplace. Anyone in organization can discover it, subscribe
   - Author: Legal Department
   - Rating: 4.9  (31 reviews)
**SUBSCRIBING TO COMMUNITY KNOWLEDGE:**

Consumer perspective. Marketplace → Knowledge Repositories.
```

FEATURED COLLECTIONS:

1. GDPR Compliance Guidelines | □ 47 chunks | 4.9□ Author: Legal Department | 127 deployments “Complete GDPR regulations, requirements, compliance checklists”
2. SQL Optimization Best Practices | □ 89 chunks | 4.8□ Author: @DBA | 203 deployments “Teradata optimization, index strategies, 10 years DBA experience”
3. Fitness Equipment Product Catalog | □ 117 chunks | NEW Author: You | 0 deployments “Product specs, pricing history, warranty terms…” ← YOUR COLLECTION

```
Click "SQL Optimization Best Practices" to explore.
- **Sample chunks:**
  - "When designing composite indexes, column order matters significantly. Place the
    most selective columns first to maximize index efficiency..."
  - "COLLECT STATISTICS is not optional in Teradata. Without statistics, the optimizer
    makes uninformed decisions leading to poor query plans..."

This is 10 years of DBA expertise, distilled into 89 searchable chunks. Available instantly.

**DEPLOYING KNOWLEDGE COLLECTION:**

Click "Deploy to My Repository."
```

DEPLOYMENT:

Target Profile: @DBA Mode: Subscribe (auto-updates enabled)

```
Click "Deploy."

Activate @DBA → Intelligence View → Knowledge Repository:
```

SUBSCRIBED COLLECTIONS: ├— SQL Optimization Best Practices (89 chunks) └— GDPR Compliance Guidelines (47 chunks)

Total available: 136 chunks

```
@DBA profile now has instant access to 136 expert knowledge chunks. Zero learning curve. Immedi

**USING DEPLOYED KNOWLEDGE:**

Let's test it. Ask:
```

```
"What's the best approach for indexing a table with 500 million rows?"

Watch the RAG retrieval:
```

 KNOWLEDGE RETRIEVAL:

Searching knowledge repositories for relevant information…

Matches found:

1. SQL Optimization Best Practices → Chunk #34 (Similarity: 0.91) “For tables exceeding 100 million rows, consider partitioned primary indexes (PPI) rather than non-partitioned. PPIs enable partition elimination, drastically reducing I/O for time-series queries. Choose partition column based on most common query filters…”
2. SQL Optimization Best Practices → Chunk #22 (Similarity: 0.87) “Index design for very large tables requires balancing insert performance with query speed. Consider: (1) Fewer indexes = faster inserts, (2) Covering indexes for common queries, (3) Index compression for space efficiency…”
3. SQL Optimization Best Practices → Chunk #48 (Similarity: 0.82) “Common mistake: Over-indexing large tables. Each index adds overhead to INSERT/UPDATE operations. Aim for 3-5 strategic indexes rather than 15+ reactive indexes…”

Injecting top 3 chunks into context…

```
The LLM receives:
- Your question
- Three expert knowledge chunks (PPI strategy, design tradeoffs, common mistakes)
- No generic advice-actual Teradata-specific expertise

Execution:
```

Based on the SQL Optimization Best Practices knowledge repository:

For a 500M row table, I recommend:

1. **Primary Index Strategy:** Use Partitioned Primary Index (PPI) rather than traditional NUPI. This enables partition elimination for time-series queries, reducing I/O by 90%+.

   Choose partition column based on your most common filter (e.g., SaleDate for sales data).

2. **Secondary Indexes (Limit to 3-5 strategic indexes):**
   - Covering index for most frequent query pattern
   - Index on foreign key columns for join optimization
   - Consider compressed indexes to save space
3. **Avoid Over-Indexing:** Each index adds 15-20% overhead to INSERT operations. With 500M rows, this compounds significantly. Focus on high-impact indexes only.
4. **Statistics Collection:** Run COLLECT STATISTICS on indexed columns monthly. Without stats, optimizer makes poor decisions regardless of index quality.

Would you like me to generate the DDL for a PPI table with recommended indexes?

This isn't generic AI advice. This is expert DBA knowledge, retrieved from the deployed collection, applied to your specific question.

THE KNOWLEDGE MULTIPLICATION:

WITHOUT KNOWLEDGE REPOSITORY: ├─ Junior DBA asks indexing question ├─ AI gives generic advice (“add indexes on queried columns”) ├─ Quality: 5/10 (not wrong, not expert) └─ Implements mediocre solution

WITH KNOWLEDGE REPOSITORY: ├─ Junior DBA asks same question ├─ RAG retrieves expert DBA knowledge chunks ├─ AI gives Teradata-specific, battle-tested advice ├─ Quality: 9/10 (expert-level, first try) └─ Implements optimal solution

ROI OF KNOWLEDGE REPOSITORIES:

TRADITIONAL APPROACH: ├─ Expert DBA: 10 years learning optimization ├─ Knowledge resides in expert’s head ├─ Junior DBAs ask repeatedly (bottleneck) └─ Expert leaves → Knowledge lost

KNOWLEDGE REPOSITORY APPROACH: ├─ Expert documents once (8 hours) ├─ Publishes to marketplace (89 chunks) ├─ 50 users deploy instantly ├─ Every user: 10 years expertise immediately └─ Expert leaves → Knowledge persists

From knowledge locked in experts' heads to knowledge accessible to everyone, instantly, forever

Capture expertise. Chunk and embed. Publish and deploy. Multiply knowledge., instantly, forever

Capture expertise. Chunk and embed. Publish and deploy. Multiply knowledge.

Screen Capture Plan: - [ ] Marketplace → Knowledge Repositories → “Create Collection” button - [ ] Collection setup form: - Name: “Fitness Equipment Product Catalog” - Description (multi-line) - Category: Product Knowledge - Tags: #products #specifications #warranty - Visibility: Public radio selected - [ ] Click “Create Collection” → Empty collection created - [ ] “Upload Documents” button → Upload interface - [ ] Supported formats list (PDF, TXT, DOCX, MD, CSV) - [ ] Drag-and-drop three files: - product_catalog_2024.pdf (2.3 MB, 47 pages) □ - warranty_policies.docx (487 KB, 12 pages) □ - pricing_history_q4.txt (23 KB) □ - [ ] “Process Documents” button - [ ] Chunking strategy options modal: - Fixed Size (1000 chars, 200 overlap) - Semantic Paragraphs (variable) - Hybrid (recommended) ← selected - [ ] “Start Chunking” → Processing status panel - [ ] Processing progress showing: - Document 1: Extracting → Chunking (89 chunks) → Embedding → Complete - Document 2: 23 chunks → Complete - Document 3: 5 chunks → Complete - Summary: 3 docs, 117 chunks, 18 seconds - [ ] “Publish to Marketplace” button - [ ] Publication settings: - Visibility: Public - Tags: (shown) - Collection Purpose (multi-line) - Recommended For (bullet list) - [ ] Click “Publish” → Success confirmation - [ ] Marketplace → Knowledge Repositories → Featured collections - [ ] Collection cards showing: - GDPR Compliance Guidelines (47 chunks, 4.9□, 127 deployments) - SQL Optimization Best Practices (89 chunks, 4.8□, 203 deployments) - Fitness Equipment Product Catalog (117 chunks, NEW, 0 deployments) - [ ] Click “SQL Optimization Best Practices” → Collection preview - [ ] Document sources list (5 files, chunk counts) - [ ] Sample

chunks displayed with text previews - [ ] “Deploy to My Repository” button → Deployment modal - [ ] Target Profile: @DBA, Mode: Subscribe - [ ] Click “Deploy” → Success - [ ] Switch to @DBA profile → Intelligence View → Knowledge Repository tab - [ ] Subscribed collections: - SQL Optimization (89 chunks) - GDPR Compliance (47 chunks) - Total: 136 chunks - [ ] New query: “What’s the best approach for indexing a table with 500M rows?” - [ ] Live Status Panel → Knowledge Retrieval section - [ ] 3 chunk matches displayed: - Chunk #34 (0.91 similarity) - PPI recommendation - Chunk #22 (0.87) - Design tradeoffs - Chunk #48 (0.82) - Over-indexing warning - [ ] AI response using injected knowledge chunks - [ ] Expert-level advice (PPI strategy, 3-5 indexes, statistics) displayed

Supporting Materials: - Knowledge repository architecture documentation - Document upload and processing guide - Chunking strategies technical reference - Embedding model specifications (text-embedding-ada-002) - RAG retrieval algorithm documentation - Knowledge quality best practices - ROI analysis for knowledge repository adoption

---

6.4 Subscribe, Fork, Rate: Community Ecosystem

Duration: 3-4 minutes
Learning Objectives: - Subscribe to collections (reference-based) - Fork collections for customization - Rate and review for quality assurance

Key Topics to Cover: - [ ] Subscribe workflow (no data duplication) - [ ] Fork workflow (independent copy) - [ ] Rating system (1-5 stars) - [ ] Publishing options (Public, Unlisted, Private) - [ ] Community quality assurance

Narration Script:

```
Marketplaces don't work without ecosystems.

Consume content (Subscribe). Customize it (Fork). Ensure quality (Rate).

These three mechanisms transform Uderia's marketplace from "file dump" to "living, quality-assu

**SUBSCRIBE: REFERENCE-BASED DEPLOYMENT**

You discovered a great planner collection: "Customer Analytics Playbook" (45 patterns, 4.9 ).

Click "Deploy to My Repository" → Choose **Subscribe** mode.

What happens?

**Technical Architecture:**
- Uderia creates a REFERENCE in your profile (not a copy)
- Your profile points to the original collection in the author's repository
- When you query, RAG searches both your patterns AND subscribed collections
- No data duplication-single source of truth

**Benefits of Subscribe:**
```

```
  **Auto-Updates:** Author adds new patterns? You get them immediately (no re-deployment)
  **Space Efficiency:** No storage duplication (one 45-pattern collection, infinite subscribers
  **Quality Assurance:** Author improves a pattern? You benefit instantly
  **Author Attribution:** Usage stats flow back to author (recognition for contributions)

**Use Subscribe When:**
- You trust the author to maintain quality
- You want automatic updates
- The collection serves your needs as-is
- You're consuming standard organizational knowledge

**Example Workflow:**

Day 1: You subscribe to "Customer Analytics Playbook" (45 patterns)
Day 15: Author adds "Customer Lifetime Value Prediction" pattern (now 46 patterns)
Day 15 (30 seconds later): You automatically have access to new pattern. Zero action required.

This is live, continuous knowledge sharing.

**FORK: INDEPENDENT COPY**

"Sales Forecasting Templates" (23 patterns, 4.7 ).

Good starting point. Customize for your industry? "Deploy to My Repository" → **Fork**.
```

TECHNICAL ARCHITECTURE: ├— Uderia creates COMPLETE COPY in your repository ├— Your fork is independent (no link to author) ├— Modify, delete, add patterns freely ├— Changes don't affect original └— Original updates don't propagate to fork

BENEFITS: Full Control (modify patterns to fit your needs) Experimentation (test changes without affecting others) Specialization (adapt for industry-specific requirements) Independence (no dependency on author)

USE FORK WHEN: • Need to customize patterns heavily • Adapting for specific industry/use case • Want frozen version (no surprise updates) • Plan to extend significantly

```
Example: Fork (23 patterns) → Delete 5 (fitness irrelevant) → Modify 10 (fitness_db tables) → A

Original unchanged. Your fork tailored perfectly.

**SUBSCRIBE VS FORK DECISION MATRIX:**

| Scenario | Subscribe | Fork |
|----------|-----------|------|
| Standard organizational knowledge |   | |
| Trust author to maintain quality |   | |
| Want automatic updates |   | |
| Need heavy customization | |   |
```

```
| Industry-specific adaptation | |   |
| Want frozen version | |   |
| Experimental changes | |   |
| Minimal storage duplication |   | |

Think of it like software:
- **Subscribe** = Using npm package (linked, updates automatically)
- **Fork** = Copying code into your repo (independent, you control changes)

**RATE & REVIEW: QUALITY ASSURANCE**

You've been using "Customer Analytics Playbook" for two weeks. It's excellent. Time to
rate it.

Go to Marketplace → Find the collection → Click "Rate & Review."

**Rating Interface:**

**Overall Rating:**     (5 stars)

**Category Ratings:**
- **Accuracy:**     (Patterns execute correctly every time)
- **Usefulness:**     (Covers all common customer analysis scenarios)
- **Documentation:**     (Good descriptions, could use more examples)
- **Cost Efficiency:**     (Patterns well-optimized, no wasteful queries)

**Written Review:**
"Deployed this to @ANALYST profile two weeks ago. Has saved me hours of query development.
The 'Customer Churn Risk' pattern alone has been used 34 times with perfect results every
time. Highly recommend for any analytics team."

**Usage Stats (Automatically Included):**
- Deployed to: @ANALYST profile
- Patterns used: 18 of 45 (40% adoption)
- Total executions: 67 times
- Success rate: 98.5% (66 successful, 1 failed due to missing table)
- Avg cost savings: $1.85 per pattern use

Click "Submit Review."

Your review is now visible in the marketplace. Other users can see:
- Your rating (5 )
- Your specific category feedback
- Your usage experience
- Your empirical results (67 executions, 98.5% success)

This isn't subjective opinion. It's data-backed quality assurance.
```

```
**HOW RATINGS AFFECT MARKETPLACE:**

Collection: "Customer Analytics Playbook"
- **Before your review:** 4.8  (67 reviews)
- **After your review:** 4.8  (68 reviews) ← Your 5  maintains high average

**Marketplace Sorting:**

When users browse, they see:
- **Top Rated** (4.5 +): Your review helps maintain this collection's featured status
- **Most Used** (deployments): Your deployment adds to popularity
- **Trending** (recent activity): Your review signals active community engagement

High ratings = More visibility = More deployments = More feedback = Quality reinforcement cycle

**LOW RATINGS = QUALITY SIGNAL:**

Not all collections are great. You try "Database Quick Fixes" (3.2 , 12 reviews).

Result: Patterns fail 40% of the time. Outdated SQL syntax. Poor documentation.

You rate it:
- Overall:     (2 stars)
- Accuracy:     (Multiple patterns failed)
- Usefulness:     (Concepts good, execution poor)
- Documentation:     (Insufficient examples)

Review: "Attempted 5 patterns, 2 failed with SQL syntax errors. Appears unmaintained.
Recommend avoiding until author updates."

This review:
- Warns other users (saves them wasted time)
- Signals to author (feedback for improvement)
- Triggers marketplace moderators (if rating drops below 2.5 , admin review)

Community-driven quality control. Bad collections get flagged. Good collections get promoted.

**VISIBILITY SETTINGS: PUBLISHING OPTIONS**

When you create a collection, you choose visibility:

**Public (Searchable):**
- Appears in marketplace browse/search
- Anyone in organization can discover
- Eligible for "Top Rated" and "Trending" features
- Use for: Broadly useful, production-ready collections

**Unlisted (Link-Only):**
```

- Does NOT appear in marketplace search
- Accessible only via direct link
- You control who can access (share link selectively)
- Use for: Beta testing, team-specific collections, works-in-progress

Private (You Only):
- Completely invisible to others
- Only creator can access
- Not deployable by anyone else
- Use for: Personal experiments, sensitive content, drafts

Workflow Example:

Week 1: Create "Advanced Cohort Analysis" collection → Set to **Private** (testing)
Week 2: Validate patterns with your team → Change to **Unlisted** (share link with team)
Week 3: Team validates quality → Change to **Public** (marketplace launch)
Week 4: Community deploys, rates, reviews → Becomes **Top Rated** (4.9)

Progressive publication. Quality gates at every stage.

TOP-RATED COLLECTIONS: QUALITY SHOWCASE

Go to Marketplace → Filter by "Top Rated" (4.5 +).

You see the organizational hall of fame:

1. **Database Performance Optimization** (4.9 , 203 deployments)
   - 31 patterns for query tuning, index strategies, statistics
   - 98.7% success rate across 4,521 executions
   - Author: @DBA profile (senior DBA team)

2. **Customer Analytics Playbook** (4.8 , 189 deployments)
   - 45 patterns for churn, segmentation, lifetime value
   - 97.2% success rate across 3,847 executions
   - Author: @ANALYST profile (business intelligence team)

3. **GDPR Compliance Knowledge Base** (4.9 , 127 deployments)
   - 47 document chunks covering regulations, requirements, checklists
   - Used in 2,143 queries
   - Author: Legal Department

These collections are battle-tested, community-validated, production-ready. Deploy with confid

THE ECOSYSTEM IN ACTION:

Month 1: 10 users, 5 collections, 23 deployments
Month 3: 50 users, 28 collections, 187 deployments
Month 6: 100 users, 67 collections, 892 deployments

```
**Month 12:** 200 users, 134 collections, 3,456 deployments

**Network Effect:**
- More users → More collections
- More collections → More deployments
- More deployments → More ratings/reviews
- More ratings → Better quality signals
- Better quality → More user trust
- More trust → More contributions

The marketplace becomes self-sustaining, self-improving, self-governing.

**ROI OF COMMUNITY ECOSYSTEM:**

**Without Marketplace:**
- Knowledge siloed in individuals
- Redundant pattern development
- No quality assurance
- Expertise lost when employees leave

**With Marketplace + Subscribe/Fork/Rate:**
- Knowledge shared organization-wide
- Pattern reuse reduces costs by 40%
- Community ratings ensure quality
- Expertise becomes institutional asset

From isolated experts to collective intelligence. From redundant work to efficient reuse.
From unknown quality to community-validated excellence.

Subscribe for convenience. Fork for customization. Rate for quality. Build the ecosystem.
```

Screen Capture Plan: - [ ] Collection detail page with “Deploy to My Repository” button - [ ] Deployment modal showing Subscribe vs Fork options: - Subscribe (radio selected): “Reference-based, auto-updates, space-efficient” - Fork: “Independent copy, full control, no auto-updates” - [ ] Select Subscribe → Click Deploy → Success notification - [ ] Profile Intelligence View showing subscribed collection (reference icon) - [ ] Subscribe vs Fork decision matrix table visualization - [ ] Two weeks later: Author adds new pattern - [ ] Your profile automatically shows new pattern count (45→46) - [ ] Same collection, now choose Fork - [ ] Fork creation showing: - “Creating independent copy…” - Copy complete: 23 patterns cloned - “You can now modify freely” - [ ] Forked collection in your repository (edit mode available) - [ ] Modify pattern example (delete 2, edit 3, add 1) - [ ] Result: Your customized version (22 patterns, modified) - [ ] “Rate & Review” interface: - Overall rating: 5 star selector - Category ratings: Accuracy, Usefulness, Documentation, Cost Efficiency - Written review text box - Usage stats auto-populated (67 executions, 98.5% success) - [ ] Submit review → Success confirmation - [ ] Collection page updated showing your review - [ ] Rating distribution: 68 reviews, 4.8□ average - [ ] Low-rated collection example (3.2□) - [ ] Poor review with specific failures documented - [ ] Warning badge: “Below 3.5□ - Use with caution” - [ ] Visibility settings interface: - Public radio (“Searchable, discoverable”) - Unlisted radio (“Link-only access”) - Private radio (“You only”) - [ ] Progressive publication workflow diagram: - Week

1: Private (testing) - Week 2: Unlisted (team validation) - Week 3: Public (marketplace launch) - Week 4: Top Rated (community validation) - [ ] Marketplace filtered by “Top Rated” - [ ] Three featured collections with 4.8-4.9□ ratings - [ ] Collection stats: Deployments, executions, success rates - [ ] Ecosystem growth chart: - X-axis: Months 1-12 - Y-axis: Collections, deployments, users - Exponential growth curves - [ ] Network effect visualization diagram - [ ] ROI comparison: Siloed vs Marketplace ecosystem

Supporting Materials: - Subscribe vs Fork technical architecture documentation - Reference-based deployment specifications - Rating system algorithm and quality scoring - Community moderation guidelines - Visibility and publication workflow guide - Network effects analysis and ROI projections - Marketplace governance policies

---

Module 7: FINANCIAL GOVERNANCE

Goal: Complete transparency and control over LLM spending

7.1 Consumption Profiles & Rate Limiting

Duration: 3-4 minutes
Learning Objectives: - Understand consumption profile tiers - Configure rate limits and quotas - Monitor usage enforcement

Key Topics to Cover: - [ ] Four profile tiers (Free, Pro, Enterprise, Unlimited) - [ ] Per-user rate limits (hourly/daily prompts) - [ ] Monthly token quotas - [ ] Configuration change limits - [ ] Admin bypass capabilities - [ ] Real-time enforcement

Narration Script:

```
LLM costs spiral out of control fast.

One user runs 500 queries in a day experimenting. Another accidentally creates an infinite loo

Without governance, your LLM bill becomes unpredictable.

Uderia solves this with Consumption Profiles-fine-grained rate limiting and quota management a

**THE FOUR CONSUMPTION TIERS:**

Admin → User Management → Consumption Profiles. Four tiers:
```

TIER 1: FREE (Evaluation Users) ├— Hourly: 10 prompts ├— Daily: 50 prompts ├— Monthly: 100K tokens (~$0.15 ceiling) ├— Config Changes: 3/day └— Use: Trial users, demos, proof-of-concept

TIER 2: PRO (Individual Contributors) ├— Hourly: 30 prompts ├— Daily: 200 prompts ├— Monthly: 1M tokens (~$1.50 ceiling) ├— Config Changes: 10/day └— Use: Analysts, developers, standard users

TIER 3: ENTERPRISE (Power Users) ├— Hourly: 100 prompts ├— Daily: 1,000 prompts ├— Monthly: 10M tokens (~$15.00 ceiling) ├— Config Changes: 50/day └— Use: Data scientists, DBAs, automation

TIER 4: UNLIMITED (Administrators & Production) ├— Hourly: None ├— Daily: None ├— Monthly: None ├— Config Changes: Unlimited └— Use: Admins, CI/CD, production API

Each tier balances access with cost control. Most start at PRO. Power users → ENTERPRISE. Admi

ASSIGNING CONSUMPTION PROFILES:
ASSIGNING CONSUMPTION PROFILES:

Click user: alice@company.com

USER DETAILS:

Name: Alice Johnson Role: User (standard permissions) Current Profile: PRO (1M tokens/month)

Usage This Month: ├— Tokens: 347,521 / 1,000,000 (34.8%) ├— Prompts Today: 23 / 200 (11.5%) └— Status: Within limits

Alice operating normally. Needs more capacity for special project?

"Edit Profile" → Change to **ENTERPRISE** → Save.

Alice now has 10M tokens/month (10x increase), 1,000 prompts/day, 100 prompts/hour.

RATE LIMIT ENFORCEMENT IN ACTION:

Test user: bob@company.com with **FREE** profile (10/hour, 50/day).

Bob starts querying:

QUERY SEQUENCE:

Prompt 1: “Show me Q4 sales by region” (1/10) Prompt 2: “Break down by product category” (2/10) Prompt 3: “Add customer segmentation” (3/10) … Prompt 10: “Generate executive summary” (10/10) Prompt 11: “Create visualization chart” RATE LIMITED

Bob hits hourly limit. Uderia responds:

 RATE LIMIT EXCEEDED

You have reached your hourly limit (10 prompts).

Profile: FREE | Next reset: 42 minutes

To increase limits, contact admin or upgrade to PRO (200 prompts/day, 1M tokens/month).

Bob can't query for 42 minutes. System enforces automatically. No manual policing.r 42 minutes policing required.

QUOTA EXHAUSTION:

alice@company.com (PRO profile, 1M tokens/month) at month-end:

USAGE STATS: Tokens Used: 987,432 / 1,000,000 (98.7%) Remaining: 12,568 tokens (~19 prompts left)

Alice runs complex query:

QUERY VALIDATION:

Query: “Analyze 500K customer records, identify churn patterns, generate report”

Estimated: 23,450 tokens Available: 12,568 tokens Result: INSUFFICIENT QUOTA

Uderia blocks:

 MONTHLY QUOTA EXCEEDED

This query requires 23,450 tokens. You have 12,568 tokens remaining.

Profile: PRO (1M tokens/month) Quota resets: December 1, 2025 (3 days)

Options: 1. Wait for reset (3 days) 2. Contact admin for quota increase 3. Simplify query to fit remaining tokens

Alice can't run expensive queries. Waits for reset or requests upgrade.

Financial governance in action-preventing runaway costs before they happen.

CONFIGURATION CHANGE LIMITS:

Subtle but important control. FREE profile: 3 configuration changes/day. Why?

SCENARIO WITHOUT LIMITS: ├─ User creates 10 profiles (different LLM providers) ├─ Switches between profiles 100 times/day ├─ Each switch triggers LLM calls for initialization └─ Result: Excessive overhead, wasted tokens

Configuration limit prevents abuse. Bob (FREE) tries:

PROFILE SWITCHES:

Change 1: @ANALYST → @DBA (1/3) Change 2: @DBA → @QUALITY (2/3) Change 3: @QUALITY → @LOCAL (3/3) Change 4: @LOCAL → @PROD BLOCKED

Limit reached:

 CONFIGURATION CHANGE LIMIT REACHED

You have made 3 profile switches today (FREE tier limit).

Next reset: 6 hours

Upgrade to PRO (10 changes/day) or ENTERPRISE (50 changes/day).

Can't profile-hop excessively. Prevents abuse and accidental overhead.
ADMIN BYPASS & MONITORING:

Admins have UNLIMITED profile by default:

ADMIN CAPABILITIES: • No rate limits (query 1000 times/hour) • No quota restrictions (unlimited tokens) • No configuration limits (switch profiles freely)

WHY: • Troubleshooting access (testing for users) • Maintenance operations (bulk queries, validation) • Emergency overrides (time-sensitive analysis)

Admins monitor global usage. Admin → Consumption Analytics:

GLOBAL USAGE SUMMARY:

Total Users: 127 Active This Month: 89 (70%) Total Tokens: 34.2M tokens Total Cost: $51.30 Avg Cost Per Active User: $0.58

USERS NEAR QUOTA (Top 5): 1. alice@company.com (PRO): 987K/1M (98.7%) 2. sarah@company.com (ENTERPRISE): 9.2M/10M (92%) 3. mike@company.com (PRO): 856K/1M (85.6%)

RATE LIMIT VIOLATIONS (Last 24h): 1. bob@company.com (FREE): 12 violations 2. carol@company.com (PRO): 3 violations

Admins can:
EMERGENCY QUOTA OVERRIDE:

Alice hits monthly quota but has urgent Q4 board presentation tomorrow.

Admin action:

EMERGENCY QUOTA GRANT:

1. User Management → alice@company.com
2. Click “Grant Emergency Quota”
3. Add: 500,000 tokens (temporary extension)
4. Expiration: 48 hours
5. Save

Alice now has:

QUOTA BREAKDOWN: ├— Base Quota: 1M tokens (exhausted) ├— Emergency Extension: +500K tokens (granted) └— Total Available: 500K tokens for 48 hours

After 48 hours → Emergency quota expires. Monthly reset → Replenishes base quota.

Controlled flexibility-governance with escape hatches for legitimate needs.

ROI OF CONSUMPTION PROFILES:

ORGANIZATION: 100 users, NO GOVERNANCE

10 power users × $50/month = $500 30 moderate users × $15/month = $450 60 light users × $3/month = $180 ——————————————————————— Total: $1,130/month

Issue: No visibility, no control, cost creep

ORGANIZATION: 100 users, WITH PROFILES

10 power users → ENTERPRISE ($15 cap) = $150 30 moderate users → PRO ($1.50 cap) = $45 60 light users → FREE ($0.15 cap) = $9 ——————————————————————— Total: $204/month

Savings: $926/month (82% reduction)

Consumption profiles don't just track costs. They eliminate runaway spending.

From "unpredictable bills" to "controlled, predictable LLM budgets."

From "after-the-fact shock" to "proactive governance."

Rate limits. Quotas. Visibility. Control. Financial governance.

Screen Capture Plan: - [ ] Admin → User Management → Consumption Profiles - [ ] Four tier cards displayed: - FREE (10/hr, 50/day, 100K tokens/month) - PRO (30/hr, 200/day, 1M tokens/month) - ENTERPRISE (100/hr, 1000/day, 10M tokens/month) - UNLIMITED (no limits) - [ ] User details for alice@company.com: - Current: PRO profile - Usage: 347,521/1,000,000 tokens (34.8%) - Prompts today: 23/200 - Status: □ Within limits - [ ] Edit profile → Change to ENTERPRISE → Save - [ ] Confirmation: Limits updated instantly - [ ] Test user: bob@company.com (FREE profile) - [ ] Query sequence showing rate limit hit: - Prompts 1-10: □ Success - Prompt 11: □ Rate limited - [ ] Rate limit error message: - "You have reached your hourly limit (10 prompts)" - "Next reset: 42 minutes" - Upgrade suggestion - [ ] alice@company.com month-end: - Usage: 987,432/1,000,000 tokens (98.7%) - Remaining: 12,568 tokens - [ ] Complex query attempt: - Required: 23,450 tokens - Available: 12,568 tokens - Result: □ Insufficient quota - [ ] Quota exceeded error with options: - Wait for reset (3 days) - Contact admin - Simplify query - [ ] Configuration change limit demonstration: - bob (FREE): 3 switches □, 4th switch □ blocked - Error: "Configuration change limit reached" - [ ] Admin → Consumption Analytics dashboard: - Global usage: 127 users, 34.2M tokens, $51.30 - Users near quota (top 5 list) - Rate limit violations (last 24h) - [ ] Emergency quota override interface: - alice@company.com selected - "Grant Emergency Quota" button - Add 500K tokens, 48hr expiration - Save → Emergency extension granted - [ ] ROI comparison chart: - No governance: $1,130/month - With profiles: $204/month - Savings: $926/month (82%)

Supporting Materials: - Consumption profiles documentation - Rate limiting technical specifications - Quota management guide - Admin override procedures - Cost control best practices - ROI calculator for consumption governance

---

7.2 Cost Analytics & Optimization

Duration: 4-5 minutes
Learning Objectives: - Analyze cost distribution by provider - Identify most expensive models and queries - Explore cost trends and optimization opportunities

Key Topics to Cover: - [ ] Cost distribution by provider chart - [ ] Top 5 most expensive models - [ ] 30-day cost trend visualization - [ ] Most expensive sessions/queries - [ ] RAG efficiency savings metrics - [ ] Optimization recommendations

Narration Script:

```
Consumption profiles control spending. But how do you optimize what you're already spending?

Where are inefficiencies? Which models cost most? Which queries need redesign?

Uderia's Cost Analytics dashboard answers with forensic precision.

**COST DISTRIBUTION BY PROVIDER:**

Admin → Cost Analytics → Provider Breakdown. Pie chart:
```

PROVIDER COST DISTRIBUTION (Last 30 Days):

Anthropic (Claude): $127.45 (48.2%) — 342,891 tokens Google (Gemini): $78.32 (29.6%) — 5,234,782 tokens OpenAI (GPT-4): $43.21 (16.3%) — 87,432 tokens AWS Bedrock: $12.18 (4.6%) — 102,341 tokens Ollama (Local): $3.45 (1.3%) — Infrastructure only ——————————————————————————————— Total: $264.61 for 5.77M tokens

```
Story:

**Anthropic dominates cost (48%)** - Claude Sonnet heavily used. Only 342K tokens = high per-to

**Google delivers volume (5.2M tokens)** - Low cost (29.6% spend). Gemini Flash = workhorse fo

**OpenAI expensive per token** - $43 for 87K tokens. GPT-4 costs $0.49/execution average. Use s

**Ollama nearly free** - $3.45 infrastructure, zero API cost. Only 102K tokens. Opportunity: SL

Breakdown reveals optimization opportunities immediately.

**TOP 5 MOST EXPENSIVE MODELS:**

Click "Model Breakdown" tab. Ranked by total cost:
```

1. Claude Sonnet 3.5 (Anthropic) Cost: $89.34 | Tokens: 241,203 | Avg: $0.37/query Usage: 243 queries | Profile: @DBA, @QUALITY
2. Gemini 1.5 Flash (Google) Cost: $78.32 | Tokens: 5,234,782 | Avg: $0.015/query Usage: 5,221 queries | Profile: @ANALYST, @COST

3. GPT-4o (OpenAI) Cost: $43.21 | Tokens: 87,432 | Avg: $0.49/query Usage: 88 queries | Profile: @PROD (rarely)
4. Claude Opus 3 (Anthropic) Cost: $38.11 | Tokens: 101,688 | Avg: $0.95/query Usage: 40 queries | Profile: Custom (special projects)
5. Claude Sonnet 3.5 v2 (AWS Bedrock) Cost: $12.18 | Tokens: 102,341 | Avg: $0.12/query Usage: 101 queries | Profile: @PROD (Bedrock routing)

Key Insight: Claude Opus ($0.95/query) + GPT-4o ($0.49/query) = only 128 queries. But repre

Optimization: Reserve Opus/GPT-4 for complex reasoning. Use Sonnet or Gemini otherwise. Potenti

30-DAY COST TREND:

Click "Trend Analysis" tab. Line chart:

DAILY COST (Last 30 Days):

Nov 7: $4.23 Nov 8: $6.45 Nov 9: $5.87 … Nov 20: $12.34 ← Spike! Nov 21: $11.89 Nov 22: $14.56 ← Another spike! … Dec 5: $7.23 Dec 6: $8.91 Dec 7: $9.12 (today) ——————————————————————— Avg Daily: $8.82 Projected Monthly: $264.60

Spikes on Nov 20-22 ($12-14/day vs $8 average). Click Nov 20 spike:

HIGH-COST SESSIONS (Nov 20):

1. sarah@company.com | Q4_Board_Prep Cost: $4.87 | Queries: 23
2. mike@company.com | Customer_Analysis Cost: $3.21 | Queries: 47
3. alice@company.com | Performance_Tuning Cost: $2.45 | Queries: 12

Sarah: 23 expensive queries for Q4 board presentation. Legitimate use case, one-time event.

Mike: 47 queries on customer analysis. Might be inefficient-exploring without clear plan.

Optimization Opportunity: Provide Mike with pre-built planner templates for customer analys

MOST EXPENSIVE SESSIONS:

Click "Expensive Sessions" tab. Sorted by cost:

TOP 10 COSTLIEST SESSIONS (Last 30 Days):

1. Q4_Board_Presentation | sarah@company.com Cost: $18.45 | Queries: 67 | Avg: $0.28/query Models: Opus (15), Sonnet (52) □ High-stakes analysis, appropriate selection
2. Database_Migration_Planning | mike@company.com Cost: $14.87 | Queries: 31 | Avg: $0.48/query Models: GPT-4o (23), Opus (8) □ Complex reasoning justified premium models

3. Customer_Churn_Exploration | alice@company.com Cost: $11.23 | Queries: 89 | Avg: $0.13/query Models: Sonnet (67), Flash (22) Good mix, but 89 queries exploratory → Deploy “Customer Insights Playbook”
4. Product_Performance_Deep_Dive | bob@company.com Cost: $9.87 | Queries: 156 | Avg: $0.06/query Models: Flash (156) Cost-efficient model, but high query count → Consider batch processing or planners

```
Patterns:
- High-stakes sessions (board prep, migrations) justify premium models
- Exploratory sessions (churn, product) would benefit from planners
- High query counts signal inefficiency-users iterating without guidance

**RAG EFFICIENCY SAVINGS:**

Where it gets interesting. Click "RAG Impact" tab.

Uderia tracks two scenarios per query:
1. **Actual cost** (with RAG champion case injection)
2. **Estimated cost without RAG** (cold start, trial-and-error)
```

RAG SAVINGS ANALYSIS (Last 30 Days):

Total Queries with RAG Boost: 1,247 queries Average Match Quality: 0.78 (good)

COST BREAKDOWN: Actual (with RAG): $87.34 Estimated (without RAG): $143.21 ——————————————— Savings: $55.87 (39% reduction)

TOP RAG CONTRIBUTORS: 1. Customer Insights Playbook $18.45 saved (267 pattern matches) 2. Database Performance Optimization $14.23 saved (189 pattern matches) 3. Sales Analytics Accelerators $11.34 saved (312 pattern matches) 4. SQL Best Practices (Knowledge) $8.76 saved (143 knowledge retrievals)

```
Planner repositories saved $55.87 this month. That's 39% of what you would have spent without t

Remember Module 6.2? We said planners deliver 36% savings. This proves it-empirically.

**ROI:** Spent 8 hours creating these collections. They save $55/month. Payback period: less tl

**OPTIMIZATION RECOMMENDATIONS:**

Uderia generates automated insights. Click "Optimization Insights":
```

 COST OPTIMIZATION RECOMMENDATIONS:

1. Shift Gemini Workload to Ollama (Local) Current: 5.2M tokens on Gemini Flash ($78.32) Opportunity: 60% are simple data retrieval Action: Configure @LOCAL profile for basic queries Potential Savings: $47/month
2. Reserve Claude Opus for Complex Reasoning Only Current: 40 queries on Opus ($38.11, avg $0.95/query) Observation: 18 queries similar to existing Sonnet patterns Action: Use

Sonnet ($0.37) for 18, Opus for remaining 22 Potential Savings: $10.44/month

3. Deploy Missing Planner Collections Current: Mike ran 89 churn queries exploratorily ($11.23) Opportunity: “Customer Insights Playbook” exists but not deployed Action: Deploy collection to Mike’s @ANALYST profile Potential Savings: $4.50/month (40% reduction)
4. Cross-Tier Knowledge Transfer (Module 4.2 Concept) Current: Using Opus ($0.95/query) for complex analysis Opportunity: Train RAG on Opus, execute on Flash ($0.015/query) Example: Segmentation trained on Opus (5 queries, $4.75) then executed on Flash (95 queries, $1.43) vs all Opus (100 queries, $95) Potential Savings: $88.82 per 100-query workflow (93% reduction)
5. Batch Processing for High-Volume Users Current: Bob ran 156 queries in one session ($9.87) Opportunity: Many queries similar, batch into multi-part request Action: Recommend TDA_LLMTask for batch analysis Potential Savings: $6/month (156→40 batch requests)

——————————————————————————————————— Total Potential Savings: $156.76/month (59% reduction)

```
Not generic tips. Data-driven, user-specific, action-oriented recommendations.

Admins click "Apply Recommendation" to:
- Auto-deploy collections to specific users
- Adjust profile configurations
- Send optimization suggestions to users

From "black box spending" to "forensic cost visibility and actionable optimization."

From "after-the-fact reports" to "real-time savings opportunities."

Cost analytics. Pattern detection. RAG ROI tracking. Automated recommendations. Financial intel
```

Screen Capture Plan: - [ ] Admin → Cost Analytics → Provider Breakdown - [ ] Pie chart showing: - Anthropic: $127.45 (48.2%) - Google: $78.32 (29.6%) - OpenAI: $43.21 (16.3%) - AWS Bedrock: $12.18 (4.6%) - Ollama: $3.45 (1.3%) - Total: $264.61 - [ ] Model Breakdown tab → Top 5 models table: - Claude Sonnet 3.5: $89.34, 241K tokens, $0.37/query, 243 queries - Gemini Flash: $78.32, 5.2M tokens, $0.015/query, 5,221 queries - GPT-4o: $43.21, 87K tokens, $0.49/query, 88 queries - Claude Opus: $38.11, 101K tokens, $0.95/query, 40 queries - Bedrock Sonnet: $12.18, 102K tokens, $0.12/query, 101 queries - [ ] Key insight callout: Opus+GPT-4 = 128 queries but 30.6% of spend - [ ] Trend Analysis tab → 30-day line chart - [ ] Daily costs plotted with spikes on Nov 20-22 - [ ] Click Nov 20 → Drill-down showing 3 high-cost sessions - [ ] Expensive Sessions tab → Top 10 table: - Q4_Board_Presentation: $18.45, 67 queries, sarah - Database_Migration: $14.87, 31 queries, mike - Customer_Churn: $11.23, 89 queries, alice (with recommendation) - Product_Performance: $9.87, 156 queries, bob (with recommendation) - [ ] RAG Impact tab → Savings analysis: - 1,247 queries with RAG boost - Actual: $87.34, Without RAG: $143.21 - Savings: $55.87 (39%) - [ ] Top RAG contributors list: - Customer Insights: $18.45 saved - DB Performance: $14.23 saved - Sales Analytics: $11.34 saved - [ ] Optimization Insights panel → 5 recommendations: - Shift to Ollama: $47/month savings - Reserve Opus: $10.44/month savings - Deploy collections: $4.50/month savings - Cross-tier transfer: $88.82 savings (example) - Batch processing: $6/month savings - Total potential: $156.76/month (59% reduction) - [ ] “Apply

Recommendation" buttons next to each insight

Supporting Materials: - Cost analytics technical documentation - Provider pricing comparison reference - Model selection optimization guide - RAG savings calculation methodology - Cross-tier knowledge transfer case studies - Batch processing best practices

---

7.3 Budget Alerts & Forecasting

Duration: 3-4 minutes
Learning Objectives: - Configure budget thresholds and alerts - Monitor spending against forecasts - Receive proactive cost warnings

Key Topics to Cover: - [ ] Monthly budget threshold configuration - [ ] Real-time spending tracking - [ ] Automated alert triggers (50%, 75%, 90%, 100%) - [ ] Cost forecasting based on usage trends - [ ] Email/Slack notifications - [ ] Budget enforcement options

Narration Script:

Cost analytics show where money went. But what about preventing overspending before it happens'

Budget Alerts and Forecasting-proactive cost management.

SETTING MONTHLY BUDGET THRESHOLDS:

Admin → Financial Governance → Budget Configuration:

ORGANIZATION BUDGET SETTINGS:

Monthly Budget: $500.00 (Approved LLM spending limit)

ALERT THRESHOLDS: ├─ □ 50% Warning ($250) → Email to admins ├─ □ 75% Warning ($375) → Email to admins + finance ├─ □ 90% Critical ($450) → Email + Slack + restrict new users └─ □ 100% Exhausted ($500) → Email + Slack + BLOCK non-admin queries

ENFORCEMENT MODE: • Soft Cap (Alerts only, no blocking) □ Selected • Hard Cap (Block queries when exhausted)

Soft Cap-admins get warnings, users aren't blocked. For tighter control, switch to Hard Cap.

Click "Save Budget Settings."

REAL-TIME SPENDING TRACKER:

Budget Dashboard (same page, scroll down):

□ BUDGET STATUS (December 2025):

Spent: $264.61  (52.9%) Remaining: $235.39 Days Left: 24 days

PACE ANALYSIS: Days Elapsed: 7 of 31 (22.6%) Budget Used: 52.9% Status: □ SPENDING FASTER THAN PACE

Daily Average: $37.80/day Budget Allows: $16.13/day (to stay within $500) Overage: +$21.67/day (135% over pace)

```
Warning signal. 7 days into December, already used 53% of budget. At current pace, you'll hit $
18 days early.

System detected automatically. No manual calculation needed.

**ALERT TRIGGER HISTORY:**

Scroll to "Alert History":
```

ALERT LOG (December 2025):

Dec 1, 08:00 AM - Budget Reset Budget: $500.00 | Spend: $0.00

Dec 4, 02:34 PM - □ 50% WARNING TRIGGERED Spent: $252.18 (50.4%) Recipients: admin@company.com, finance@company.com Action: Email sent

Dec 6, 11:47 AM - □ 75% WARNING TRIGGERED Spent: $377.45 (75.5%) Recipients: admin, finance, ceo Action: Email + Slack notification Message: “LLM budget at 75%. Projected overage by Dec 13. Review cost analytics.”

Dec 7 (Today), 09:12 AM - □ 90% CRITICAL (PENDING) Current: $264.61 (52.9%) Forecast: Will trigger Dec 9 at current pace Action: Pre-alert sent to admins

```
Already triggered 50% and 75% warnings. 90% critical alert forecasted for Dec 9 if spending co

**COST FORECASTING:**

Click "View Forecast" tab. Uderia projects spend based on:
1. Historical usage (last 30 days)
2. Current month trajectory (first 7 days)
3. Known upcoming events (user activity patterns)
```

FORECAST MODEL: Linear Projection

Current Spend (Dec 7): $264.61 Daily Average (7 days): $37.80/day Remaining Days: 24 days

PROJECTED SPEND: Dec 31 Forecast: $264.61 + (24 × $37.80) = $1,171.81

Budget: $500.00 Projected Overage: $671.81 (234% over budget)

□ BUDGET BREACH FORECAST: December 13, 2025 (6 days)

```
Alarming. Without intervention, you'll exhaust budget in 6 days and overspend by $671 for the m

But Uderia also shows alternate scenarios:
```

SCENARIO ANALYSIS:

1. Current Pace (No Changes) Dec 31 Total: $1,171.81 Overage: $671.81 Likelihood: 85%

2. Apply Optimization Recommendations (Module 7.2) Shift 60% to Ollama: -$47/month Reserve Opus: -$10/month Deploy planners: -$4.50/month Adjusted Daily: $24.75/day (35% reduction) Dec 31 Total: $264.61 + (24 × $24.75) = $858.61 Overage: $358.61 Likelihood: 60%
3. Aggressive Cost Controls All optimizations: -$61.50/month Shift 50% to cheaper models: -$80/month Batch processing: -$20/month Adjusted Daily: $14.20/day (62% reduction) Dec 31 Total: $264.61 + (24 × $14.20) = $605.41 Overage: $105.41 Likelihood: 40% (requires behavior changes)
4. Budget Increase Raise to $1,200/month No behavior changes needed Dec 31 Total: $1,171.81 Overage: $0 (within new budget) Likelihood: 100% (if finance approves)

```
Admins choose:
- **Optimize** (Scenario 2 or 3)
- **Increase Budget** (Scenario 4)
- **Enforce Hard Cap** (block queries at $500)

**EMAIL ALERT EXAMPLE:**

What admins received when 75% warning triggered (Dec 6):
```

From: uderia-alerts@company.com To: admin@company.com, finance@company.com, ceo@company.com Subject: LLM Budget Alert: 75% Threshold Reached

---

Hi Team,

Your organization has reached 75% of the monthly LLM budget:

 BUDGET STATUS: Monthly Budget: $500.00 Spent to Date: $377.45 (75.5%) Remaining: $122.55 (24.5%) Days Remaining: 25 days

 PACE WARNING: Spending 3.2x faster than budget allows Budget exhausted by: December 13 (19 days early) Projected month-end: $1,171.81 (234% over budget)

 RECOMMENDED ACTIONS: 1. Review Cost Analytics Dashboard 2. Deploy planner repositories to high-usage users (potential $61/month savings) 3. Configure consumption profiles to limit high-cost patterns 4. Consider budget increase if justified by business value

 QUICK LINKS: Cost Analytics Dashboard Optimization Recommendations Budget Configuration

Next Alert: 90% threshold ($450) - Forecasted Dec 9

Uderia Financial Governance System

```
Actionable email:
- Clear status (75.5% spent)
- Specific warning (exhausted Dec 13)
- Concrete recommendations (deploy planners, configure profiles)
- Direct links (one-click access)
```

Admins don't guess. They act on data.

SLACK INTEGRATION:

Uderia also sends Slack notifications (if configured). In your #finance-alerts channel:

 Uderia Budget Alert

75% of monthly LLM budget reached ($377.45 / $500.00)

Projected overage: $671.81 by month-end Forecast: Budget exhausted by Dec 13 at current pace

[View Dashboard] [Optimization Tips] [Adjust Budget]

Real-time visibility. Instant awareness. Immediate action.

HARD CAP ENFORCEMENT:

Let's say you decide to enforce a Hard Cap. Go back to Budget Configuration.
HARD CAP ENFORCEMENT:

Decide to enforce Hard Cap. Back to Budget Configuration:

ENFORCEMENT MODE: • Soft Cap (Alerts only) ← Current • Hard Cap (Block queries when exhausted) Select

Save. Simulate reaching 100% budget:

Dec 13, 10:23 AM - 100% BUDGET EXHAUSTED

Spent: $500.00 (100%) | Remaining: $0.00

ENFORCEMENT: HARD CAP ACTIVE

ACTIONS TAKEN: ├─ All non-admin queries BLOCKED ├─ Email sent to admin, finance, ceo ├─ Slack notification to #finance-alerts └─ User-facing message updated:

“ Monthly LLM budget exhausted. Organization reached $500/month limit. Contact administrator to increase budget or wait for reset (Jan 1). Admins can query for emergency operations.”

Users see when they try to query:

 BUDGET LIMIT REACHED

Your organization has exhausted its monthly LLM budget ($500.00).

Next reset: January 1, 2025 (19 days)

Contact administrator if you need urgent access.

Hard Cap protects against runaway costs. No overspend. Absolute control.
But what if CEO needs urgent analysis despite budget exhaustion?

ADMIN EMERGENCY OVERRIDE:

CEO needs urgent analysis despite budget exhaustion? Admin action:

EMERGENCY EXTENSION:

1. Budget Configuration → "Grant Emergency Extension"
2. Amount: $200 (temporary increase)
3. Expiration: 48 hours
4. Reason: "Q4 board presentation - CEO urgent request"
5. Save

Budget temporarily increases to $700. CEO can query. After 48 hours → Extension expires.

Controlled flexibility-governance with emergency escape hatches.

SCENARIO: NO BUDGET ALERTS

December spend: $1,171.81 (uncontrolled) January surprise: Finance reviews bill, shocked Reaction: "Why did we spend $1,200 on LLMs?!" Result: Post-facto investigation, no prevention

SCENARIO: WITH BUDGET ALERTS & FORECASTING

Dec 4: 50% warning → Admin reviews Dec 6: 75% warning → Optimization deployed Dec 7-9: Planner repositories deployed, users educated Result: Daily spend drops to $22/day (42% reduction) Dec 31 Final: $733 spent (vs $1,171 without action)

Savings: $438 (37% reduction from uncontrolled spending)

Proactive alerts prevented $438 in overspend. That's 87% of the original $500 budget.

From "surprised by bills" to "controlling costs in real-time."

From "reactive panic" to "proactive governance."

Budget thresholds. Real-time tracking. Forecasting. Automated alerts. Emergency controls. Finar

Screen Capture Plan: - [ ] Admin → Financial Governance → Budget Configuration - [ ] Monthly budget: $500.00 - [ ] Alert thresholds: - 50%: $250 (Email to admins) - 75%: $375 (Email to admins + finance) - 90%: $450 (Email + Slack + restrict new users) - 100%: $500 (Email + Slack + BLOCK queries) - [ ] Enforcement mode: - Soft Cap (alerts only) ← selected - Hard Cap (block when exhausted) - [ ] Budget Dashboard showing: - Spent: $264.61 (52.9% of budget) - Remaining: $235.39 - Progress bar visualization - Pace analysis: Spending 135% faster than pace - Daily average: $37.80 vs allowed $16.13 - [ ] Alert History log: - Dec 1: Budget reset - Dec 4: 50% warning triggered - Dec 6: 75% warning triggered - Dec 9: 90% forecast - [ ] Forecast tab → Projections: - Current pace: $1,171.81 (234% over) - Budget breach: Dec 13 (6 days) - [ ] Scenario analysis table: - Scenario 1 (no changes): $1,171.81 overage - Scenario 2 (optimizations): $858.61 - Scenario 3 (aggressive): $605.41 - Scenario 4 (increase budget): $1,200 new budget - [ ] Email alert example (75% warning): - Subject line - Budget status summary - Pace warning

with specific dates - Recommended actions (numbered list) - Quick links to dashboards - [ ] Slack notification in #finance-alerts: - 75% alert message - Projected overage - Action buttons - [ ] Hard Cap enforcement demonstration: - Change to Hard Cap mode - 100% budget reached simulation - User-facing block message - Admin notification - [ ] Emergency extension interface: - Grant Emergency Extension form - Amount: $200 - Expiration: 48 hours - Reason field - Save → Temporary budget increase to $700 - [ ] ROI comparison: - No alerts: $1,171.81 spent - With alerts: $733 spent - Savings: $438 (37% reduction)

Supporting Materials: - Budget configuration guide - Alert threshold best practices - Forecasting methodology documentation - Hard Cap vs Soft Cap comparison - Email/Slack integration setup guide - Emergency override procedures - Cost governance case studies

---

Module 8: Administration

Goal: Complete platform administration and governance capabilities

8.1 User & Access Management

Duration: 3-4 minutes
Learning Objectives: - Manage user accounts - Assign roles and permissions - Monitor user activity

Key Topics to Cover: - [ ] User administration panel - [ ] Creating/editing users - [ ] Role assignment (User, Developer, Admin) - [ ] Consumption profile assignment - [ ] User activity monitoring - [ ] Account suspension/activation

Narration Script:

```
Multi-user platforms need governance. Who can access what? Who's consuming resources?

Uderia's User & Access Management provides complete administrative control.

**USER ADMINISTRATION PANEL:**

Admin → User Management. Table of all users:
```

ACTIVE USERS (127 total):

Name Email Role Profile Status Last Active Cost ———
Alice Johnson alice@company.com User PRO Active 2 hrs ago $34.56 Bob Smith bob@company.com User FREE Active 1 day ago $2.18 Sarah Davis sarah@company.com Developer ENTERPRISE Active 30 min ago $67.89 Mike Chen mike@company.com Admin UNLIMITED Active 5 min ago $12.34 Carol Williams carol@company.com User PRO Suspended 14 days ago $0.00

```
At a glance:
- 127 users across three roles (User, Developer, Admin)
- Four consumption profiles (FREE, PRO, ENTERPRISE, UNLIMITED)
- Activity status (active vs dormant)
- Cost accountability (per-user spend)
```

Complete visibility into your user base.

CREATING A NEW USER:

Click "Add User." User creation form:

NEW USER DETAILS:

BASIC INFORMATION: ├─ Username: john.doe ├─ Email: john.doe@company.com ├─ Full Name: John Doe └─ Department: Analytics (optional)

PASSWORD: ├─ Initial Password: [Auto-generated: xK9$mP2#vL7q] └─ □ Force password change on first login

ROLE ASSIGNMENT: ├─ □ User (Standard permissions) □ Selected ├─ □ Developer (Advanced tools + debugging) └─ □ Admin (Full system control)

CONSUMPTION PROFILE: ├─ □ FREE (10/hr, 50/day, 100K tokens/month) ├─ □ PRO (30/hr, 200/day, 1M tokens/month) □ Selected ├─ □ ENTERPRISE (100/hr, 1000/day, 10M tokens/month) └─ □ UNLIMITED (No limits)

DEFAULT PROFILE DEPLOYMENT: ├─ □ @ANALYST (Gemini Flash + Analytics tools) ├─ □ @DBA (if John needs database access) └─ □ @QUALITY (not needed for analysts)

Click "Create User." Uderia creates:
1. User account with credentials
2. PRO consumption profile (1M tokens/month)
3. @ANALYST and @DBA profiles deployed automatically
4. Welcome email sent with login instructions

John can log in immediately and start working-zero friction onboarding.

EDITING EXISTING USERS:

Click alice@company.com to edit. Full profile:

USER: Alice Johnson ├─ Email: alice@company.com ├─ Current Role: User ├─ Consumption Profile: PRO (1M tokens/month) └─ Status: Active

USAGE STATISTICS (Current Month): ├─ Tokens: 347,521 / 1,000,000 (34.8%) ├─ Queries: 234 ├─ Sessions: 12 ├─ Total Cost: $34.56 └─ Most Used: @ANALYST (187 queries)

DEPLOYED PROFILES: ├─ @ANALYST (Gemini Flash, Analytics tools) ├─ @DBA (Claude Sonnet, Database tools) └─ Custom_CustomerAnalysis (Alice's personal)

Alice needs elevated permissions for special project.

Change Role: User → **Developer**

Grants Alice:

- Developer tools (custom MCP servers, debugging)
- Advanced integrations
- Debug mode visibility

Save. Role updates instantly. Next login → Developer features visible.

ROLE PERMISSIONS COMPARISON:

What each role can do:

USER (Standard): □ Execute queries via UI □ Create and manage own profiles □ Access marketplace (subscribe to collections) □ View own cost analytics □ Access system configuration □ Manage other users □ Generate long-lived access tokens

DEVELOPER (Advanced): □ All User permissions □ Create custom MCP servers □ Access debug logs and advanced monitoring □ Use developer mode features □ API integration with full documentation □ Manage other users □ Modify system-wide settings

ADMIN (Full Control): □ All Developer permissions □ Manage users (create, edit, suspend, delete) □ Configure LLM providers and MCP servers □ Access global cost analytics (all users) □ Generate long-lived access tokens □ Modify consumption profiles and budgets □ System configuration and maintenance

Role determines capability. Choose appropriately.

CONSUMPTION PROFILE REASSIGNMENT:

Alice (PRO profile) consistently hits 1M token limit. Time to upgrade.

Alice's user edit page:

CONSUMPTION PROFILE CHANGE: ├─ Current: PRO (1M tokens/month) ├─ Change to: ENTERPRISE (10M tokens/month) ├─ Reason: “High-volume Q4 analytics project” └─ Expiration: December 31, 2025 (temporary upgrade)

Save. Alice now has:
- 10M tokens/month (10x capacity)
- 1,000 prompts/day
- 100 prompts/hour

January 1 → Automatically reverts to PRO (scheduled downgrade).

Time-bound escalation-grant power when needed, revoke when project ends.

USER ACTIVITY MONITORING:

Click "Activity Log" tab for alice@company.com. Detailed history:

USER ACTIVITY LOG (Last 30 Days):

Dec 7, 09:15 AM - Login (IP: 192.168.1.45) Dec 7, 09:18 AM - Profile Switch: @ANALYST activated Dec 7, 09:20 AM - Query: “Show Q4 revenue by region” ($0.23) Dec 7, 09:25 AM - Query: “Break down by product category” ($0.31) Dec 7, 10:12 AM - Profile Switch: @DBA activated Dec 7, 10:15 AM - Query: “Analyze slow queries in last 7 days” ($0.45) Dec 7, 11:03 AM - Marketplace: Subscribed to “Customer Insights Playbook” Dec 7, 11:30 AM - Created profile: “Custom_CustomerAnalysis” Dec 7, 02:45 PM - Logout

Dec 6, 08:30 AM - Login (IP: 192.168.1.45) …

```
Every action tracked:
- Logins/logouts (IP addresses for security)
- Profile switches (which personas)
- Query executions (with costs)
- Marketplace interactions (subscriptions, forks)
- Profile creations (custom configurations)

Forensic accountability. Know exactly what Alice did, when, and how much it cost.

**ACCOUNT SUSPENSION:**

Carol Williams left the company two weeks ago. Still "Active" in system.

carol@company.com → Click "Suspend Account":
```

SUSPENSION CONFIRMATION: ├─ User: carol@company.com ├─ Reason: “Employee departure - offboarding” └─ Effect: ├─ User cannot log in ├─ Existing sessions terminated immediately ├─ All scheduled queries cancelled ├─ Profile data preserved (for auditing) └─ No token consumption allowed

```
Click "Suspend." Status → **Suspended**. If she tries to log in:
```

 ACCOUNT SUSPENDED

Your account has been suspended by your administrator.

Reason: Employee departure - offboarding

Contact: admin@company.com

```
Data remains for auditing. Zero access.

**ACCOUNT REACTIVATION:**

Two months later, Carol returns as a contractor. Reactivate her account.

Go to carol@company.com (Status: Suspended) → Click "Reactivate Account."

**Reactivation Options:**
-   Restore previous role (User)
```

- Restore previous consumption profile (PRO)
- Restore deployed profiles (@ANALYST, @DBA)
- Reset password (force new password)
- Clear activity history (start fresh)

Check "Reset password" for security (she's returning after absence).

Click "Reactivate."

Carol receives:
- Reactivation email with temporary password
- Access to all previous profiles and data
- Audit log entry: "Account reactivated by admin@company.com on Dec 7, 2025"

Seamless offboarding and re-onboarding.

BULK USER OPERATIONS:

Need to upgrade 15 analysts from PRO to ENTERPRISE for Q4 project.

Click "Bulk Actions" → Select users:

BULK USER SELECTION: □ alice@company.com □ john.doe@company.com □ mike@company.com … (15 total selected)

Bulk operation:

BULK OPERATION: ├─ Action: Change Consumption Profile ├─ New Profile: ENTERPRISE ├─ Expiration: December 31, 2025 (auto-revert to PRO) └─ Notification: Send email to affected users

Click "Apply." Uderia updates all 15 users in 2 seconds. Each receives:

Subject: Your Uderia consumption profile has been upgraded

Hi Alice,

Your consumption profile has been upgraded from PRO to ENTERPRISE for the Q4 analytics project.

NEW LIMITS: Hourly: 100 prompts (was 30) Daily: 1,000 prompts (was 200) Monthly: 10M tokens (was 1M)

Expires: December 31, 2025 Auto-revert to PRO on January 1.

Happy querying!

No manual user-by-user editing. Efficient bulk management.

USAGE INSIGHTS:

Click "Usage Insights" tab. Organizational patterns:

TOP 5 MOST ACTIVE USERS (Last 30 Days):

1. alice@company.com - 234 queries, $34.56
2. sarah@company.com - 189 queries, $67.89 (expensive models)
3. mike@company.com - 156 queries, $12.34
4. bob@company.com - 98 queries, $2.18 (cost-efficient)
5. john.doe@company.com - 45 queries, $8.21 (new user, ramping up)

DORMANT USERS (No activity 30+ days): • emma@company.com (Last active: 47 days ago) • ryan@company.com (Last active: 62 days ago)

Recommendation: Suspend dormant users to prevent credential leakage.

ROLE DISTRIBUTION: Users: 112 (88%) Developers: 12 (9%) Admins: 3 (3%)

Balanced hierarchy. Most users standard, few with elevated privileges (security best practice)

From "unmanaged user chaos" to "complete administrative control."

From "who has access to what?" to "forensic visibility and governance."

User creation. Role management. Activity tracking. Suspension/reactivation. Bulk operations. C

Screen Capture Plan: - [ ] Admin → User Management landing page - [ ] User table showing 127 users: - Columns: Name, Email, Role, Profile, Status, Last Active, Total Cost - 5 sample users displayed (Alice, Bob, Sarah, Mike, Carol) - [ ] "Add User" button → User creation form: - Basic info (username, email, name, department) - Auto-generated password with "Force change on first login" checkbox - Role radio buttons (User selected) - Consumption profile radio (PRO selected) - Default profile deployment checkboxes (@ANALYST, @DBA checked) - [ ] Click "Create User" → Success confirmation - [ ] Welcome email preview sent to john.doe@company.com - [ ] Click alice@company.com → User detail view: - Current role: User - Consumption: PRO, 347,521/1M tokens (34.8%) - Usage stats: 234 queries, $34.56 - Deployed profiles: @ANALYST, @DBA, Custom_CustomerAnalysis - [ ] Edit role: User → Developer dropdown - [ ] Save → Role updated confirmation - [ ] Role permissions comparison table: - USER: UI access, own profiles, marketplace, own analytics - DEVELOPER: + custom MCP, debug logs, API integration - ADMIN: + user management, system config, global analytics - [ ] Change Alice's consumption profile: - Current: PRO → New: ENTERPRISE - Expiration: Dec 31, 2025 - Reason field: "High-volume Q4 analytics project" - [ ] Save → Profile upgraded, scheduled revert displayed - [ ] Activity Log tab for alice@company.com: - Timestamped entries: Login, profile switches, queries (with costs), marketplace actions - Last 30 days of activity - [ ] carol@company.com (Active) → "Suspend Account" button - [ ] Suspension confirmation modal: - Reason: "Employee departure" - Effects listed (no login, sessions terminated, data preserved) - [ ] Click "Suspend" → Status changes to Suspended - [ ] Carol's login attempt showing: "Account Suspended" error - [ ] Later: carol@company.com (Suspended) → "Reactivate Account" - [ ] Reactivation options: - Restore role, profile, deployed profiles (checked) - Reset password (checked) - Clear history (unchecked) - [ ] Click "Reactivate" → Reactivation email sent - [ ] Bulk Actions interface: - 15 users selected (checkboxes) - Action: Change Consumption Profile → ENTERPRISE - Expiration: Dec 31, 2025 - Notification: Send email (checked) - [ ] Click "Apply" → 15 users updated in 2 seconds - [ ] Bulk notification email

preview - [ ] Usage Insights tab: - Top 5 active users with query counts and costs - Dormant users list (30+ days inactive) - Role distribution pie chart (88% User, 9% Developer, 3% Admin)

Supporting Materials: - User management documentation - Role permissions matrix (detailed) - Consumption profile assignment guide - Activity monitoring and auditing guide - Suspension vs deletion comparison - Bulk operations reference - Security best practices for user administration

---

8.2 Long-Lived Access Tokens

Duration: 3-4 minutes
Learning Objectives: - Generate long-lived access tokens for API integrations - Understand token security and scope - Manage token lifecycle and revocation

Key Topics to Cover: - [ ] Token generation (90 days or 1 year) - [ ] One-time display security pattern - [ ] SHA256 hashed storage - [ ] Usage tracking (last used, count, IP) - [ ] Token revocation and soft-delete - [ ] API authentication with tokens

Narration Script:

```
JWT session tokens (Module 5.3) expire after 24 hours. Perfect for human users.

But what about CI/CD pipelines? Airflow workflows? Production automation?

Long-Lived Access Tokens-persistent credentials for unattended programmatic access.

**TOKEN GENERATION:**

Admin → API Tokens → "Generate New Token":
```

TOKEN CONFIGURATION: ├— Name: airflow_prod_pipeline ├— Description: “Production Airflow DAG for nightly analytics” ├— Associated User: automation@company.com (service account) ├— Lifespan: 1 year ├— Scopes: query:execute, session:read (least privilege) ├— Profile Restriction: @COST only (prevent expensive models) └— Rate Limit: 100/hour, 200/day

```
Click "Generate."

**ONE-TIME DISPLAY:**

Uderia shows:
```

□ TOKEN: uderia_sk_live_xK9mP2vL7qR4nT8hS6jY3wZ1aF5cD9bE

□ SECURITY WARNING: Shown ONCE only. Store securely: □ AWS Secrets Manager / HashiCorp Vault □ NOT in code repositories □ NOT in plaintext files

```
Copy token → Store in AWS Secrets Manager → Click "I Have Saved This Token."

Plaintext disappears forever. Only SHA256 hash stored in database.
```

API USAGE:

Airflow DAG:

```python
import requests
import os

token = os.environ['UDERIA_TOKEN']  # From AWS Secrets

headers = {'Authorization': f'Bearer {token}'}
payload = {'profile': '@COST', 'query': '...'}

response = requests.post(
    'https://uderia.company.com/api/v1/query',
    headers=headers,
    json=payload
)
```

Token authenticates as automation@company.com with limited scope.

TOKEN MANAGEMENT:

Admin → API Tokens shows all active tokens:

Token Name	User	Scopes	Last Used	Created	Expires	Status
airflow_prod	automation	query:execute	2h ago	Dec 1	Dec 1, 2025	Active
legacy_token	old-service	admin:all	47 days ago	Mar 5	NEVER	□ Risky

Security red flag: legacy_token has admin:all scope, never expires, dormant 47 days.

Click “Revoke” → Token immediately invalidated → API calls return 401 Unauthorized.

TOKEN ROTATION:

Best practice: Rotate every 90-180 days.

1. Generate new token: airflow_prod_v2
2. Update Airflow with new token
3. Test in staging
4. Deploy to production
5. Verify 1 week
6. Revoke old token

Seamless rotation with zero downtime.

AUDIT LOGGING:

Every token operation logged:

```
TOKEN AUDIT LOG:

Dec 7, 10:23 AM - Token Generated
  Name: airflow_prod_v2
  User: automation@company.com
  Scopes: query:execute
  By: admin@company.com

Dec 7, 09:45 AM - Token Revoked
  Name: legacy_token
  Reason: "Over-privileged, dormant"
  By: admin@company.com
```

Forensic accountability for all API access.

From “authentication blocking automation” to “secure, always-on programmatic access.”

Long-lived tokens. Scoped permissions. Rotation. Audit trails. Automation enabled.

```
**Screen Capture Plan:**
- [ ] Admin → API Tokens → "Generate New Token"
- [ ] Token creation form:
  - Name: airflow_prod_pipeline
  - Description, User: automation@company.com
  - Lifespan: 1 year
  - Scopes: query:execute, session:read (checkboxes)
  - Profile restriction: @COST
  - Rate limits: 100/hr, 200/day
- [ ] Click "Generate" → One-time token display:
  - Token: uderia_sk_live_xK9mP2vL7qR4nT8hS6jY3wZ1aF5cD9bE
  - Security warning (red box)
  - "Shown ONCE only"
  - [Copy] [Download .env] buttons
- [ ] AWS Secrets Manager storage command
- [ ] Click "I Have Saved" → Plaintext disappears
- [ ] Python code example (Airflow DAG with Bearer token)
- [ ] Admin → API Tokens → Active tokens table:
  - 2 tokens shown
  - legacy_token with  warnings (admin:all, NEVER expires, 47 days dormant)
- [ ] Click legacy_token → "Revoke" button
- [ ] Revocation confirmation → Status: Revoked
- [ ] API call with revoked token → 401 Unauthorized
- [ ] Token rotation workflow diagram (5 steps)
- [ ] Audit log showing token operations

**Supporting Materials:**
- Long-lived token documentation
- API authentication guide with code samples
```

```
- Scope and permissions reference
- Token rotation procedures
- Security best practices
- AWS Secrets Manager integration

---

### 8.3 System Configuration & Monitoring
**Duration:** 3-4 minutes
**Learning Objectives:**
- Configure system-wide settings
- Monitor platform health
- Manage LLM providers and MCP servers

**Key Topics to Cover:**
- [ ] System settings panel
- [ ] LLM provider configuration
- [ ] MCP server management
- [ ] Security settings
- [ ] Audit log access
- [ ] Platform customization

**Narration Script:**
```

Users manage profiles. Admins manage the entire system.

System Configuration—control panel for platform-wide settings.

LLM PROVIDER CONFIGURATION:

Admin → System Configuration → “LLM Providers”:

```
ACTIVE PROVIDERS:
  Anthropic (Claude)
  OpenAI (GPT-4)
  Google AI (Gemini)
  AWS Bedrock
  Ollama (Local)
  Azure OpenAI (Not Configured)
```

Add Azure OpenAI. Click “Add Provider” → Select “Azure OpenAI”:

```
AZURE OPENAI CONFIGURATION:
   API Endpoint: https://your-resource.openai.azure.com/
   API Key: ****************************** (encrypted)
   API Version: 2024-06-01
   Deployment Mapping:
      gpt-4o-prod → GPT-4o
      gpt-4o-mini-prod → GPT-4o-mini
   Region: East US
   Retry Config: 3 max, 2s delay, 120s timeout
```

Click “Test Connection” → □ All checks passed → “Save Provider.”

Azure OpenAI now available system-wide.

MCP SERVER REGISTRATION:

Click “MCP Servers”:

```
ACTIVE SERVERS:
1. base_mcp (Database) -  Running
2. qlty_mcp (Quality) -  Running
3. dba_mcp (Admin) -  Running
4. TDA_mcp (Platform) -  Running
5. custom_erp (ERP Integration) -  Unstable
```

Register Salesforce MCP server. Click “Register MCP Server”:

```
SALESFORCE MCP CONFIGURATION:
  Server Name: salesforce_mcp_server
  Description: "Salesforce CRM integration"
  Endpoint: https://salesforce-mcp.company.com/mcp
  Authentication: OAuth 2.0
     Client ID: 3MVG9...
     Client Secret: ********************************
     Token Endpoint: https://login.salesforce.com/.../token
  Tools Auto-Discovered:
      salesforce_getAccount
      salesforce_getOpportunities
      salesforce_createLead
      salesforce_updateContact
```

Click “Test Connection” → □ 4 tools registered → “Register Server.”

Salesforce tools now available for custom profiles.

DEFAULT PROFILES:

Click “Default Profiles.”

New users automatically get: - **@ANALYST** (Gemini Flash, base tools) - **@LOCAL** (optional for compliance users)

Edit @ANALYST → Add: - Knowledge: “SQL Optimization Best Practices” - Planner: “Customer Insights Playbook”

Save. All new users get @ANALYST pre-loaded with expert knowledge.

SECURITY SETTINGS:

Click “Security Settings”:

```
AUTHENTICATION:
  JWT Secret: ******* (last rotated 23 days ago)
  Session Duration: 24 hours
  Password Requirements: 12 chars, complexity rules
```

```
ENCRYPTION:
  Algorithm: AES-256-GCM
  All credentials encrypted
  Last key rotation: 68 days ago

NETWORK SECURITY:
  HTTPS enforcement
  CORS policy configured
  Rate limiting: 5 login attempts, 1000 API/hour

AUDIT LOGGING:
  Retention: 90 days
  Events: Logins, auth failures, role changes, config changes
```

Enterprise-grade security configured.

PLATFORM SETTINGS:

Click “Platform Settings”:

```
BRANDING:
  Platform Name: Uderia (or customize)
  Logo: [Upload Custom]
  Primary Color: #4A90E2
  Contact: support@company.com

FEATURES:
 Marketplace Enabled
 Hybrid Intelligence
 Cost Analytics
 Beta Features

NOTIFICATIONS:
  Email: SMTP configured
  Slack: Webhook configured
```

Save all settings.

CONFIGURATION AUDIT:

Admin → System Configuration → “Audit Log”:

```
CONFIGURATION AUDIT LOG:

Dec 7, 11:23 AM - MCP Server Registered
  Server: salesforce_mcp
  By: admin@company.com

Dec 7, 10:15 AM - Default Profile Modified
  Profile: @ANALYST
  Added: SQL Best Practices collection
  By: admin@company.com
```

```
Dec 5, 02:34 PM - LLM Provider Added
  Provider: Azure OpenAI
  By: admin@company.com

Nov 15, 09:00 AM - JWT Secret Rotated
  Sessions invalidated: 127 users
  By: admin@company.com
```

Forensic accountability for every system change.

From “fixed configuration” to “customizable, enterprise-ready system.”

Providers. MCP servers. Security. Branding. Complete control.

```
**Screen Capture Plan:**
- [ ] Admin → System Configuration landing page (5 categories)
- [ ] Click "LLM Providers" → Provider list (6 providers)
- [ ] "Add Provider" → Azure OpenAI form:
  - Endpoint, API Key (masked), API Version
  - Deployment mapping
  - Retry config
- [ ] "Test Connection" → Validation results
- [ ] "Save Provider" → Azure added
- [ ] Click "MCP Servers" → Server list (5 servers)
- [ ] "Register MCP Server" → Salesforce form:
  - Name, endpoint, OAuth config
  - Auto-discovered tools (4 listed)
- [ ] "Test Connection" →  Registered
- [ ] Click "Default Profiles" → @ANALYST shown
- [ ] Edit → Add collections → Save
- [ ] Click "Security Settings" → Configuration panel:
  - Authentication (JWT, session, passwords)
  - Encryption (AES-256, rotations)
  - Network (HTTPS, CORS, rate limits)
  - Audit logging (90-day retention)
- [ ] Click "Platform Settings" → Branding panel:
  - Name, logo, color, contact
  - Feature flags
  - Email/Slack notifications
- [ ] Audit Log tab → Timestamped configuration changes

**Supporting Materials:**
- System configuration comprehensive guide
- LLM provider setup docs (per provider)
- MCP server integration guide
- Security settings reference
- Platform customization guide
```

```
- Audit logging documentation

---

## Module 9: Advanced Topics
**Goal:** Deep-dive into specialized integrations and deployment scenarios

### 9.1 Flowise Integration: Low-Code Workflows
**Duration:** 4-5 minutes
**Learning Objectives:**
- Understand Flowise + Uderia architecture
- Import pre-built agent flows
- Build custom workflows visually

**Key Topics to Cover:**
- [ ] Flowise overview and value proposition
- [ ] Pre-built Uderia agent flow
- [ ] Async submit + poll pattern implementation
- [ ] Session management in workflows
- [ ] Bearer token authentication
- [ ] TTS payload extraction for voice chatbots

**Narration Script:**
```

Uderia is powerful. But what if you're not a developer?

What if you want visual, drag-and-drop workflow orchestration?

Flowise—the low-code integration layer for Uderia.

FLOWISE OVERVIEW:

Flowise: open-source platform for building AI agent workflows visually.

No Python. No REST API code. Just drag, drop, connect.

Uderia provides **pre-built Flowise agent flow** implementing complete async pattern: - Submit conversation - Poll for status - Extract final response - Format for voice/text chatbots

Import it.

IMPORT UDERIA TEMPLATE:

Flowise → "Import Flow" → Select `uderia_async_agent.json`. Flow appears:

```
FLOW ARCHITECTURE:

[User Input] → [Uderia Submit] → [Polling Loop] → [Response Processor] → [Output]
                      ↓
                [Session Store]

NODE 1: Uderia Submit
  Endpoint: https://uderia.company.com/api/v1/submit
  Method: POST
```

```
  Headers: Authorization: Bearer {token}
  Body:
  {
    "query": "{{userInput}}",
    "profile": "@ANALYST",
    "session_id": "{{sessionId}}"
  }
  Output: session_id for tracking

NODE 2: Polling Loop
  Endpoint: https://uderia.company.com/api/v1/status/{{sessionId}}
  Method: GET
  Interval: 2 seconds
  Max attempts: 30 (1 minute timeout)
  Stop condition: status == "complete"

NODE 3: Response Processor
  Extracts final_answer from status response
  Applies text formatting:
     Remove Markdown for TTS (voice chatbots)
     Preserve Markdown for text chat
     Extract charts separately (if needed)

NODE 4: Session Store
  Persists session_id for context continuity
  Enables multi-turn conversations
  Links follow-up questions to original analysis
```

AUTHENTICATION:

Configure Bearer Token:

```
BEARER TOKEN SETUP:
  Click "Uderia Submit" node
  Headers → Add Authorization
  Value: Bearer uderia_sk_live_xK9mP2vL...
  Token from Admin → API Tokens (Module 8.2)
```

Secure. Persistent. No user login required.

TESTING THE FLOW:

Click “Test Flow.”

Input: “What were total sales in Q4 2024?”

Flowise executes:

```
EXECUTION LOG:
1. Submit to Uderia → session_id: 82b9...
2. Poll status (attempt 1): in-progress
3. Poll status (attempt 2): in-progress
```

```
4. Poll status (attempt 3): complete
5. Extract response:
   Total Q4 2024 sales: $892,450
   - October: $287,320
   - November: $312,180
   - December: $292,950
```

Output: Clean text (no Markdown, TTS-ready).

VOICE CHATBOT INTEGRATION:

Connect Flowise output to:

```
VOICE PLATFORMS:
• Amazon Polly (TTS)
• Twilio Voice (phone calls)
• Alexa Skills
• Google Assistant
```

User speaks: “What were sales in Q4?” Alexa responds: “Total Q4 2024 sales were $892,450…”

Powered by Uderia’s @ANALYST profile. Zero custom code.

TEXT CHATBOT INTEGRATION:

Connect to:

```
TEXT PLATFORMS:
• Slack (workplace assistant)
• WhatsApp (customer support)
• Web chat widgets
```

User types: “Show top 5 customers by revenue” Chatbot responds with formatted Markdown table.

Uderia handles SQL, analysis, charting. Flowise handles routing.

CUSTOM WORKFLOW EXAMPLE:

Multi-step workflow:

```
MULTI-PROFILE ORCHESTRATION:

[User Query] → [Uderia @ANALYST] → [Extract SQL]
                                          ↓
                                [Uderia @DBA optimize SQL]
                                          ↓
                                [Execute Optimized Query]
                                          ↓
                                [Uderia @QUALITY validate results]
                                          ↓
                                [Final Report]
```

Three profiles orchestrated visually. No Python.

From “developer-only platform” to “accessible to business users.”

Flowise + Uderia: Low-code AI orchestration for everyone.

```
**Screen Capture Plan:**
- [ ] Flowise dashboard homepage
- [ ] Click "Import Flow" → Select `uderia_async_agent.json`
- [ ] Flow visualization:
  - User Input node
  - Uderia Submit node (show configuration panel)
  - Polling Loop node (show 2s interval, 30 attempts)
  - Response Processor node
  - Output node
  - Session Store (sidebar)
- [ ] Click "Uderia Submit" → Configuration:
  - Endpoint URL
  - Method: POST
  - Headers: Authorization Bearer token
  - Body JSON template
- [ ] Click "Polling Loop" → Configuration:
  - GET endpoint with {{sessionId}}
  - 2-second interval
  - Stop condition logic
- [ ] Click "Response Processor" → Text formatting options:
  - TTS mode (remove Markdown)
  - Text mode (preserve Markdown)
- [ ] Test Flow panel → Input: "What were Q4 2024 sales?"
- [ ] Execution log showing:
  - Submit → session_id returned
  - Poll 1 → in-progress
  - Poll 2 → in-progress
  - Poll 3 → complete
  - Response extracted
- [ ] Output panel showing clean text response
- [ ] Voice integration diagram:
  - Flowise → Amazon Polly → Alexa
- [ ] Text integration diagram:
  - Flowise → Slack API → Workplace chat
- [ ] Custom multi-profile workflow:
  - 3 Uderia nodes (@ANALYST, @DBA, @QUALITY)
  - Visual connections between nodes

**Supporting Materials:**
- docs/Flowise/FLOWISE_INTEGRATION.md
- uderia_async_agent.json (pre-built flow template)
- Flowise integration guide
- Voice chatbot setup guide
- Multi-profile workflow examples
```

```
---

### 9.2 Docker Deployment: Production Setup
**Duration:** 4-5 minutes
**Learning Objectives:**
- Build Uderia Docker image
- Configure production deployment
- Manage multi-user container

**Key Topics to Cover:**
- [ ] Dockerfile overview
- [ ] Docker Compose configuration
- [ ] Environment variable overrides
- [ ] Volume mounts (sessions, logs, keys)
- [ ] Multi-user shared container
- [ ] Load balancer configuration
- [ ] Horizontal scaling considerations

**Narration Script:**
```

Local development works. Now: production deployment.

Docker containerization for consistent, scalable, multi-user environments.

Uderia includes production-ready `Dockerfile` and `docker-compose.yml`.

DOCKERFILE OVERVIEW:

Open `Dockerfile`:

```
FROM python:3.13-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc postgresql-client curl \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements and install Python packages
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY src/ ./src/
COPY static/ ./static/
COPY templates/ ./templates/
COPY tda_config.json .

# Create directories for persistent data
```

```
RUN mkdir -p /app/tda_sessions /app/logs /app/tda_keys

# Expose port
EXPOSE 5000

# Run application
CMD ["python", "src/trusted_data_agent/main.py"]
```

Lightweight base (Python 3.13). Minimal dependencies. Production-ready.

DOCKER COMPOSE CONFIGURATION:

Open `docker-compose.yml`:

```
version: '3.8'

services:
  uderia:
    build: .
    container_name: uderia_production
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
      - JWT_SECRET=${JWT_SECRET}
      - DATABASE_URL=${DATABASE_URL}
      - ANTHROPIC_API_KEY=${ANTHROPIC_API_KEY}
      - GOOGLE_API_KEY=${GOOGLE_API_KEY}
      - OPENAI_API_KEY=${OPENAI_API_KEY}
    volumes:
      - ./tda_sessions:/app/tda_sessions
      - ./logs:/app/logs
      - ./tda_keys:/app/tda_keys
      - ./tda_config.json:/app/tda_config.json
    restart: unless-stopped
    networks:
      - uderia_network

  chromadb:
    image: chromadb/chroma:latest
    container_name: uderia_chromadb
    volumes:
      - chroma_data:/chroma/chroma
    ports:
      - "8000:8000"
    restart: unless-stopped
    networks:
      - uderia_network

volumes:
```

```
  chroma_data:

networks:
  uderia_network:
    driver: bridge
```

Two services: Uderia app + ChromaDB vector store.

ENVIRONMENT VARIABLES:

Create `.env` file (never commit to Git):

```
JWT_SECRET=your-secret-key-here
DATABASE_URL=teradata://prod-server:1025/fitness_db
ANTHROPIC_API_KEY=sk-ant-api03-...
GOOGLE_API_KEY=AIza...
OPENAI_API_KEY=sk-proj-...
```

Secure credential management. No hardcoded secrets.

VOLUME MOUNTS:

Three critical volumes:

```
1. tda_sessions/ - Multi-user conversation persistence
     Each user gets isolated directory
     Survives container restarts
     Example: tda_sessions/82b9231b-.../session.json

2. logs/ - Application and audit logging
     app.log (General application events)
     consumption.log (Token usage tracking)
     audit.log (Security events)

3. tda_keys/ - Encrypted credential storage
     API keys for MCP servers
     OAuth tokens
     Encrypted at rest (AES-256-GCM)
```

Data persists across deployments.

BUILD AND RUN:

Terminal:

```
# Build image
docker-compose build

# Start services
docker-compose up -d

# View logs
docker-compose logs -f uderia
```

```
# Check status
docker-compose ps
```

Output:

```
NAME                  STATUS    PORTS
uderia_production     Up        0.0.0.0:5000->5000/tcp
uderia_chromadb       Up        0.0.0.0:8000->8000/tcp
```

Both services running. Multi-user ready.

MULTI-USER ACCESS:

Single container supports 100+ concurrent users:

```
CONCURRENT SESSIONS:
• User 1: sarah@company.com → Session 82b9... → @ANALYST
• User 2: marcus@company.com → Session f4c2... → @DBA
• User 3: priya@company.com → Session a1d8... → @QUALITY
```

Isolated sessions. Shared infrastructure. Cost-efficient.

PRODUCTION DEPLOYMENT:

Deploy to:

```
CLOUD PLATFORMS:
• AWS ECS (Elastic Container Service)
• Google Cloud Run
• Azure Container Instances
• Kubernetes (advanced orchestration)

ADD:
• NGINX reverse proxy (SSL termination, load balancing)
• PostgreSQL for user management (replace SQLite)
• Redis for session caching (optional performance boost)
• Prometheus monitoring
• Grafana dashboards
```

Enterprise-ready architecture.

HORIZONTAL SCALING:

Multiple Uderia containers behind load balancer:

```
SCALE ARCHITECTURE:

[Load Balancer]
    |
      [Uderia Container 1] (users 1-50)
      [Uderia Container 2] (users 51-100)
      [Uderia Container 3] (users 101-150)
         |
    [Shared ChromaDB]
    [Shared PostgreSQL]
```

Stateless design enables infinite horizontal scaling.

From “local prototype” to “production-grade, multi-user platform.”

Docker. Compose. Volumes. Environment variables. Scale with confidence.

```
**Screen Capture Plan:**
- [ ] VS Code: Open `Dockerfile`
  - Highlight: FROM python:3.13-slim
  - Highlight: COPY requirements.txt, RUN pip install
  - Highlight: EXPOSE 5000, CMD
- [ ] VS Code: Open `docker-compose.yml`
  - Highlight: Two services (uderia, chromadb)
  - Highlight: Ports mapping (5000:5000)
  - Highlight: Environment variables block
  - Highlight: Volume mounts (3 directories)
  - Highlight: restart: unless-stopped
- [ ] VS Code: Open `.env` file
  - Show masked API keys
  - JWT_SECRET, DATABASE_URL, provider keys
- [ ] Terminal: `docker-compose build`
  - Build progress (downloading base image, installing packages)
  - "Successfully built" message
- [ ] Terminal: `docker-compose up -d`
  - Starting services
  - "Started" confirmation
- [ ] Terminal: `docker-compose ps`
  - Table showing 2 containers running
  - Status: Up, Ports exposed
- [ ] Terminal: `docker-compose logs -f uderia`
  - Live log stream showing:
    - "Starting Uderia platform..."
    - "Connected to ChromaDB"
    - "Server running on 0.0.0.0:5000"
- [ ] File explorer: `tda_sessions/` directory
  - Multiple session folders (82b9..., f4c2..., a1d8...)
  - Show session.json inside one folder
- [ ] File explorer: `logs/` directory
  - app.log, consumption.log, audit.log
  - Show file sizes growing
- [ ] Browser: http://localhost:5000
  - Login as User 1 (sarah@company.com)
  - New tab: Login as User 2 (marcus@company.com)
  - New tab: Login as User 3 (priya@company.com)
  - All three sessions active simultaneously
- [ ] Production architecture diagram:
  - Load balancer → 3 Uderia containers
```

```
  - Shared ChromaDB, PostgreSQL
  - Monitoring stack (Prometheus, Grafana)
- [ ] Deployment checklist overlay:
  -   Environment variables secured
  -   Volumes mounted
  -   SSL configured
  -   Monitoring enabled
  -   Backup strategy

**Supporting Materials:**
- Docker deployment comprehensive guide
- docker-compose.yml reference with all options
- Production deployment checklist
- Cloud provider deployment guides (AWS, GCP, Azure)
- Kubernetes manifests (advanced)
- Monitoring and logging setup guide

---

### 9.3 Developer Mode & Custom Integrations
**Duration:** 3-4 minutes
**Learning Objectives:**
- Enable developer features
- Create custom MCP servers
- Build custom integrations

**Key Topics to Cover:**
- [ ] Developer role activation
- [ ] Custom MCP server development
- [ ] Tool/prompt registration
- [ ] Custom prompt templates
- [ ] Plugin system for Planner Repository Constructors
- [ ] REST API extension patterns

**Narration Script:**
```

Uderia provides 6 pre-built profiles. But what if you need custom integrations?

Proprietary systems? Internal APIs? Custom workflows?

Developer Mode unlocks the extensibility layer.

DEVELOPER ROLE ACTIVATION:

Admin → Users → Edit your account → Role: **Developer**.

New sidebar menu appears:

```
DEVELOPER MENU:
```
- Developer Tools
- Custom MCP Servers

- Plugin System
- Custom Prompts
- API Explorer

Build a custom integration.

CUSTOM MCP SERVER:

Scenario: Company uses **internal ERP system** (not publicly available). Want @ANALYST to query ERP data directly.

Create custom MCP server: `erp_mcp`

Step 1: Define Tools

Developer Tools → "Create MCP Server" → `erp_mcp`. Add tool:

```
TOOL DEFINITION:
name: "erp_getInventory"
description: "Query current inventory levels from ERP"
parameters:
  - product_code: string (required)
  - warehouse: string (optional)
returns: JSON with stock levels, location, reorder status
```

Step 2: Implement Tool

Connect to internal ERP API:

```python
import requests

def erp_getInventory(product_code, warehouse=None):
    headers = {'Authorization': f'Bearer {ERP_TOKEN}'}
    params = {'product': product_code}
    if warehouse:
        params['warehouse'] = warehouse

    response = requests.get(
        'https://erp.company.com/api/inventory',
        headers=headers,
        params=params
    )

    return response.json()
```

Step 3: Register with Uderia

Developer Tools → MCP Servers → `erp_mcp` → "Deploy." Uderia validates:

```
VALIDATION:
 Tool callable
 Parameters schema valid
 Authentication working
 Response format JSON
```

Tool now available.

Step 4: Attach to Profile

Profiles → Edit @ANALYST → Tools → Add `erp_getInventory`. Save profile.

CUSTOM PROMPT TEMPLATES:

Create prompt for ERP context:

Developer Tools → Custom Prompts → "Create Prompt"

Prompt Name: `erp_inventoryContext` **Prompt Template:**

```
You have access to internal ERP inventory data.

When analyzing product availability:
1. Query current stock levels with erp_getInventory
2. Check warehouse distribution
3. Flag products below reorder threshold (< 50 units)
4. Recommend restocking priorities

Always include:
- Product code
- Current stock
- Warehouse location
- Days until stockout (if low)
```

Save → Attach to @ANALYST profile.

Now @ANALYST understands ERP context automatically.

PLUGIN SYSTEM:

Planner Repository Constructors use plugins for custom case types.

Example: Custom "Procurement Analysis" case type.

Developer Tools → Plugins → "Create Plugin"

Plugin Name: `procurement_analysis_plugin` **Case Type:** Procurement **Template Structure:**

```
class ProcurementAnalysisPlugin:
    def extract_metadata(self, conversation):
        return {
            'suppliers': extract_suppliers(conversation),
            'products': extract_products(conversation),
            'cost_metrics': extract_costs(conversation),
            'delivery_times': extract_timelines(conversation)
        }

    def champion_criteria(self):
        return {
            'min_cost_savings': 10000,  # $10k minimum
            'supplier_diversity': True,
```

```
            'on_time_delivery': '>95%'
        }
```

Deploy plugin → Planner Repositories now support custom case type.

REST API EXTENSIONS:

Developer Mode exposes advanced API endpoints:

Endpoint: `/api/v1/developer/execute-custom-workflow`

```
import requests

payload = {
    'workflow': 'multi_profile_pipeline',
    'steps': [
        {'profile': '@ANALYST', 'query': 'Find top revenue products'},
        {'profile': '@DBA', 'query': 'Optimize the SQL query'},
        {'profile': '@QUALITY', 'query': 'Validate data accuracy'}
    ],
    'session_id': '82b9...'
}

response = requests.post(
    'https://uderia.company.com/api/v1/developer/execute-custom-workflow',
    headers={'Authorization': f'Bearer {token}'},
    json=payload
)
```

Orchestrate multiple profiles programmatically. Full control.

DEBUGGING TOOLS:

Developer Tools → "Live MCP Debugger"

Test tools in real-time: - Input: `erp_getInventory(product_code='BIKE-001')` - Output: `{"stock": 127, "warehouse": "WH-East", "reorder": false}` - Latency: 234ms

Debug before deploying to profiles.

From "closed platform" to "fully extensible ecosystem."

Custom MCP servers. Plugins. API extensions. Developer Mode. Build anything.

```
**Screen Capture Plan:**
- [ ] Admin → Users → Edit user account
  - Change Role from "User" to "Developer"
  - Save
- [ ] Sidebar: New "Developer Tools" menu appears
  -  Developer Tools
  -  Custom MCP Servers
  -  Plugin System
```

```
  -   Custom Prompts
  -   API Explorer
- [ ] Developer Tools → "Create MCP Server"
  - Server Name: erp_mcp
  - Description: "Internal ERP integration"
- [ ] Add Tool: `erp_getInventory`
  - Tool definition form:
    - Name, description
    - Parameters: product_code (string), warehouse (optional)
    - Returns: JSON schema
- [ ] VS Code: Show Python implementation
  - Function connecting to ERP API
  - Authentication, parameters, response handling
- [ ] Developer Tools → "Deploy" button
  - Validation checks (4 green checkmarks)
  - "Tool Deployed Successfully"
- [ ] Profiles → Edit @ANALYST → Tools tab
  - Search: erp_getInventory
  - Add to profile
  - Save
- [ ] Developer Tools → Custom Prompts → "Create Prompt"
  - Prompt Name: erp_inventoryContext
  - Prompt Template editor (showing full template)
  - Attach to @ANALYST profile
  - Save
- [ ] Developer Tools → Plugins → "Create Plugin"
  - Plugin Name: procurement_analysis_plugin
  - Case Type: Procurement
  - VS Code: Show plugin class code
- [ ] Developer Tools → "Live MCP Debugger"
  - Input: erp_getInventory(product_code='BIKE-001')
  - Execute button
  - Output JSON displayed
  - Latency: 234ms
- [ ] VS Code: Custom workflow API example
  - Python code showing multi-profile pipeline
  - POST to /api/v1/developer/execute-custom-workflow
- [ ] Terminal: Execute custom workflow
  - Response showing 3 steps completed

**Supporting Materials:**
- Developer Mode comprehensive guide
- Custom MCP server development tutorial
- Plugin system architecture documentation
- REST API extension reference
- Debugging tools guide
- Example custom integrations (Salesforce, SAP, ServiceNow)
```

```
---

### 9.4 Installing Your Own Uderia Platform
**Duration:** 4-5 minutes
**Learning Objectives:**
- Clone repository and setup
- Configure environment
- Run platform locally
- Verify installation

**Key Topics to Cover:**
- [ ] Prerequisites (Python, Node.js)
- [ ] Git clone and virtual environment
- [ ] Dependencies and configuration
- [ ] Database initialization
- [ ] First run and verification

**Narration Script:**
```

You've seen what Uderia can do. Now: install your own instance.

Zero to running platform in 10 minutes.

PREREQUISITES:

Verify:

```
python3 --version  # Python 3.11 or higher
node --version     # Node.js 18+ (for optional UI builds)
git --version      # Git 2.0+
```

If missing: Install Python (python.org), Node (nodejs.org), Git (git-scm.com).

Also needed: Database (Teradata/PostgreSQL/MySQL) + LLM API keys.

CLONE & SETUP:

```
cd ~/projects
git clone https://github.com/your-org/uderia.git
cd uderia

# Virtual environment
python3 -m venv venv
source venv/bin/activate  # macOS/Linux (or venv\Scripts\activate on Windows)
venv\Scripts\activate     # Windows
```

Prompt changes: `(venv) $`

INSTALL DEPENDENCIES:

```
pip install --upgrade pip
pip install -r requirements.txt
```

Installs 47 packages (~2-3 minutes):

```
KEY PACKAGES:
```
- Flask (web framework)
- anthropic, openai, google-generativeai (LLM clients)
- chromadb (vector store)
- teradatasql (database connector)
- PyJWT (authentication)
- And more...

Output:

```
Successfully installed:
  anthropic-0.39.0
  chromadb-0.5.23
  Flask-3.1.0
  ...
    All dependencies installed
```

CONFIGURE PLATFORM:

Open `tda_config.json` in editor:

```
{
  "database": {
    "type": "teradata",
    "host": "your-teradata-server.com",
    "port": 1025,
    "username": "dbc",
    "password": "your-password",
    "default_database": "fitness_db"
  },
  "llm_providers": {
    "anthropic": {
      "api_key": "sk-ant-api03-...",
      "default_model": "claude-sonnet-4-20250514"
    },
    "google": {
      "api_key": "AIza...",
      "default_model": "gemini-1.5-flash"
    },
    "openai": {
      "api_key": "sk-proj-...",
      "default_model": "gpt-4o"
    },
    "ollama": {
      "base_url": "http://localhost:11434",
      "default_model": "llama3.1:70b"
    }
  },
  "chromadb": {
    "host": "localhost",
```

```
    "port": 8000,
    "persist_directory": "./chroma_data"
  },
  "security": {
    "jwt_secret": "your-random-secret-key-here",
    "session_duration_hours": 24
  },
  "server": {
    "host": "0.0.0.0",
    "port": 5000,
    "debug": false
  }
}
```

Critical: Replace placeholder values with your actual credentials.

For testing without real database, use SQLite:

```
"database": {
  "type": "sqlite",
  "path": "./test.db"
}
```

INITIALIZE DATABASE:

Create user management tables:

```
python maintenance/initialize_database.py
```

Output:

```
INITIALIZATION:

Creating tables...
  users table created
  profiles table created
  consumption_logs table created
  api_tokens table created

Creating admin user...
  Admin: admin@uderia.local / admin123

Initialization complete!
```

Default admin credentials created.

START CHROMADB:

Separate terminal:

```
chroma run --path ./chroma_data --port 8000
```

Output:

```
Running Chroma
Listening on http://localhost:8000
```

Vector store ready.

START UDERIA:

Original terminal:

```
python src/trusted_data_agent/main.py
```

Output:

```
 UDERIA PLATFORM STARTING...

 Configuration loaded
 Database connected: Teradata (fitness_db)
 ChromaDB connected: localhost:8000
 LLM Providers initialized:
  • Anthropic (Claude Sonnet 3.5)
  • Google AI (Gemini 1.5 Flash)
  • OpenAI (GPT-4o)
  • Ollama (Llama 3.1 70B)
 MCP Servers loaded: 4 active
 Profile system ready: 6 default profiles

 Server running on http://localhost:5000

Press CTRL+C to stop
```

Platform running!

FIRST ACCESS:

Open browser: `http://localhost:5000`

Login screen appears.

Credentials: - Username: `admin@uderia.local` - Password: `admin123`

Click “Login” → Dashboard loads.

You see: - Welcome message - 6 default profiles (@ANALYST, @DBA, @QUALITY, @LOCAL, @PROD, @COST) - “Start Conversation” button

VERIFICATION TEST:

Click @ANALYST → “Start Conversation.”

Query: “List all databases available.”

Response:

```
VERIFICATION RESULTS:

Available databases:
1. fitness_db (default)
```

- Tables: Customers, Products, Sales, ServiceTickets
- Size: 2.3 GB
- Last updated: Dec 7, 2025

```
Database connection:  Successful
MCP tools:  Working
LLM provider:  Gemini Flash responding
```

Installation verified!

TROUBLESHOOTING:

Common issues:

```
ISSUE 1: "Database connection failed"
Fix:
• Check tda_config.json credentials
• Verify server reachable: ping your-server.com

ISSUE 2: "ChromaDB not responding"
Fix:
• Start ChromaDB first: chroma run --port 8000
• Check port not in use: lsof -i :8000

ISSUE 3: "LLM provider error"
Fix:
• Verify API keys valid
• Test: curl to provider API directly

ISSUE 4: "Module not found"
Fix:
• Activate virtual environment: source venv/bin/activate
• Reinstall: pip install -r requirements.txt
```

From “interested observer” to “platform owner.”

Installed. Configured. Running. Your Uderia instance is live.

```
**Screen Capture Plan:**
- [ ] Terminal: Prerequisites check
  - `python3 --version` → Python 3.13.0
  - `node --version` → v20.11.0
  - `git --version` → git 2.43.0
- [ ] Browser: GitHub repository page
  - https://github.com/your-org/uderia
  - Green "Code" button
  - Copy HTTPS URL
- [ ] Terminal: Clone repository
  - `cd ~/projects`
```

```
  - `git clone https://github.com/your-org/uderia.git`
  - Progress: Cloning... 100%
  - `cd uderia`
  - `ls -la` showing repository structure
- [ ] Terminal: Virtual environment
  - `python3 -m venv venv`
  - `source venv/bin/activate`
  - Prompt changes to `(venv) $`
- [ ] Terminal: Install dependencies
  - `pip install --upgrade pip`
  - `pip install -r requirements.txt`
  - Progress bar, package installation
  - "Successfully installed" message with 47 packages
- [ ] VS Code: Open `tda_config.json`
  - Show database configuration section
  - Show LLM providers section with API keys (masked)
  - Show ChromaDB configuration
  - Show security settings
  - Highlight: "Replace with your credentials"
- [ ] Terminal: Initialize database
  - `python maintenance/initialize_database.py`
  - Output showing table creation
  - Admin user created message
- [ ] Terminal 2: Start ChromaDB
  - `chroma run --path ./chroma_data --port 8000`
  - "Running Chroma" message
  - "Listening on http://localhost:8000"
- [ ] Terminal 1: Start Uderia
  - `python src/trusted_data_agent/main.py`
  - Startup sequence (7 checkmarks)
  - "Server running on http://localhost:5000"
- [ ] Browser: Navigate to http://localhost:5000
  - Login screen appears
  - Enter: admin@uderia.local / admin123
  - Click "Login"
- [ ] Browser: Dashboard loads
  - Welcome message
  - 6 profile cards displayed
  - "Start Conversation" button
- [ ] Browser: Click @ANALYST → Start Conversation
  - Input: "List all databases available."
  - Send
  - Response showing databases, tables, verification checkmarks
- [ ] Troubleshooting overlay:
  - 4 common issues with fixes
  - Terminal commands for debugging

**Supporting Materials:**
```

- README.md complete installation section
- Installation and Setup comprehensive guide
- Troubleshooting documentation with 20+ common issues
- Configuration reference for all tda_config.json options
- Video walkthrough (installation screencast)
- Docker alternative guide (Module 9.2)

Profile Ecosystem Reference
Complete Persona Lineup Used Throughout Tutorials

This section defines the core profiles referenced across all modules and demonstrated in user j

---

Profile 1: @ANALYST - Business Intelligence Persona

User Role: Business Analyst / Product Manager / Marketing Analyst
LLM Provider: Gemini 1.5 Flash
Cost Profile: Budget-conscious ($0.075 per 1M input tokens)
Speed: Fast (optimized for exploratory work)

MCP Tools Available:
- `base_databaseList` - List databases
- `base_tableList` - List tables
- `base_tableDDL` - Get table structure
- `base_readQuery` - Execute SQL queries
- `TDA_Charting` - Generate visualizations (bar charts, line graphs, pie charts)
- `TDA_LLMTask` - Natural language analysis and synthesis
- `TDA_FinalReport` - Format executive-ready reports
- `TDA_CurrentDate` - Date/time utilities

MCP Prompts Available:
- `base_databaseBusinessDesc` - Business context for databases
- `base_tableBusinessDesc` - Business context for tables

Knowledge Collections Attached:
- "Business Metrics Definitions" (15 documents) - KPI calculation formulas, reporting standards
- "fitness_db Business Context" (8 documents) - Product categories, customer segments, sales cy
- "Executive Reporting Templates" (12 documents) - Dashboard layouts, summary formats

Planner Repositories Attached:
- "Executive_Reporting_Patterns" (23 champion cases) - Board presentations, monthly reports
- "Customer_Segmentation_Analysis" (17 champion cases) - Demographic analysis, cohort studies
- "Trend_Analysis_Patterns" (31 champion cases) - Time-series queries, YoY comparisons

Primary Use Cases:

```
- Revenue analysis and trending
- Customer demographics and segmentation
- Product performance analysis
- Executive dashboards and visualizations
- Business insight generation

**Behavioral Characteristics:**
- Business-friendly language (avoids technical jargon)
- Always includes visualizations when appropriate
- Frames insights in actionable recommendations
- Focuses on "why" and "so what" not just "what"

**Tutorial Appearances:**
- Module 1.3: First conversation (revenue analysis)
- Module 2.2: Profile introduction and "Pro Mode" demonstration
- Module 4.1: Strategic planning with business context
- Module 6.2: Marketplace planner repository usage
- Module 10.1: Complete business analyst journey

---

### Profile 2: @DBA - Database Administrator Persona

**User Role:** Database Administrator / Data Engineer / SQL Developer
**LLM Provider:** Claude Sonnet 3.5
**Cost Profile:** Balanced ($3 per 1M input tokens)
**Focus:** Reliability and SQL optimization

**MCP Tools Available:**
- `base_databaseList` - List databases
- `base_tableList` - List tables
- `base_tableDDL` - Get table DDL
- `base_readQuery` - Execute SQL queries
- `dba_databaseVersion` - System version info
- `qlty_profileColumn` - Data profiling and statistics
- `qlty_validateData` - Data quality checks
- `TDA_FinalReport` - Technical report formatting
- `TDA_SystemLog` - System logging

**MCP Prompts Available:**
- `dba_databaseLineage` - Data lineage analysis
- `dba_databaseHealthAssessment` - System health diagnostics
- `dba_userActivityAnalysis` - User activity tracking
- `dba_systemVoice` - System status narratives
- `dba_tableArchive` - Archiving recommendations
- `dba_tableDropImpact` - Impact analysis for schema changes

**Knowledge Collections Attached:**
```

- "Teradata SQL Optimization Guide" (23 documents) - Query performance, index strategies, join
- "fitness_db Schema Documentation" (5 documents) - Table relationships, constraints, business
- "Database Performance Tuning" (18 documents) - Execution plan analysis, statistics management
- "SQL Best Practices" (14 documents) - Code standards, anti-patterns

Planner Repositories Attached:
- "DBA_Champion_Cases" (47 champion cases) - Optimized SQL patterns, performance-tuned queries
- "Performance_Troubleshooting_Playbooks" (22 cases) - Diagnostic workflows, optimization strat
- "Schema_Analysis_Patterns" (15 cases) - DDL extraction, relationship mapping

Primary Use Cases:
- Query optimization and tuning
- Schema analysis and documentation
- Performance troubleshooting
- Data profiling and quality assessment
- System health monitoring

Behavioral Characteristics:
- Technical precision (includes execution plans, statistics)
- Optimization-focused (always suggests performance improvements)
- Security-conscious (validates permissions, checks constraints)
- Uses technical terminology appropriately

Tutorial Appearances:
- Module 2.2: Profile creation and "Pro Mode" demonstration (first persona)
- Module 3.2: Capabilities discovery (MCP tools showcase)
- Module 4.1: Strategic planning with SQL optimization
- Module 8.2: System administration tasks
- Module 10.2: Complete DBA journey

Profile 3: @QUALITY - Data Quality Engineer Persona

User Role: Data Quality Analyst / Data Steward / Governance Specialist
LLM Provider: Claude Sonnet 3.5
Cost Profile: Balanced ($3 per 1M input tokens)
Focus: Thoroughness and validation

MCP Tools Available:
- `base_readQuery` - Data sampling queries
- `base_tableList` - List tables for audit
- `base_tableDDL` - Get table structure for validation
- `qlty_profileColumn` - Column profiling (nulls, distributions, outliers)
- `qlty_validateData` - Constraint validation, referential integrity
- `TDA_LLMTask` - Anomaly analysis and pattern detection
- `TDA_FinalReport` - Quality report formatting
- `TDA_SystemLog` - Audit logging

```
**MCP Prompts Available:**
- `qlty_databaseQuality` - Comprehensive quality assessment
- `base_tableBusinessDesc` - Business rules verification

**Knowledge Collections Attached:**
- "Data Quality Standards" (20 documents) - Organization-specific validation rules, acceptable
- "fitness_db Constraints Documentation" (6 documents) - Expected value ranges, referential int
- "Industry Data Validation Best Practices" (16 documents) - ISO standards, quality frameworks
- "Anomaly Detection Techniques" (11 documents) - Statistical methods, outlier identification

**Planner Repositories Attached:**
- "Quality_Validation_Patterns" (29 champion cases) - Standard validation workflows, constraint
- "Anomaly_Detection_Workflows" (18 cases) - Statistical analysis, outlier investigation
- "Data_Profiling_Templates" (24 cases) - Column analysis, distribution studies

**Primary Use Cases:**
- Data quality audits
- Constraint validation
- Anomaly detection
- Data profiling and statistics
- Compliance verification

**Behavioral Characteristics:**
- Thorough and methodical (validates all constraints)
- Flags issues with severity levels (critical, warning, info)
- Provides remediation recommendations
- Documents quality scores and metrics

**Tutorial Appearances:**
- Module 2.2: Profile introduction and "Pro Mode" demonstration (second persona)
- Module 3.1: Transparent execution (validation workflows)
- Module 10.3: Complete data quality analyst journey

---

### Profile 4: @LOCAL - Privacy-First Persona

**User Role:** Compliance Officer / Privacy-Conscious User / Regulated Industry Analyst
**LLM Provider:** Ollama Llama 3.1 70B (On-Premises)
**Cost Profile:** Zero cloud cost ($0.00)
**Deployment:** Fully local execution, zero external API calls

**MCP Tools Available:**
- Same tools as @ANALYST profile
- All execution happens on-premises
- No data leaves organizational boundaries
```

Knowledge Collections Attached:
- Same collections as @ANALYST
- RAG retrieval happens locally
- Documents stored on-premises

Planner Repositories Attached:
- Same patterns as @ANALYST
- Champion cases stored locally
- Learning happens within sovereign infrastructure

Primary Use Cases:
- Customer PII analysis (names, emails, addresses)
- Financial data analysis (sensitive transactions)
- Healthcare data (HIPAA compliance)
- Regulated industry workflows
- Competitive intelligence (confidential data)

Behavioral Characteristics:
- Identical capabilities to cloud profiles
- Zero latency for data access (local MCP tools)
- Slightly slower LLM inference (local model)
- Complete data sovereignty guaranteed

Key Differentiator:
- Cloud-level reasoning architecture (strategic planning, RAG, optimization)
- Local execution (all sensitive data stays on-premises)
- No compromise on intelligence or features

Tutorial Appearances:
- Module 2.2: Profile introduction and "Pro Mode" demonstration (fourth persona)
- Module 5.2: Hybrid intelligence (local model showcase)
- Module 5.3: Security and sovereignty
- Module 10.4: Complete compliance officer journey

Profile 5: @PROD - Production Workhorse Persona

User Role: Default profile for automation, API calls, scheduled jobs
LLM Provider: Claude Sonnet 3.5
Cost Profile: Balanced ($3 per 1M input tokens)
Focus: Reliability for production workloads

MCP Tools Available:
- All `base_*` tools (complete database access)
- All `TDA_*` platform tools (charting, reporting, logging)
- `dba_databaseVersion` (system info)
- `qlty_*` tools (optional, can be enabled)

Knowledge Collections Attached:
- "General Purpose Documentation" (12 documents) - Cross-domain knowledge base
- "fitness_db Complete Schema Guide" (10 documents) - Full database documentation

Planner Repositories Attached:
- "Cross_Domain_Champion_Cases" (89 cases) - Diverse query patterns from all domains
- "Production_Automation_Patterns" (34 cases) - Scheduled reports, batch jobs

Primary Use Cases:
- REST API automation
- Apache Airflow DAG execution
- Scheduled batch reports
- Production data pipelines
- Generic multi-purpose queries

Behavioral Characteristics:
- Reliable and consistent (minimizes experimental features)
- Optimized for automation (predictable outputs)
- Error-tolerant (graceful handling of failures)
- Well-documented results (structured JSON outputs)

Tutorial Appearances:
- Module 2.1: Two-in-one paradigm (conversational → API)
- Module 2.3: REST API demonstrations
- Module 2.4: Airflow integration
- Module 9.1: Flowise workflow automation
- Module 10.5: "God mode" orchestration (as production executor)

Profile 6: @COST - Budget-Conscious Persona (Optional)

User Role: Cost optimization, high-volume queries, development/testing
LLM Provider: Gemini 1.5 Flash
Cost Profile: Minimal ($0.075 per 1M input tokens - cheapest available)
Focus: Maximum cost efficiency

MCP Tools Available:
- Same as @PROD (all standard tools)

Knowledge Collections Attached:
- Minimal collections (reduces RAG retrieval overhead)
- "Essential Reference Only" (3 documents)

Planner Repositories Attached:
- "Fast_Path_Patterns" (56 cases) - Optimized for speed and low token usage
- "Simple_Query_Templates" (42 cases) - Straightforward, single-phase executions

```
**Primary Use Cases:**
- High-frequency simple queries
- Development and testing environments
- Batch jobs with simple logic
- Proof-of-concept demonstrations
- Training and learning scenarios

**Behavioral Characteristics:**
- Prefers fast-path execution (bypasses strategic planning when possible)
- Minimizes token usage (concise outputs)
- Leverages RAG heavily (reuses patterns instead of replanning)
- Avoids complex multi-phase strategies

**Tutorial Appearances:**
- Module 4.4: Cost tracking and optimization
- Module 7.2: Financial governance (cost comparison)
- Module 10.5: "God mode" orchestration (optional cost-conscious execution)

---

## Module 10: User Journey Scenarios
**Goal:** Demonstrate end-to-end workflows mapping personas to real-world user stories

### 10.1 Business Analyst Journey - @ANALYST Profile
**Duration:** 4-5 minutes
**Persona:** Sarah, Product Manager preparing Q4 board presentation
**Profile Used:** Primarily @ANALYST, brief @QUALITY validation

**Learning Objectives:**
- Follow complete exploratory data analysis workflow
- See conversational iteration and refinement
- Understand context retention across multi-turn sessions

**Scenario Narrative:**
Sarah needs to present Q4 fitness equipment sales performance to the board. She has 2 hours to

**Key Topics to Cover:**
- [ ] Initial question: "What are Q4 revenue trends by product category?"
- [ ] Drill-down: "Why did Cardio equipment decline 18%?"
- [ ] Customer analysis: Demographics of churned Cardio buyers
- [ ] Validation: Switch to @QUALITY for data completeness
- [ ] Final synthesis: Executive summary with recommendations

**Narration Script:**
```

Meet Sarah. Product Manager. Board presentation in 2 hours.

Topic: Q4 fitness equipment sales performance.

Traditional approach: Email data team → Wait 4 hours → Generic report → Still have questions → Email again → Miss deadline.

Uderia approach: Ask. Refine. Visualize. Present. Done.

THE WORKFLOW:

Sarah logs into Uderia. Profile selector shows 6 options. Selects **@ANALYST** (Gemini Flash + business tools).

Chat opens. Profile indicator: "@ANALYST Active."

QUERY 1: EXPLORATORY

Sarah types: *"What are Q4 2024 revenue trends by product category?"*

@ANALYST activates:

```
STRATEGIC PLANNING:
  Tools needed: base_readQuery, TDA_Charting
  Champion case: "Trend_Analysis_Patterns" (31 cases)
  Knowledge: "Business Metrics Definitions"
```

Transparent execution:

```
 Tool: base_readQuery
  Query: SELECT ProductType, SUM(TotalAmount) as Revenue
         FROM Sales s JOIN Products p ON s.ProductID = p.ProductID
         WHERE SaleDate >= '2024-10-01' AND SaleDate <= '2024-12-31'
         GROUP BY ProductType

 Tool: TDA_Charting (Bar Chart)
  Categories: Cardio ($287K), Strength ($456K), Accessories ($149K)
```

Response:

```
Q4 2024 REVENUE BY CATEGORY:

[Bar chart visualization]
• Strength Equipment: $456,230 (51%, +23% vs Q3)
• Cardio Equipment: $287,450 (32%, -18% vs Q3)
• Accessories: $149,320 (17%, +8% vs Q3)

Total Q4 Revenue: $892,450

Key Insight: Cardio equipment declined 18% QoQ. Requires investigation.
**QUERY 2: DRILL-DOWN**

Sarah: *"Why did Cardio equipment decline 18%?"*

Context retained. @ANALYST executes 3 queries:
```

 Tool: base_readQuery • Q4 Cardio sales by month: Oct $112K → Nov $95K → Dec $80K (progressive decline) • Customer purchase frequency: -34% repeat buyers • Service ticket correlation:

+47% complaints about treadmill reliability

Response: Progressive monthly decline linked to product quality issues.

CARDIO EQUIPMENT DECLINE ANALYSIS:

[Line graph showing monthly decline]

ROOT CAUSE INDICATORS: 1. Progressive monthly decline: Oct→Nov→Dec (not one-time event) 2. Customer retention drop: 67% of Q3 Cardio buyers did NOT repurchase 3. Average transaction value stable: $1,245 (not pricing issue) 4. New customer acquisition normal: +12% new buyers

Hypothesis: Existing customer churn, not acquisition problem.

Sarah's hypothesis: It's a retention problem.ition problem.

QUERY 3: CUSTOMER ANALYSIS

Sarah: *"Show demographics of customers who stopped buying Cardio in Q4"*

 Tool: base_readQuery JOIN: Sales → Customers → Products Filter: Purchased Cardio in Q3 BUT NOT in Q4 Result: 127 churned Cardio customers identified

Response:

CHURNED CARDIO CUSTOMER PROFILE (n=127):

Demographics: • Age: 78% are 35-54 (prime fitness demographic) • Location: 62% in urban areas (Seattle, Portland, SF) • Avg Lifetime Value: $3,450 (valuable customers!) • Purchase History: Average 3.2 prior Cardio purchases (loyal)

Concern: Losing high-value, loyal customers in key markets.

Pattern emerging. Sarah needs one more piece.mers in key markets.

Pattern emerging. But Sarah needs one more piece.

QUERY 4: CORRELATION ANALYSIS

Sarah: *"Is there a relationship between service tickets and Cardio sales decline?"*

Strategic Planning retrieves case: "Service Impact on Revenue Analysis"

 Tool: base_readQuery JOIN: ServiceTickets → Sales → Products Analysis: Ticket volume vs sales by product type

 Tool: TDA_LLMTask Correlation: Cardio service tickets +34% in Q4

Response:

SERVICE TICKET CORRELATION:

Product Type | Q4 Service Tickets | Q4 Revenue Change
Strength | 23 (+5%) | +23% □ Accessories | 12 (-8%) | +8% □ Cardio | 89 (+34%) □ | -18% □

Critical Finding: Cardio had 34% MORE service tickets in Q4.

Top Issues: 1. Treadmill belt malfunction (23 tickets) 2. Display screen failure (18 tickets) 3. Motor overheating (15 tickets)

Insight: Quality issues driving churn. Customers stopped buying after poor service experiences.

Sarah has her story. Now needs validation.

QUICK VALIDATION: Switch to @QUALITY

Profile dropdown → Select **@QUALITY**

Capabilities reload (Claude Sonnet + validation tools).

Sarah: *"Validate completeness of Q4 sales data for Cardio products"*

□ Tool: qlty_profileColumn Table: Sales Columns: TotalAmount, SaleDate, ProductID

□ Completeness: 100% (no nulls) □ Referential Integrity: All ProductIDs valid □ Date Range: Oct 1 - Dec 31 complete □ Amount Distribution: Normal (no outliers)

Response: **Data Quality Score: 99.7/100**

"Q4 Cardio sales data fully validated. Analysis conclusions are reliable."

Data validated. Back to synthesis.

RETURN TO @ANALYST: Final Report

Profile dropdown → Back to **@ANALYST**

Sarah: *"Generate executive summary with recommendations"*

Strategic Planning activates "Executive_Reporting_Patterns" repository (23 champion cases).

Knowledge retrieval: "Executive Reporting Templates" collection.

□ Tool: TDA_FinalReport Format: Executive Summary Audience: Board of Directors Length: 1-page, 3-minute read

Response:

EXECUTIVE SUMMARY: Q4 2024 Fitness Equipment Sales
Prepared by: Sarah Mitchell, Product Manager
Date: December 7, 2024

OVERALL PERFORMANCE
Q4 total revenue: **$892,450** (+8% vs Q3, -5% vs target)

KEY FINDINGS

 Strength Equipment: $456K, +23% growth (exceeding expectations)
 Accessories: $149K, +8% growth (on track)
 Cardio Equipment: $287K, -18% decline (critical concern)

ROOT CAUSE ANALYSIS: CARDIO DECLINE

The Problem:
- 127 high-value customers (avg $3,450 LTV) stopped purchasing Cardio equipment in Q4
- 78% are prime demographic (age 35-54) in key urban markets

The Cause:
- Service tickets for Cardio equipment increased 34% (89 tickets)
- Top issues: Treadmill malfunctions (23), display failures (18), motor problems (15)
- Customer sentiment: Quality concerns driving churn

RECOMMENDATIONS

1. **Immediate:** Supplier quality audit for Cardio equipment (Est. 2 weeks)
2. **Short-term:** Proactive outreach to 127 churned customers with service credits ($50K budg
3. **Long-term:** Enhanced quality testing before product launches

FINANCIAL IMPACT
- Lost revenue from churned customers: ~$438K annually
- Recovery potential with retention campaign: ~$280K (64% recovery rate based on historical da
- **Net ROI:** $230K revenue recovery for $50K investment (4.6x return)

BOARD ASK
Approve $50K retention campaign budget to recover high-value Cardio customers.

Sarah: *Perfect.*

EXPORT & SHARE

```
Click "Export" → Download PDF

Email to board members.

1 hour 45 minutes total. 15 minutes to spare.

**WHAT JUST HAPPENED:**

Traditional approach:
- Data team request: 4 hours
- Generic report received
- Still need answers
- Second request: 4 more hours
- Manually create slides: 2 hours
- **Total: 10+ hours, missed deadline**

Uderia approach with @ANALYST:
- Exploratory query: 30 seconds
- Drill-down analysis: 45 seconds
- Customer demographics: 35 seconds
- Correlation analysis: 40 seconds
- Switch to @QUALITY validation: 20 seconds
- Executive summary generation: 50 seconds
- **Total: 1 hour 45 minutes (including thinking time)**

**THE POWER:**
- **No SQL writing** (business questions in plain English)
- **Context retention** (each query built on previous answers)
- **Automatic visualizations** (charts generated without manual work)
- **Knowledge-enhanced** (champion cases + business context)
- **Multi-profile validation** (@QUALITY confirmed data reliability)
- **Board-ready output** (executive summary formatted automatically)

Sarah went from "question" to "boardroom-ready presentation" in under 2 hours.

One person. One profile. Zero data team bottleneck.

This is @ANALYST. This is Uderia. This is the future of business intelligence.
```

Screen Capture Plan: - [ ] Profile indicator: @ANALYST active - [ ] Initial query with revenue trends - [ ] TDA_Charting generates bar chart visualization - [ ] Follow-up questions building context - [ ] Switch to @QUALITY briefly for validation - [ ] Return to @ANALYST for final synthesis - [ ] TDA_FinalReport creates executive summary - [ ] Export/download functionality

Supporting Materials: - Complete Q4 sales dataset - Expected visualizations (mockups) - Executive summary template

---

10.2 Database Administrator Journey - @DBA Profile

Duration: 4-5 minutes
Persona: Marcus, DBA investigating slow query performance
Profile Used: Primarily @DBA, switch to @PROD for API testing

Learning Objectives: - Demonstrate technical SQL optimization workflow - Show schema analysis and profiling - Illustrate performance tuning process

Scenario Narrative: Marcus received complaints that the “Monthly Sales Report” query is taking 45 seconds. He needs to optimize it and deploy the fix.

Key Topics to Cover: - [ ] Analyze execution plan for performance bottleneck - [ ] Schema inspection and index analysis - [ ] Generate optimized SQL with proper indexing - [ ] Benchmark: Compare old vs new execution time (45s → 2.1s) - [ ] Document optimization and deploy via API

Narration Script:

```
Meet Marcus. Database Administrator. 15 years tuning SQL.

Just got paged: "Monthly Sales Report taking 45 seconds. Users complaining."

The report runs 200 times per day. 45 seconds × 200 = 2.5 hours of cumulative wait time daily.

Traditional approach: Request query → Analyze execution plan manually → Research index strategy

Uderia approach: Ask. Analyze. Optimize. Validate. Deploy. Done.

**THE INVESTIGATION:**

Marcus logs into Uderia. Profile selector: **@DBA** (Claude Sonnet + database tools).

Chat opens. Profile indicator: "@DBA Active."

**QUERY 1: ANALYZE CURRENT PERFORMANCE**

Marcus types: *"Show me the execution plan and performance for the monthly sales aggregation qu

He pastes the problematic query:

```sql
SELECT
    p.ProductName,
    COUNT(*) as SalesCount,
    SUM(s.TotalAmount) as TotalRevenue,
    AVG(s.TotalAmount) as AvgSaleAmount
FROM Sales s
JOIN Products p ON s.ProductID = p.ProductID
WHERE s.SaleDate >= CURRENT_DATE - INTERVAL '30' DAY
GROUP BY p.ProductName
```
```

```
ORDER BY TotalRevenue DESC;
```

@DBA activates:

```
STRATEGIC PLANNING:
  Champion case: "Performance_Troubleshooting_Playbooks" (22 cases)
  Tool: base_readQuery with EXPLAIN
```

Execution:

```
 Tool: base_readQuery (EXPLAIN mode)
  Analyzing query execution plan...

 PERFORMANCE BOTTLENECK DETECTED:
   • Full table scan on Sales (237,450 rows)
   • No index on SaleDate column
   • JOIN operation: 237K × 1,285 rows = 305M comparisons
   • Execution time: 45.3 seconds
```

Response:

```
PERFORMANCE ANALYSIS: Monthly Sales Report

Current Performance: 45.3 seconds

Execution Plan Issues:
1. Full table scan on Sales table (237,450 rows scanned)
2. No index utilized for SaleDate filter
3. Inefficient JOIN (nested loop with 305M row comparisons)
4. Temporary sort for ORDER BY (67MB memory usage)

Impact:
• Query runs 200x daily
• Cumulative wait time: 2.5 hours/day
• Database load: 15% CPU usage during each execution

Recommendation: Index creation + query restructuring required.
```

QUERY 2: SCHEMA & OPTIMIZATION

Marcus: *"Show DDL and generate optimized version"*

```
 Tool: base_tableDDL → Analyzes schema
 Tool: TDA_LLMTask → Generates optimized SQL with index recommendation
```

Optimization:

```sql
-- Add index
CREATE INDEX idx_sales_date ON Sales(SaleDate);

-- Optimized query with BETWEEN and covering index
    TotalAmount DECIMAL(10,2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
```

```
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

-- Existing Indexes:
PRIMARY KEY idx_sales_pk ON (SaleID)
--   NO index on SaleDate
--   NO index on ProductID (FK not indexed)

CRITICAL FINDINGS:
1. Missing index on SaleDate (filter column)
2. Missing index on ProductID (JOIN column, despite being FK)
3. Primary key only (SaleID not used in this query)

Schema Issue Confirmed: Date filters and JOINs have no index support.
```

Marcus needs data distribution insights before recommending indexes.

QUERY 3: DATA PROFILING

Marcus: *"Profile the SaleDate and ProductID columns to understand data distribution and cardinality"*

```
 Tool: qlty_profileColumn
   Table: Sales | Columns: SaleDate, ProductID
   Analysis: Distribution, cardinality, selectivity
```

Response:

```
COLUMN PROFILING RESULTS:

SaleDate Column:
• Date range: 2022-01-15 to 2024-12-06 (1,056 days)
• Distribution: Relatively uniform (220-240 sales per day)
• Last 30 days: 6,847 rows (2.9% of total)
• Selectivity: High (good index candidate)

ProductID Column:
• Distinct values: 1,285 products
• Distribution: Skewed (top 20 products = 45% of sales)
• Average rows per ProductID: 185
• Selectivity: Moderate (JOIN benefits from index)

INDEX STRATEGY RECOMMENDATION:
1. Composite index: (SaleDate, ProductID) for optimal performance
2. Alternative: Separate indexes if storage constrained
3. Estimated selectivity improvement: 97.1% (access 2.9% of rows, not 100%)
```

Data supports the index strategy. Now: optimization.

QUERY 4: GENERATE OPTIMIZED QUERY QUERY 4: GENERATE OPTIMIZED QUERY

Marcus: *"Rewrite this query with optimal index usage and query structure for Teradata"*

```
STRATEGIC PLANNING:
  Repository: "DBA_Champion_Cases" (47 cases)
  Knowledge: "Teradata SQL Optimization Guide"

  Tool: TDA_LLMTask
   Task: SQL optimization with index hints
   Context: Teradata-specific (COLLECT STATISTICS, index strategy)
```

Response:

```
-- Step 1: Create optimal indexes (run ONCE)
CREATE INDEX idx_sales_date_product ON Sales (SaleDate, ProductID);
COLLECT STATISTICS ON Sales COLUMN (SaleDate, ProductID);

-- Step 2: Optimized query structure
SELECT
    p.ProductName,
    COUNT(*) as SalesCount,
    SUM(s.TotalAmount) as TotalRevenue,
    AVG(s.TotalAmount) as AvgSaleAmount
FROM Sales s
INNER JOIN Products p ON s.ProductID = p.ProductID
WHERE s.SaleDate >= CURRENT_DATE - INTERVAL '30' DAY
GROUP BY p.ProductName
ORDER BY TotalRevenue DESC;

OPTIMIZATION TECHNIQUES APPLIED:
  Composite index on (SaleDate, ProductID)
  COLLECT STATISTICS for query optimizer
  INNER JOIN explicit (same result, clearer intent)
  Filter-first strategy (date filter before JOIN reduces rows)

Expected Performance: 2-3 seconds (93% improvement)
```

Marcus needs to validate before deploying. **QUERY 5: BENCHMARK COMPARISON**

Marcus: *"Execute both versions and compare performance"*

```
**QUERY 5: BENCHMARK COMPARISON**

Marcus: *"Execute both versions and compare performance"*
```

□ Tool: base_readQuery (with timing) Test 1: Original query Test 2: Optimized query (after index creation)

```
Response:
```

PERFORMANCE BENCHMARK RESULTS:

Metric | Original | Optimized | Improvement
Execution Time | 45.3s | 3.2s | 93.0% □ Rows Scanned | 237,450 | 6,847 | 97.1% □ Memory Used

| 67 MB | 4 MB | 94.0% □ CPU Usage | 15% | 1.2% | 92.0% □

DAILY IMPACT: • Before: 200 queries × 45.3s = 2.5 hours cumulative wait time • After: 200 queries × 3.2s = 10.7 minutes cumulative wait time • Time saved: 2 hours 19 minutes daily per user • Database load reduction: 92% less CPU usage

Validation: Optimized query returns identical results □

```
Validated. Now: operationalize for production.
Capabilities reload (production automation tools).

Marcus: *"Generate REST API call for this optimized query with JSON response format"*
```

□ Tool: TDA_FinalReport Format: REST API documentation Profile: @PROD (production execution)

```
Response:

**REST API Implementation**

**Endpoint:** POST `/api/v1/query`

**Request:**
```json
{
  "profile": "@PROD",
  "query": "Execute optimized monthly sales report",
  "sql": "SELECT p.ProductName, COUNT(*) as SalesCount, SUM(s.TotalAmount) as TotalRevenue, AV
  "format": "json"
}
```

Response (3.2s):

```
{
  "status": "complete",
  "execution_time": "3.2s",
  "rows": 1285,
  "data": [
    {"ProductName": "ProRunner 3000", "SalesCount": 234, "TotalRevenue": 291450.00, "AvgSaleAm
    {"ProductName": "FlexBench Pro", "SalesCount": 189, "TotalRevenue": 245678.00, "AvgSaleAmou
    ...
  ]
}
```

Python Client Example:

```
import requests

response = requests.post(
    'https://uderia.company.com/api/v1/query',
```
```

```
    headers={'Authorization': 'Bearer uderia_sk_live_...'},
    json={
        'profile': '@PROD',
        'sql': optimized_query,
        'format': 'json'
    }
)

results = response.json()
print(f"Execution: {results['execution_time']}")  # 3.2s
```

QUERY 6: AIRFLOW DAG UPDATE

Marcus: *"Generate Airflow DAG code to run this optimized report daily at 6 AM"*

```
Tool: TDA_FinalReport
  Format: Python/Airflow DAG
  Schedule: Daily 6 AM
```

Response:

Airflow DAG: Monthly Sales Report (Optimized)

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta
import requests

def run_monthly_sales_report():
    """Execute optimized sales report via Uderia API"""

    response = requests.post(
        'https://uderia.company.com/api/v1/query',
        headers={'Authorization': f'Bearer {os.environ["UDERIA_TOKEN"]}'},
        json={
            'profile': '@PROD',
            'sql': '''
                SELECT p.ProductName, COUNT(*) as SalesCount,
                       SUM(s.TotalAmount) as TotalRevenue,
                       AVG(s.TotalAmount) as AvgSaleAmount
                FROM Sales s
                INNER JOIN Products p ON s.ProductID = p.ProductID
                WHERE s.SaleDate >= CURRENT_DATE - INTERVAL '30' DAY
                GROUP BY p.ProductName
                ORDER BY TotalRevenue DESC;
            ''',
            'format': 'json'
        }
    )
```

```
    if response.status_code == 200:
        results = response.json()
        print(f"  Report generated: {results['rows']} products")
        print(f"  Execution time: {results['execution_time']}")
        # Save to S3, send email, etc.
    else:
        raise Exception(f"Report failed: {response.text}")

default_args = {
    'owner': 'marcus',
    'retries': 2,
    'retry_delay': timedelta(minutes=5)
}

dag = DAG(
    'monthly_sales_report_optimized',
    default_args=default_args,
    description='Optimized monthly sales report (3.2s vs 45s)',
    schedule_interval='0 6 * * *',  # Daily at 6 AM
    start_date=datetime(2024, 12, 7),
    catchup=False
)

report_task = PythonOperator(
    task_id='generate_monthly_report',
    python_callable=run_monthly_sales_report,
    dag=dag
)
```

Deployment: 1. Save as `dags/monthly_sales_report_optimized.py` 2. Deploy to Airflow 3. Verify execution in Airflow UI 4. Monitor performance (should see 3.2s execution time)

FINAL DOCUMENTATION

Marcus: *"Generate technical documentation for this optimization"*

```
Tool: TDA_FinalReport
  Format: Technical documentation
  Audience: DBA team
```

Response:

TECHNICAL DOCUMENTATION: Monthly Sales Report Optimization *Author: Marcus Chen, DBA Date: December 7, 2024*

PROBLEM Monthly sales aggregation query executing in 45.3 seconds, causing user complaints and database load.

ROOT CAUSE 1. Full table scan on Sales table (237,450 rows) 2. No index on SaleDate (date range filter column) 3. No index on ProductID (JOIN column) 4. Inefficient nested loop JOIN

SOLUTION Created composite index: `idx_sales_date_product ON Sales (SaleDate, ProductID)`

SQL CHANGES

```
-- Index creation
CREATE INDEX idx_sales_date_product ON Sales (SaleDate, ProductID);
COLLECT STATISTICS ON Sales COLUMN (SaleDate, ProductID);

-- Query unchanged (optimizer now uses index automatically)
```

PERFORMANCE IMPROVEMENT - Execution time: 45.3s → 3.2s (93% faster) - Rows scanned: 237K → 6.8K (97% reduction) - Daily time saved: 2h 19m across 200 executions - CPU usage: 15% → 1.2% per execution

DEPLOYMENT - Index created: 2024-12-07 14:23 UTC - Statistics collected: 2024-12-07 14:25 UTC - Airflow DAG updated: `monthly_sales_report_optimized.py` - First production run: 2024-12-08 06:00 UTC

VALIDATION - Query results identical before/after □ - Production API tested: 3.2s execution □ - No regressions on other queries □

MAINTENANCE - Monitor index size (currently 12 MB) - Re-collect statistics monthly: `COLLECT STATISTICS ON Sales COLUMN (SaleDate, ProductID);` - Review query plan quarterly

---

Done. Marcus closes laptop.

WHAT JUST HAPPENED:

```
TRADITIONAL DBA WORKFLOW:
1. Get slow query from user → 15 min
2. Manually run EXPLAIN → 10 min
3. Research index strategy → 30 min
4. Check table DDL in schema docs → 15 min
5. Profile data distribution manually → 20 min
6. Design optimal index → 20 min
7. Test in dev environment → 30 min
8. Write deployment script → 15 min
9. Update Airflow DAG → 20 min
10. Write documentation → 30 min
Total: 3+ hours

UDERIA @DBA WORKFLOW:
1. Paste query, ask for analysis → 45 sec
2. Request DDL inspection → 30 sec
3. Profile data distribution → 35 sec
4. Generate optimized query → 50 sec
5. Benchmark performance → 40 sec
```

```
6. Switch to @PROD, get API code → 30 sec
7. Generate Airflow DAG → 45 sec
8. Generate documentation → 40 sec
Total: 25 minutes
```

THE POWER: - No manual EXPLAIN plans (automated execution plan analysis) - Automatic DDL retrieval (base_tableDDL tool) - AI-driven profiling (qlty_profileColumn with interpretation) - Context-aware optimization (champion cases from 47 DBA patterns) - Knowledge-enhanced (Teradata SQL optimization guide) - Production-ready output (REST API + Airflow DAG + documentation)

OUTCOME: 93% faster query. 88% faster resolution. Production-deployed.

Marcus went from “performance complaint” to “optimized, documented, and deployed” in 25 minutes.

One DBA. One profile. Zero manual execution plan reading.

This is @DBA. This is Uderia. This is database administration evolved.

```
**SCREEN CAPTURE PLAN:**
- Profile indicator: @DBA active
- Query analysis with `base_readQuery` showing slow performance
- `base_tableDDL` revealing table structure
- `qlty_profileColumn` showing data distribution
- Optimized SQL generation with index hints
- Performance comparison: 45s → 3.2s
- Switch to @PROD profile
- REST API call with optimized query
- Airflow DAG update

**SUPPORTING MATERIALS:**
- Sample slow query
- Optimized query comparison
- Performance metrics

---

### 10.3 Data Quality Analyst Journey - @QUALITY Profile
**Duration:** 4-5 minutes
**Persona:** Priya, Data Steward conducting monthly quality audit
**Profile Used:** @QUALITY, brief @DBA for schema context

**Learning Objectives:**
- Systematic data quality workflow
- Validation and profiling capabilities
- Anomaly detection
- Compliance-ready quality report
```

Scenario:
Priya performs monthly data quality audits for the fitness_db database. Must validate constrai

Key Topics:
- Comprehensive quality assessment
- Constraint validation: referential integrity, null constraints
- Column profiling with `qlty_profileColumn`
- Anomaly detection in Sales.TotalAmount
- Business rule validation: SaleDetails.UnitPrice vs Products.Price
- Switch to @DBA for schema context
- Quality score and remediation recommendations
- Compliance audit report

Narration Script:

Meet Priya. Data Quality Analyst. Guardian of data integrity.

Monthly audit due. fitness_db database. 5 tables. 250,000+ rows.

Traditional approach: Write validation queries → Export to Excel → Check constraints → Flag anomalies → Write report → 6 hours.

Uderia approach: Ask. Validate. Profile. Detect. Report. Done.

THE AUDIT:

Priya logs into Uderia. Selects **@QUALITY** (Claude Sonnet + quality tools).

Chat opens. Profile indicator: "@QUALITY Active."

QUERY 1: COMPREHENSIVE QUALITY ASSESSMENT

Priya types: *"Perform comprehensive quality assessment on fitness_db database"*

@QUALITY activates:

```
STRATEGIC PLANNING:
  Repository: "Quality_Validation_Patterns" (29 champion cases)
  Knowledge: "Data Quality Standards" (20 documents)
```

Execution:

```
  Tool: base_tableList
    Database: fitness_db
    Tables: Customers, Products, Sales, SaleDetails, ServiceTickets

  Tool: qlty_validateData
    Running constraint checks across all tables...

  Validation Progress:
    • NULL constraint validation
    • Referential integrity checks
    • Data type consistency
    • Value range validation
```

Response:

```
DATA QUALITY ASSESSMENT: fitness_db

Overall Quality Score: 97.3/100

Tables Audited: 5
Total Rows: 252,847
Validation Rules Applied: 47

**Quality Breakdown:**

| Table | Rows | Completeness | Integrity | Accuracy | Score |
|-------|------|--------------|-----------|----------|-------|
| Customers | 8,945 | 100% | 100% | 99.2% | 99.7/100 |
| Products | 1,285 | 100% | 100% | 100% | 100/100 |
| Sales | 237,450 | 99.8% | 99.9% | 98.1% | 97.9/100 |
| SaleDetails | 4,892 | 100% | 99.7% | 97.8% | 98.5/100 |
| ServiceTickets | 275 | 100% | 100% | 92.4% | 97.5/100 |

**Issues Detected:** 6 (3 critical, 2 warnings, 1 info)

---

Good overall score. But those issues need investigation.

**QUERY 2: CONSTRAINT VALIDATION DETAILS**

Priya: *"Show me details on the constraint violations found"*
```

 Tool: qlty_validateData (detailed mode) Focus: Referential integrity, NULL constraints

```
Response:

**Constraint Violation Details**

**CRITICAL ISSUES (3):**

**1. Orphaned Sales Records**
- **Table:** Sales
- **Issue:** 437 rows with CustomerID not in Customers table
- **Impact:** Referential integrity violated
- **Sample IDs:** 182547, 189234, 195678 (CustomerID: 9999, 9998, 9997)
- **Root Cause:** Test data not cleaned after migration
- **Remediation:** DELETE or UPDATE with valid CustomerIDs

**2. SaleDetails Price Mismatch**
- **Table:** SaleDetails
```

```
- **Issue:** 12 rows where UnitPrice  Products.Price
- **Impact:** Revenue calculation discrepancies
- **Sample:** SaleDetailID 4523 (UnitPrice: $1,299 vs Products.Price: $1,199)
- **Potential Loss:** $876 revenue misreported
- **Remediation:** Verify pricing rules, correct discrepancies

**3. NULL TotalAmount**
- **Table:** Sales
- **Issue:** 5 rows with NULL TotalAmount (should be NOT NULL)
- **Impact:** Revenue reporting incomplete
- **SaleIDs:** 234789, 234790, 234812, 235001, 235002
- **Date Range:** December 5-6, 2024 (data entry error?)
- **Remediation:** Calculate and populate from SaleDetails

**WARNINGS (2):**

**4. Duplicate Customer Emails**
- **Table:** Customers
- **Issue:** 18 email addresses used by multiple CustomerIDs
- **Example:** john.smith@email.com (CustomerID: 1234, 5678)
- **Impact:** Potential data quality issue or valid shared accounts
- **Recommendation:** Review and consolidate or flag as shared

**5. Future ServiceTicket Dates**
- **Table:** ServiceTickets
- **Issue:** 3 tickets dated in future (TicketDate > CURRENT_DATE)
- **IDs:** 256, 267, 271 (dates: Dec 15-18, 2024)
- **Impact:** Data entry error or system clock issue
- **Remediation:** Correct dates to valid values

**INFO (1):**

**6. Low Stock Products**
- **Table:** Products
- **Issue:** 23 products with StockQuantity < 10
- **Impact:** Potential stockout risk (not a data quality issue)
- **Recommendation:** Inventory replenishment (business process)

---

Priya now has specific issues to investigate. Time for detailed profiling.

**QUERY 3: COLUMN PROFILING**

Priya: *"Profile the Sales.TotalAmount column to identify outliers and distribution patterns"*
```

 Tool: qlty_profileColumn Table: Sales Column: TotalAmount Analysis: Distribution, outliers, statistical summary

Response:

Column Profile: Sales.TotalAmount

Statistical Summary:
- **Row Count:** 237,450 (excluding 5 NULLs)
- **Min:** $15.99
- **Max:** $18,450.00
- **Mean:** $875.34
- **Median:** $624.50
- **Std Dev:** $456.23

Distribution:

$0-500: ████████████████████████ 45.2% (107,321 rows) $501-1000: ███████████ 28.7% (68,128 rows) $1001-2000: ████████ 18.3% (43,453 rows) $2001-5000: ███ 6.9% (16,384 rows) $5000+: ▌ 0.9% (2,164 rows)

Outliers Detected (3):

SaleID	TotalAmount	Date	ProductID	Status
189456	**$18,450**	Nov 23	1047	21x standard deviation
201234	**$16,890**	Dec 1	1047	19x standard deviation
198765	**$0.01**	Nov 30	523	Test transaction?

Analysis:
- **High outliers (2):** ProductID 1047 = "Commercial Gym Bundle" ($18K legitimate)
- **Low outlier (1):** $0.01 likely test/error transaction
- **Recommendation:** Verify $0.01 transaction, flag commercial bundles separately

---

Outliers explained. Now: business rule validation.

QUERY 4: BUSINESS RULE VALIDATION

Priya: *"Verify that SaleDetails.UnitPrice matches Products.Price for all sales transactions"*

 Tool: base_readQuery Query: JOIN SaleDetails → Products Validation: UnitPrice consistency check

Response:

Business Rule Validation: Price Consistency

Rule: SaleDetails.UnitPrice MUST equal Products.Price at time of sale

```
**Validation Results:**
- **Total SaleDetails Records:** 4,892
- **Matching Prices:** 4,880 (99.8%)
- **Price Mismatches:** 12 (0.2%)

**Mismatches Identified:**

| SaleDetailID | ProductName | Expected | Actual | Diff | SaleDate |
|--------------|-------------|----------|--------|------|----------|
| 4523 | ProRunner 3000 | $1,199 | $1,299 | +$100 | Dec 3 |
| 4534 | FlexBench Pro | $899 | $799 | -$100 | Dec 3 |
| 4567 | Yoga Mat Premium | $49 | $59 | +$10 | Dec 4 |
| 4589 | Resistance Bands | $29 | $19 | -$10 | Dec 4 |
| ... | (8 more) | ... | ... | ... | ... |

**Total Revenue Impact:** +$876 (net overcharge)

**Root Cause Hypothesis:**
- All mismatches in Dec 3-4 (same date range)
- Pattern suggests manual price override during promotion
- Promotion pricing not reflected in Products table

**Remediation:**
1. **Immediate:** Correct SaleDetails records or Products.Price
2. **Process:** Implement promotion price tracking
3. **Audit:** Review all December promotional sales

---

Priya needs context on table relationships. Quick switch to @DBA.

**BRIEF SWITCH TO @DBA**

Profile dropdown → **@DBA**

Priya: *"Show me the DDL for SaleDetails table to understand constraints and relationships"*
```

 Tool: base_tableDDL Table: SaleDetails

```
Response:

```sql
CREATE TABLE SaleDetails (
    SaleDetailID INTEGER PRIMARY KEY,
    SaleID INTEGER NOT NULL,
    ProductID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL CHECK (Quantity > 0),
```
```

```
    UnitPrice DECIMAL(10,2) NOT NULL CHECK (UnitPrice >= 0),
    FOREIGN KEY (SaleID) REFERENCES Sales(SaleID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

Constraints: Primary Key: SaleDetailID Foreign Keys: SaleID → Sales, ProductID → Products NOT NULL: All critical fields **Missing:** UnitPrice vs Products.Price validation constraint

Recommendation: Add CHECK constraint or trigger to enforce price consistency.

Context obtained. Back to quality report.

RETURN TO @QUALITY

Profile dropdown → **@QUALITY**

Priya: *"Generate monthly data quality audit report with all findings and remediation recommendations"*

Strategic Planning: "Data_Profiling_Templates" repository (24 champion cases)

Knowledge: "Industry Data Validation Best Practices" collection

```
Tool: TDA_FinalReport
  Format: Compliance audit report
  Audience: Data Governance Committee
```

Response:

MONTHLY DATA QUALITY AUDIT REPORT *fitness_db Database Audit Period: December 2024 Auditor: Priya Sharma, Data Quality Analyst Date: December 7, 2024*

EXECUTIVE SUMMARY

Overall Quality Score: 97.3/100 (Target: ≥95%)

fitness_db database quality meets organizational standards with minor issues requiring remediation.

Audit Scope: - **Tables:** 5 (Customers, Products, Sales, SaleDetails, ServiceTickets) - **Total Records:** 252,847 - **Validation Rules:** 47 automated checks - **Execution Time:** 4 minutes 23 seconds

KEY FINDINGS

 STRENGTHS 1. **High completeness:** 99.9% of required fields populated 2. **Strong referential integrity:** 99.8% FK relationships valid 3. **Consistent data types:** 100% type conformance 4. **Products table:** Perfect quality score (100/100)

 CRITICAL ISSUES (3)

Issue 1: Orphaned Sales Records - **Count:** 437 records (0.18% of Sales table) - **Impact:** Revenue attribution impossible, reporting inaccurate - **Severity:** HIGH - **Remediation:** DELETE

invalid records or reconcile CustomerIDs - **Owner:** Database Administration - **Deadline:** December 15, 2024

Issue 2: Price Mismatches (SaleDetails vs Products) □ - **Count:** 12 records (0.25% of SaleDetails) - **Financial Impact:** $876 net revenue discrepancy - **Severity:** HIGH - **Remediation:** Implement promotional pricing tracking - **Owner:** Finance & IT - **Deadline:** December 20, 2024

Issue 3: NULL Revenue Values □ - **Count:** 5 Sales records with NULL TotalAmount - **Impact:** Incomplete Q4 revenue reporting - **Severity:** CRITICAL - **Remediation:** Calculate and populate from SaleDetails immediately - **Owner:** Data Engineering - **Deadline:** December 8, 2024 (within 24 hours)

□ **WARNINGS (2)**

Warning 1: Duplicate Customer Emails □ - **Count:** 18 email addresses, 36+ customer records - **Risk:** Marketing communications, account merges - **Recommendation:** Review for consolidation - **Priority:** MEDIUM

Warning 2: Future-Dated Service Tickets □ - **Count:** 3 records with dates > CURRENT_DATE - **Likely Cause:** Data entry error or system clock issue - **Recommendation:** Correct dates to valid values - **Priority:** LOW

OUTLIER ANALYSIS

Sales.TotalAmount Outliers: - 2 legitimate commercial sales ($18,450 and $16,890) - 1 test transaction ($0.01) - recommend deletion

DATA QUALITY SCORECARD

Dimension	Score	Status	Notes
Completeness	99.8%	□ PASS	5 NULL values in Sales.TotalAmount
Accuracy	98.1%	□ PASS	12 price mismatches, 437 orphaned records
Consistency	99.5%	□ PASS	Minor date inconsistencies
Integrity	99.8%	□ PASS	0.2% referential integrity issues
Timeliness	100%	□ PASS	All data current
Uniqueness	99.3%	□ PASS	18 duplicate emails (under review)

REMEDIATION ROADMAP

Week 1 (Dec 8-14): 1. □ Populate NULL TotalAmount values (CRITICAL) 2. □ Delete/correct test transaction ($0.01) 3. □ Correct future-dated ServiceTickets

Week 2 (Dec 15-21): 4. □ Resolve orphaned Sales records 5. □ Implement promotional pricing tracking 6. □ Correct SaleDetails price mismatches

Week 3 (Dec 22-28): 7. □ Review duplicate customer emails 8. □ Add database constraint for UnitPrice validation 9. □ Update data quality monitoring rules

COMPLIANCE STATUS

□ **SOC 2 Requirements:** COMPLIANT (≥95% quality score) □ **ISO 8000 Standards:** COMPLIANT (data accuracy >98%) □ **Internal SLA:** COMPLIANT (monthly audit completed on time)

RECOMMENDATIONS

1. **Immediate:** Address 3 critical issues (within 2 weeks)
2. **Short-term:** Implement automated validation for promotional pricing
3. **Long-term:** Add database constraints to prevent future quality issues

NEXT AUDIT: January 7, 2025

Attachments: - Detailed validation results (47 rules) - Outlier analysis (full dataset) - Remediation tracking spreadsheet

---

Done.

WHAT JUST HAPPENED:

Traditional data quality audit: 1. Manually write SQL for each validation → 45 min 2. Run queries one by one → 30 min 3. Export results to Excel → 15 min 4. Check constraints manually → 60 min 5. Profile columns in spreadsheet → 45 min 6. Identify anomalies visually → 30 min 7. Research business rules → 30 min 8. Write audit report → 90 min **Total: 6+ hours**

Uderia @QUALITY workflow: 1. Request comprehensive assessment → 45 sec 2. Review constraint violations → 30 sec 3. Profile critical columns → 35 sec 4. Validate business rules → 40 sec 5. Quick DDL lookup with @DBA → 25 sec 6. Generate compliance report → 50 sec **Total: 4 minutes 30 seconds**

THE POWER: - **Automated validation** (47 rules executed simultaneously) - **AI-driven profiling** (qlty_profileColumn with statistical analysis) - **Anomaly detection** (outliers identified automatically) - **Business rule enforcement** (price consistency checking) - **Context-aware recommendations** (champion cases from 29 quality patterns) - **Compliance-ready reporting** (SOC 2, ISO 8000 formatted)

97.3/100 quality score. 6 issues identified. Remediation plan delivered.

Priya went from “monthly audit” to “compliance-ready report” in 4.5 minutes.

One analyst. One profile. Zero manual Excel work.

This is @QUALITY. This is Uderia. This is data governance evolved.

```
**Screen Capture Plan:**
- [ ] Profile indicator: @QUALITY active
- [ ] Comprehensive quality assessment query
- [ ] `qlty_profileColumn` results with distributions
```

```
- [ ] Anomaly detection highlighting 3 outliers
- [ ] Business rule violation flagged
- [ ] Brief switch to @DBA for DDL context
- [ ] Quality score calculation: 97.3/100
- [ ] TDA_FinalReport generates audit summary

**Supporting Materials:**
- Quality audit checklist
- Expected anomalies documentation
- Compliance report template

---

### 10.4 Compliance Officer Journey - @LOCAL Profile
**Duration:** 4-5 minutes
**Persona:** James, Compliance Officer analyzing customer churn (PII-heavy)
**Profile Used:** Exclusively @LOCAL (zero cloud exposure)

**Learning Objectives:**
- Demonstrate complete privacy-preserving workflow
- Show identical capabilities with local execution
- Emphasize data sovereignty

**Scenario Narrative:**
James needs to analyze why premium customers are churning, but the analysis involves sensitive

**Key Topics to Cover:**
- [ ] Activate @LOCAL (Ollama) with sovereignty verification
- [ ] Customer analysis: Identify churned customers (full PII)
- [ ] Purchase pattern and service ticket correlation
- [ ] Generate confidential churn report
- [ ] Confirm: Zero cloud API calls, 100% on-premises

**Narration Script:**
```

Meet James. Compliance Officer. Operates under strict regulatory constraints.

Assignment: Analyze why premium customers are churning.

The problem: Analysis requires sensitive PII—full names, email addresses, home addresses, purchase history.

Regulatory requirement: **ZERO data can leave organizational boundaries.**

GDPR. CCPA. Industry regulations. Data sovereignty non-negotiable.

Traditional cloud approach: BLOCKED. Cannot send PII to Anthropic, OpenAI, or Google.

Alternative: Build custom on-prem solution → 6 months development, $300K budget.

Uderia approach: **@LOCAL profile. Same intelligence. Zero cloud.**

THE SOVEREIGNTY WORKFLOW:

James logs into Uderia. Profile selector shows 6 options.

He selects **@LOCAL** (Ollama Llama 3.1 70B - On-Premises).

Chat opens. Profile indicator: "@LOCAL Active - 100% On-Premises Execution"

System banner appears:

```
SOVEREIGNTY MODE ACTIVE
LLM: Ollama (local server at 192.168.1.50)
Database: Direct connection (no cloud proxy)
RAG: ChromaDB (local vector store)
MCP Tools: On-premises execution only
Zero outbound traffic to cloud APIs
```

NETWORK MONITORING (bottom corner):

```
Cloud API Calls This Session: 0
Local Inference Requests: Active
Data Leaving Network: 0 bytes
```

James is confident. Let's verify.

QUERY 1: VERIFY SOVEREIGNTY

James types: *"Confirm that zero data is being sent to cloud providers during this session"*

@LOCAL responds (locally):

```
System Check: Data Sovereignty

LLM Provider: Ollama Llama 3.1 70B
  - Location: 192.168.1.50:11434 (internal network)
  - API calls: Local only
  - Data transmission: 0 bytes external

Database Connection: Teradata fitness_db
  - Location: 10.0.5.100:1025 (internal network)
  - Direct JDBC connection
  - No cloud proxy

Vector Store: ChromaDB
  - Location: localhost:8000
  - Knowledge collections: Stored on-premises
  - Embeddings: Generated locally (Sentence Transformers)

MCP Tools: All local execution
  - base_readQuery: Direct database access
  - TDA_Charting: Local rendering
  - TDA_FinalReport: Local generation

**Guaranteed:** No PII, query text, or results transmitted externally
```

Response:

Data Sovereignty: CONFIRMED □

All processing happening within organizational boundaries: - LLM reasoning: Local server - Data access: Internal database - Knowledge retrieval: On-premises vector store - Tool execution: Local MCP servers

Compliance Status: GDPR □ | CCPA □ | SOC 2 □ | Industry Regulations □

Sovereignty verified. Network monitor shows: 0 cloud API calls.

Now: the actual analysis with real PII.

QUERY 2: IDENTIFY CHURNED CUSTOMERS

James: *"Identify premium customers who stopped purchasing in Q4 2024. Include full names, email addresses, and home addresses."*

```
Tool: base_readQuery
  Query: SELECT with PII fields (FirstName, LastName, Email, Address, City, State)
  Filter: Lifetime value >$5000, No Q4 2024 purchases

Sovereignty: All PII processed locally, zero external transmission
```

Response:

Churned Premium Customers (Q4 2024)

Total Identified: 47 customers **Combined Lifetime Value:** $287,450 **Average LTV:** $6,116

Sample Records (PII included):

CustomerID	Full Name	Email	Address	City	State	LTV
10234	**Jennifer Martinez**	jennifer.martinez@gmail.com	1425 Oak Street	Seattle	WA	$8,450
10567	**Michael Chen**	m.chen@techcorp.com	892 Pine Ave	Portland	OR	$7,230
10892	**Sarah Johnson**	s.johnson@gmail.com	2341 Maple Dr	San Fran-cisco	CA	$6,890
11023	**David Rodriguez**	david.r@outlook.com	567 Elm Street	Seattle	WA	$6,450
11234	**Emily Thompson**	e.thompson@company.com	3421 Cedar Ln	Portland	OR	$5,980

Geographic Concentration: - Seattle: 23 customers (49%) - Portland: 12 customers (26%) - San Francisco: 8 customers (17%) - Other: 4 customers (8%)

Network Monitor Update: Still 0 cloud API calls. All PII stayed local.

PII accessed. Zero data leakage. Continue analysis.

QUERY 3: PURCHASE PATTERN ANALYSIS

James: *"What products did these churned customers previously purchase? Show purchase history for the last 12 months."*

```
Tool: base_readQuery
  JOIN: Customers → Sales → Products
  Period: December 2023 - November 2024

Analysis: Product preferences, purchase frequency, spending patterns
```

Response:

Purchase History: Churned Premium Customers

Product Category Breakdown (12 months):

Category	Purchases	Revenue	Avg Order
Cardio Equipment	89	$124,560	$1,400
Strength Equipment	34	$78,230	$2,301
Accessories	156	$18,450	$118

Top Products Purchased: 1. **ProRunner 3000 Treadmill** - 34 units ($42,160) 2. **Elite Elliptical Pro** - 28 units ($38,920) 3. **CardioMaster 5000** - 27 units ($43,480)

Key Insight: 68% of revenue from Cardio equipment. These are Cardio-focused customers.

Purchase Frequency: - Q1 2024: 67 transactions - Q2 2024: 72 transactions - Q3 2024: 58 transactions - **Q4 2024: 0 transactions** □ (complete stop)

Pattern: Progressive decline leading to complete churn in Q4.

Clear pattern. Cardio customers stopped buying. Why?

QUERY 4: SERVICE TICKET CORRELATION

James: *"Did these 47 customers have unresolved service tickets? Show ticket details including issue descriptions."*

```
Tool: base_readQuery
  JOIN: Customers → ServiceTickets
  Filter: CustomerID IN (churned list)
  Status: Open or Recently Closed
```

Response:

Service Ticket Analysis: Churned Customers

Tickets Filed: 63 total (from 41 of 47 customers = 87%)

Ticket Status: - Open: 12 tickets (19%) - Closed (Resolved): 34 tickets (54%) - Closed (Unresolved): 17 tickets (27%) □

Issue Categories:

Issue Type	Count	Resolution Rate
Treadmill Belt Malfunction	23	52%
Display Screen Failure	18	61%
Motor Overheating	15	47%
Noise/Vibration	7	71%

Critical Finding: 17 tickets closed as "unresolved" (customer satisfaction: FAILED)

Sample Unresolved Tickets:

CustomerID 10234 (Jennifer Martinez): - Ticket #245: "ProRunner 3000 belt slipping after 3 months" - Status: Closed Unresolved - Last Update: Oct 15, 2024 - Customer Comment: "Not fixed. Belt still slipping. Very disappointed."

CustomerID 10567 (Michael Chen): - Ticket #267: "Display screen went black, won't turn on" - Status: Closed Unresolved - Last Update: Nov 2, 2024 - Customer Comment: "Replacement part never arrived. Unusable equipment."

Root Cause: Poor service resolution → Customer dissatisfaction → Churn

---

Service failures driving churn. What about demographics?

QUERY 5: DEMOGRAPHIC INSIGHTS

James: *"Analyze demographic patterns of these churned customers. Age, location, income indicators, purchase frequency."*

```
Tool: base_readQuery
  Analysis: Age demographics, geographic distribution, spending patterns

Tool: TDA_LLMTask (local inference)
  Task: Demographic segmentation and insights
```

Response:

Demographic Profile: Churned Premium Customers

Age Distribution: - 25-34: 8 customers (17%) - **35-44: 23 customers (49%)** ← Majority - 45-54: 12 customers (26%) - 55+: 4 customers (8%)

Average Age: 41 years (prime earning years)

Geographic Concentration: - **Urban areas:** 43 customers (91%) - Suburban: 4 customers (9%) - Rural: 0 customers

Income Indicators (based on ZIP codes): - High income (>$100K): 34 customers (72%) - Upper-middle ($75K-$100K): 11 customers (23%) - Middle (<$75K): 2 customers (5%)

Purchase Behavior (Pre-Churn): - **Average Annual Spend:** $6,116 - **Purchase Frequency:** Every 2.3 months - **Product Preference:** 68% Cardio, 27% Strength, 5% Accessories - **Loyalty:** 87% had active service issues (high engagement)

Customer Profile: High-value, urban professionals (age 35-44) with disposable income who invested heavily in home Cardio equipment. They engaged with customer service (filed tickets) but received poor resolution. Result: Complete churn.

Network Monitor: Still 0 cloud API calls. Full demographic analysis completed locally.

James has the complete picture. Time for the confidential report.

QUERY 6: GENERATE CONFIDENTIAL CHURN REPORT

James: *"Generate a confidential customer churn report for executive review. Include PII for retention campaign targeting."*

Strategic Planning (local): Retrieves champion case from "Executive_Reporting_Patterns"

Knowledge (local): Accesses "Business Metrics Definitions" collection

```
Tool: TDA_FinalReport (local generation)
  Format: Confidential Executive Report
  Security: Internal distribution only
  PII Included: Yes (for retention targeting)
```

Response:

CONFIDENTIAL CUSTOMER CHURN ANALYSIS *Q4 2024 Premium Customer Retention Crisis Prepared by: James Anderson, Compliance Officer Date: December 7, 2024 Classification: INTERNAL USE ONLY - Contains PII*

EXECUTIVE SUMMARY

Churn Event: 47 premium customers (combined LTV: $287,450) completely stopped purchasing in Q4 2024.

Root Cause: Poor customer service resolution on Cardio equipment issues.

Financial Impact: - **Immediate Loss:** $287K lifetime value at risk - **Annual Revenue Impact:** $143K (47 customers × avg $3,050 annual spend) - **Retention Campaign ROI:** $115K recovery potential ($40K investment, 80% success rate)

CUSTOMER PROFILE

Demographics: - **Age:** 35-44 (49% majority), high-earning professionals - **Location:** Urban (Seattle 49%, Portland 26%, SF 17%) - **Income:** $100K+ annual (72% of group) - **Product Focus:** Cardio equipment (68% of purchases)

Behavioral Indicators: - High engagement: 87% filed service tickets - Loyalty demonstrated: Average 2.3 month purchase frequency (pre-churn) - Service dissatisfaction: 17 unresolved tickets (27% failure rate)

RETENTION CAMPAIGN TARGET LIST

Tier 1: High-Priority (LTV >$7,000) - 12 customers

1. **Jennifer Martinez** | jennifer.martinez@email.com
   - LTV: $8,450 | Address: 1425 Oak Street, Seattle, WA
   - Issue: ProRunner 3000 belt malfunction (unresolved)
   - Offer: Free replacement + $200 credit
2. **Michael Chen** | m.chen@techcorp.com
   - LTV: $7,230 | Address: 892 Pine Ave, Portland, OR
   - Issue: Display screen failure (part never arrived)
   - Offer: Immediate replacement + 2-year extended warranty
3. **Sarah Johnson** | s.johnson@gmail.com
   - LTV: $6,890 | Address: 2341 Maple Dr, San Francisco, CA
   - Issue: Motor overheating (closed unresolved)
   - Offer: Equipment upgrade + free maintenance plan

(Continues for all 12 Tier 1 customers…)

Tier 2: Medium-Priority (LTV $6,000-$7,000) - 18 customers *(Full list with contact details, addresses, specific issues, personalized offers…)*

Tier 3: Standard Priority (LTV $5,000-$6,000) - 17 customers *(Full list with contact details, addresses, specific issues, personalized offers…)*

RETENTION CAMPAIGN STRATEGY

Phase 1: Immediate Outreach (Week 1) - Personal phone calls from VP of Customer Success - Acknowledge service failures, sincere apology - Offer resolution: Equipment replacement or upgrade

Phase 2: Incentive Offers (Week 2) - Tier 1: $200 credit + extended warranty - Tier 2: $150 credit + maintenance plan - Tier 3: $100 credit + priority support

Phase 3: Follow-Up (Week 3-4) - Track response rates - Adjust offers based on feedback - Document lessons learned

FINANCIAL PROJECTIONS

Campaign Investment: - Equipment replacements/upgrades: $28,000 - Account credits: $6,400 - Staff time (outreach): $5,600 **Total: $40,000**

Expected Recovery (80% success rate): - Customers retained: 38 of 47 - Annual revenue preserved: $115,000 - 3-year LTV recovered: $230,000

ROI: 5.75x ($230K recovered / $40K invested)

COMPLIANCE & PRIVACY

 Data Handling: All PII processed on-premises (Uderia @LOCAL profile) **Regulatory Compliance:** GDPR, CCPA, SOC 2 compliant **Access Control:** Report distribution limited to C-suite and retention team **Audit Trail:** All data access logged for compliance review

Zero cloud API calls during analysis - Complete data sovereignty maintained.

RECOMMENDATIONS

1. **Immediate:** Launch retention campaign targeting all 47 customers

2. **Short-term:** Improve service ticket resolution process (target 90% resolution rate)
3. **Long-term:** Implement proactive quality monitoring for Cardio equipment line

NEXT STEPS

- Approve $40K retention budget (Dec 8)
- Begin Tier 1 outreach (Dec 9)
- Weekly progress reviews (Fridays)

Confidential Annexes: - Full customer contact list with PII (47 records) - Service ticket transcripts (63 tickets) - Retention offer scripts (3 tiers)

Done.

James saves report. Marks: **CONFIDENTIAL - INTERNAL ONLY**.

Network monitor final reading: **0 cloud API calls**. **0 bytes transmitted externally**.

All PII—names, emails, addresses, service issues—processed entirely on-premises.

WHAT JUST HAPPENED:

Traditional cloud-based approach: - **BLOCKED:** Regulatory prohibition on sending PII to cloud providers - **Alternative 1:** Manual analysis (2 weeks, incomplete insights) - **Alternative 2:** Custom on-prem solution ($300K, 6 months development)

Uderia @LOCAL approach: - **Same intelligence:** Claude-level reasoning (Llama 3.1 70B) - **Same tools:** All MCP tools, strategic planning, RAG knowledge - **Same transparency:** Execution logs, tool traces - **Same output:** Executive-ready report with actionable insights - **Zero cloud:** 100% on-premises execution - **Time:** 18 minutes (vs 2 weeks or 6 months)

THE SOVEREIGNTY PROOF:

Network monitoring throughout session: - Cloud API calls: **0** - Data transmitted externally: **0 bytes** - LLM inference: **Local only** (192.168.1.50) - Database access: **Direct JDBC** (internal network) - Vector store: **On-premises ChromaDB**

THE POWER: - **No cloud compromise** (identical capabilities locally) - **Full PII access** (names, emails, addresses queried freely) - **Strategic planning** (champion cases retrieved from local RAG) - **Business intelligence** (complete churn analysis with demographics) - **Executive reporting** (confidential report with retention targeting) - **Regulatory compliance** (GDPR, CCPA, SOC 2 guaranteed)

47 customers analyzed. $287K lifetime value at risk. $40K retention campaign designed. 5.75x ROI projected.

James went from “regulatory blocked” to “confidential report with PII-based targeting” in 18 minutes.

One compliance officer. One profile. Zero cloud exposure.

This is @LOCAL. This is Uderia. This is data sovereignty without compromise.

The promise kept: Cloud-level intelligence. Local execution. Zero trade-offs.

```
**Screen Capture Plan:**
- [ ] Profile switch: @ANALYST → @LOCAL
- [ ] Profile indicator showing "Ollama Llama 3.1 70B (Local)"
- [ ] Network monitor: Zero outbound traffic to cloud APIs
- [ ] Customer PII queries with full names, emails visible
- [ ] Same MCP tools, same strategic planning, same transparency
- [ ] Slightly slower inference (local model) but complete functionality
- [ ] Confidential report generation
- [ ] Sovereignty guarantee banner

**Supporting Materials:**
- Customer PII dataset (anonymized for demo)
- Compliance documentation
- Network traffic monitoring proof

---

### 10.5 "Pro Mode" Power User Journey - Multi-Profile Orchestration
**Duration:** 5 minutes
**Persona:** Elena, Head of Analytics orchestrating complete business intelligence workflow
**Profiles Used:** @QUALITY → @DBA → @ANALYST → @LOCAL → @PROD (all profiles in sequence)

**Learning Objectives:**
- Demonstrate the ultimate "Pro Mode" workflow
- Show seamless profile switching
- Illustrate team orchestration by one person

**Scenario Narrative:**
Elena needs to deliver a comprehensive competitive analysis involving data validation, query op
all in one afternoon.

**Workflow Orchestration (5 Phases, 2 Hours):**

**Phase 1: @QUALITY** - Validates Q4 data (99.2% completeness, 2 anomalies flagged)
**Phase 2: @DBA** - Optimizes LTV query (45s → 2.1s execution)
**Phase 3: @ANALYST** - Generates executive dashboard with 4 charts
**Phase 4: @LOCAL** - Analyzes PII locally (churned customer retention list)
**Phase 5: @PROD** - Deploys Airflow DAG for weekly automation

**Key Topics to Cover:**
- [ ] Seamless profile switching (5 profiles, one session)
- [ ] Context retention across profile changes
- [ ] Traditional: 5 people, 3 days | Uderia: 1 person, 2 hours

**Narration Script:**
```

Meet Elena. Head of Analytics. Executive mandate delivered this morning.

“Deliver comprehensive customer lifetime value analysis by end of day. Include data validation, optimized queries, executive insights, PII-based retention targeting, and production deployment.”

Traditional approach: Assemble the team. - Quality analyst validates data → 4 hours - DBA optimizes queries → 3 hours
- Business analyst generates insights → 4 hours - Compliance reviews PII analysis → 2 hours - Data engineer deploys to production → 3 hours - Coordination meetings → 2 hours **Total: 5 people, 3 days, 18 hours of work**

Uderia approach: Elena. One person. God mode. 2 hours.

THE ORCHESTRATION:

Elena logs into Uderia. Profile dropdown shows all 6 options.

This workflow needs 5 of them.

Time: 2:00 PM. Deadline: 5:00 PM. Let’s go.

PHASE 1: DATA VALIDATION (@QUALITY) - 2:00 PM

Profile selector → **@QUALITY** (Claude Sonnet + validation tools)

Profile indicator: “@QUALITY Active”

Elena types: *“Validate data quality for Q4 2024 sales and customer data. Flag any issues that could affect lifetime value calculations.”*

```
Tool: qlty_validateData
  Tables: Sales, Customers, SaleDetails
  Period: Q4 2024

Automated checks: Referential integrity, NULL values, outliers
```

Response (45 seconds):

Data Quality Validation: Q4 2024

Overall Score: 99.2/100 □

Issues Detected (2):

1. NULL TotalAmount (CRITICAL) □ - Table: Sales - Count: 5 rows (SaleIDs: 234789, 234790, 234812, 235001, 235002) - Impact: Missing revenue data for LTV calculations - **Action Required:** Populate from SaleDetails before analysis

2. Price Inconsistencies (WARNING) □ - Table: SaleDetails - Count: 12 rows with UnitPrice ≠ Products.Price - Impact: $876 revenue discrepancy - **Action Advised:** Verify promotional pricing or correct records

Data Completeness: 99.8% (acceptable for analysis)

Recommendation: Proceed with analysis, but flag 5 NULL records for exclusion.

Elena notes the issues. Data quality: confirmed. Time: 45 seconds.

Quick switch to optimize performance.

PHASE 2: QUERY OPTIMIZATION (@DBA) - 2:01 PM

Profile dropdown → **@DBA** (Claude Sonnet + database tools)

Capabilities reload. Watch: Profile icon changes, tool list updates.

Profile indicator: "@DBA Active"

Elena pastes her query:

```
SELECT
    c.CustomerID,
    c.FirstName || ' ' || c.LastName as CustomerName,
    COUNT(s.SaleID) as PurchaseCount,
    SUM(s.TotalAmount) as LifetimeValue,
    AVG(s.TotalAmount) as AvgOrderValue,
    MAX(s.SaleDate) as LastPurchase
FROM Customers c
LEFT JOIN Sales s ON c.CustomerID = s.CustomerID
WHERE s.TotalAmount IS NOT NULL  -- Exclude nulls flagged by @QUALITY
GROUP BY c.CustomerID, c.FirstName, c.LastName
ORDER BY LifetimeValue DESC;
```

Elena: *"Analyze this query for performance. Optimize if needed."*

```
 Tool: base_readQuery (EXPLAIN mode)
  Analyzing execution plan...

 Bottleneck: Full table scan on Sales (237K rows)
  Missing index on CustomerID (FK not indexed)
```

Response (50 seconds):

Performance Analysis

Current Execution: 12.3 seconds **Rows Scanned:** 237,450 (Sales) + 8,945 (Customers)

Optimization:

```
-- Create index (run ONCE)
CREATE INDEX idx_sales_customerid ON Sales (CustomerID);
COLLECT STATISTICS ON Sales COLUMN (CustomerID);

-- Optimized query (same logic, uses index)
SELECT
    c.CustomerID,
    c.FirstName || ' ' || c.LastName as CustomerName,
```

```
    COUNT(s.SaleID) as PurchaseCount,
    SUM(s.TotalAmount) as LifetimeValue,
    AVG(s.TotalAmount) as AvgOrderValue,
    MAX(s.SaleDate) as LastPurchase
FROM Customers c
LEFT JOIN Sales s ON c.CustomerID = s.CustomerID
WHERE s.TotalAmount IS NOT NULL
GROUP BY c.CustomerID, c.FirstName, c.LastName
ORDER BY LifetimeValue DESC;
```

Optimized Execution: 1.8 seconds (85% faster)

Index created. Query ready for production.

Elena saves optimized query. Performance: locked in. Time: 50 seconds.

Now: business insights.

PHASE 3: BUSINESS INTELLIGENCE (@ANALYST) - 2:02 PM

Profile dropdown → **@ANALYST** (Gemini Flash + business tools)

Capabilities reload. Visualization tools appear.

Profile indicator: "@ANALYST Active"

Elena: *"Using the optimized query, analyze customer lifetime value by segment. Create executive dashboard with visualizations."*

Strategic Planning retrieves: "Customer_Segmentation_Analysis" (17 champion cases)

Knowledge: "Business Metrics Definitions" collection

```
Tool: base_readQuery (optimized query)
  Execution: 1.8 seconds

Tool: TDA_Charting (4 visualizations)
  - Segment distribution (pie chart)
  - LTV by segment (bar chart)
  - Purchase frequency (line graph)
  - Top 20 customers (table)
```

Response (65 seconds):

Customer Lifetime Value Analysis

Total Customers: 8,945 **Total LTV:** $18.4M **Average LTV:** $2,057

Segmentation:

[Pie Chart: Customer Segments]

1. Premium (LTV >$5,000) - 847 customers (9%) - Total LTV: $7.2M (39% of revenue) - Avg LTV: $8,502 - Avg Purchases: 6.8 per customer

2. Standard (LTV $2,000-$5,000) - 2,134 customers (24%) - Total LTV: $6.8M (37% of revenue) - Avg LTV: $3,186 - Avg Purchases: 3.2 per customer

3. Emerging (LTV $500-$2,000) - 3,456 customers (39%) - Total LTV: $3.6M (20% of revenue) - Avg LTV: $1,041 - Avg Purchases: 1.8 per customer

4. Entry (LTV <$500) - 2,508 customers (28%) - Total LTV: $0.8M (4% of revenue) - Avg LTV: $319 - Avg Purchases: 1.1 per customer

[Bar Chart: LTV by Segment]

Key Insights:

 Premium segment: 9% of customers generate 39% of revenue (high concentration) **Churn risk:** 47 premium customers churned in Q4 (identified earlier) **Opportunity:** Emerging segment (3,456 customers) has upsell potential to Standard

Recommendations: 1. **Retention:** Launch premium customer retention program (focus on 47 churned) 2. **Growth:** Target Emerging segment with personalized upsell campaigns 3. **Prevention:** Monitor Premium segment service tickets proactively

---

Elena has executive insights. Visualizations ready. Time: 65 seconds.

But she needs specific names for retention targeting. That's sensitive PII.

Switch to sovereign execution.

PHASE 4: SENSITIVE DATA ANALYSIS (@LOCAL) - 2:04 PM

Profile dropdown → **@LOCAL** (Ollama Llama 3.1 70B - On-Premises)

Capabilities reload. Network monitor activates.

Profile indicator: "@LOCAL Active - 100% On-Premises"

System Banner:

```
SOVEREIGNTY MODE ACTIVE
Zero cloud API calls | All data stays local
```

Elena: *"From the 847 premium customers, identify the 47 who churned in Q4. Include full names, emails, and addresses for retention campaign."*

Context retained from earlier phases. @LOCAL knows about the customer segments.

```
Tool: base_readQuery (local execution)
  Query: Premium customers with NO Q4 2024 purchases
  PII fields: FirstName, LastName, Email, Address, City, State

Sovereignty: All PII processed on-premises, zero cloud transmission
```

Response (70 seconds - slightly slower, local inference):

Churned Premium Customers - Retention Target List

Total Identified: 47 customers **Combined LTV at Risk:** $287,450

Top 10 High-Priority Targets:

CustomerID	Name	Email	Address	LTV	Last Purchase
10234	Jennifer Martinez	jennifer.martinez@email.com	1425 Oak St, Seattle, WA	$8,450	Sept 28, 2024
10567	Michael Chen	m.chen@techcorp.com	892 Pine Ave, Portland, OR	$7,230	Sept 15, 2024
10892	Sarah Johnson	s.johnson@gmail.com	2341 Maple Dr, San Francisco, CA	$6,890	Oct 3, 2024
11023	David Rodriguez	david.r@outlook.com	567 Elm St, Seattle, WA	$6,450	Sept 22, 2024
11234	Emily Thompson	e.thompson@yahoo.com	3421 Cedar Ln, Portland, OR	$5,980	Oct 12, 2024
…	(42 more)	…	…	…	…

Network Monitor: 0 cloud API calls. All PII stayed on-premises.

Export: Confidential retention list saved locally.

Elena has names and contact details. Compliance-safe. Time: 70 seconds.

Final phase: operationalize everything for production.

PHASE 5: PRODUCTION DEPLOYMENT (@PROD) - 2:06 PM

Profile dropdown → **@PROD** (Claude Sonnet + automation tools)

Capabilities reload. API documentation appears.

Profile indicator: “@PROD Active”

Elena: *“Create production-ready Airflow DAG that executes the complete workflow: data validation, optimized LTV calculation, segmentation analysis, and automated reporting. Schedule weekly.”*

Strategic Planning: “Production_Automation_Patterns” repository (34 champion cases)

```
Tool: TDA_FinalReport
  Format: Python Airflow DAG
```

```
Schedule: Weekly (Mondays 6 AM)
Profiles: @QUALITY → @DBA → @ANALYST (automated sequence)
```

Response (80 seconds):

Airflow DAG: Customer LTV Analytics Pipeline

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta
import requests
import os

def validate_data_quality():
    """Phase 1: Data validation with @QUALITY"""
    response = requests.post(
        'https://uderia.company.com/api/v1/query',
        headers={'Authorization': f'Bearer {os.environ["UDERIA_TOKEN"]}'},
        json={
            'profile': '@QUALITY',
            'query': 'Validate Q4 sales and customer data quality',
            'session_id': 'ltv_pipeline'
        }
    )

    results = response.json()
    quality_score = results['quality_score']

    if quality_score < 95:
        raise Exception(f"Data quality below threshold: {quality_score}%")

    print(f" Data quality validated: {quality_score}%")
    return results

def optimize_query():
    """Phase 2: Query optimization with @DBA"""
    optimized_sql = """
        SELECT c.CustomerID, c.FirstName || ' ' || c.LastName as CustomerName,
               COUNT(s.SaleID) as PurchaseCount,
               SUM(s.TotalAmount) as LifetimeValue,
               AVG(s.TotalAmount) as AvgOrderValue,
               MAX(s.SaleDate) as LastPurchase
        FROM Customers c
        LEFT JOIN Sales s ON c.CustomerID = s.CustomerID
        WHERE s.TotalAmount IS NOT NULL
        GROUP BY c.CustomerID, c.FirstName, c.LastName
        ORDER BY LifetimeValue DESC;
    """
```

```
    response = requests.post(
        'https://uderia.company.com/api/v1/query',
        headers={'Authorization': f'Bearer {os.environ["UDERIA_TOKEN"]}'},
        json={
            'profile': '@DBA',
            'sql': optimized_sql,
            'session_id': 'ltv_pipeline'
        }
    )

    results = response.json()
    print(f" Query executed: {results['execution_time']}")
    return results

def generate_insights():
    """Phase 3: Business intelligence with @ANALYST"""
    response = requests.post(
        'https://uderia.company.com/api/v1/query',
        headers={'Authorization': f'Bearer {os.environ["UDERIA_TOKEN"]}'},
        json={
            'profile': '@ANALYST',
            'query': 'Analyze customer LTV by segment with visualizations',
            'session_id': 'ltv_pipeline'
        }
    )

    results = response.json()

    # Save report to S3
    report_url = save_to_s3(results['report'])

    # Send Slack notification
    send_slack_notification(f" Weekly LTV Report: {report_url}")

    print(f" Report generated and distributed")
    return results

default_args = {
    'owner': 'elena',
    'retries': 2,
    'retry_delay': timedelta(minutes=5)
}

dag = DAG(
    'customer_ltv_analytics',
    default_args=default_args,
    description='Automated customer lifetime value analysis pipeline',
    schedule_interval='0 6 * * 1',  # Weekly: Mondays at 6 AM
```

```
    start_date=datetime(2024, 12, 7),
    catchup=False
)

# Task dependencies
task1 = PythonOperator(
    task_id='validate_data_quality',
    python_callable=validate_data_quality,
    dag=dag
)

task2 = PythonOperator(
    task_id='optimize_query',
    python_callable=optimize_query,
    dag=dag
)

task3 = PythonOperator(
    task_id='generate_insights',
    python_callable=generate_insights,
    dag=dag
)

# Pipeline: Quality → Optimization → Insights
task1 >> task2 >> task3
```

Deployment Instructions: 1. Save to `dags/customer_ltv_analytics.py` 2. Set environment variable: `UDERIA_TOKEN` 3. Deploy to Airflow 4. First run: Monday, Dec 9, 2024 at 6:00 AM

Automated Workflow: - **Phase 1:** @QUALITY validates data (45s) - **Phase 2:** @DBA executes optimized query (1.8s) - **Phase 3:** @ANALYST generates report with visualizations (65s) - **Total:** ~2 minutes execution weekly

Output: Executive report auto-distributed via Slack, saved to S3.

Done. Pipeline deployed.

Elena checks time: **2:08 PM**

8 minutes total. 5 profiles orchestrated. Complete workflow deployed.

Let's see what just happened.

THE PRO MODE SUMMARY

Elena started at 2:00 PM. Finished at 2:08 PM.

Profile orchestration:

 @QUALITY (45s) - Data validation, 99.2% quality score, 2 issues flagged **@DBA** (50s) - Query optimization, 12.3s → 1.8s (85% faster) **@ANALYST** (65s) - Business intelligence, 4 visualizations, 3 recommendations **@LOCAL** (70s) - PII analysis, 47 customers identified, zero cloud exposure **@PROD** (80s) - Production deployment, Airflow DAG, weekly automation

Total Time: 8 minutes (including thinking time: ~12 minutes)

Deliverables: 1. Data quality report (99.2% score) 2. Optimized SQL query (85% performance gain) 3. Executive dashboard (4 visualizations, segmentation analysis) 4. Confidential retention list (47 customers with PII) 5. Production pipeline (Airflow DAG, weekly execution)

Traditional Approach:

Team Required: - Priya (Quality Analyst) - 4 hours - Marcus (DBA) - 3 hours - Sarah (Business Analyst) - 4 hours - James (Compliance) - 2 hours - Data Engineer (Production) - 3 hours

Coordination: - Kickoff meeting - 30 min - Mid-point sync - 45 min - Final review - 45 min

Total: 5 people × 3 days = 18 hours of work + 2 hours meetings

Bottlenecks: - Waiting for quality validation before optimization - Waiting for optimization before analysis - Waiting for compliance review before retention targeting - Waiting for everything before deployment - Back-and-forth iterations at each handoff

Uderia Pro Mode:

Team Required: Elena (1 person)

Time: 8 minutes execution + 4 minutes thinking = 12 minutes total

Bottlenecks: ZERO. Seamless profile switching with context retention.

The Difference:

Dimension	Traditional	Uderia Pro Mode	Improvement
People	5 specialists	1 person	80% reduction
Time	3 days (18 hours)	12 minutes	**99.3%** faster
Meetings	3 coordination sessions	0	100% elimination
Handoffs	4 handoffs (friction points)	0	Seamless flow
Context Loss	High (explain at each step)	0	Perfect retention
Iterations	Slow (days per cycle)	Instant	Real-time refinement

THE COMPOUNDING POWER:

Each profile contributed specialized expertise:

 @QUALITY: Caught 2 data issues that would have corrupted analysis **@DBA:** Delivered 85% performance gain (12.3s → 1.8s) **@ANALYST:** Generated executive visualizations and actionable insights **@LOCAL:** Handled sensitive PII without regulatory risk **@PROD:** Operationalized entire workflow for ongoing automation

Alone, each profile is powerful. Together, orchestrated by one person, they're unstoppable.

THIS IS PRO MODE.

One person commanding a team of specialized AI personas.

No coordination overhead. No knowledge handoffs. No waiting. No friction.

Pure, compounded intelligence.

Elena went from "executive mandate" to "fully deployed production pipeline" in 12 minutes.

Traditional team: 5 people, 3 days, countless meetings. **Elena with Uderia:** 1 person, 12 minutes, zero meetings.

This is the future.

Not replacing specialists.

Empowering one person to orchestrate all of them.

This is @QUALITY + @DBA + @ANALYST + @LOCAL + @PROD.

This is Uderia.

This is Pro Mode. "'

Screen Capture Plan: - [ ] Profile dropdown showing all 5 profiles - [ ] Phase 1: @QUALITY validation (3 min segment) - [ ] Quick switch to @DBA (watch capabilities reload) - [ ] Phase 2: @DBA optimization (4 min segment) - [ ] Switch to @ANALYST (visualization tools appear) - [ ] Phase 3: @ANALYST insights with charts (5 min segment) - [ ] Switch to @LOCAL (network monitor shows zero cloud traffic) - [ ] Phase 4: @LOCAL PII analysis (3 min segment) - [ ] Switch to @PROD (API documentation visible) - [ ] Phase 5: @PROD operationalization (4 min segment) - [ ] Final montage: All 5 profiles contributed to one cohesive deliverable - [ ] Side-by-side comparison: Traditional (5 people, 3 days) vs Uderia (1 person, 2 hours)

Supporting Materials: - Complete workflow diagram showing all 5 phases - Traditional vs Uderia comparison infographic - "God mode" capability matrix - Time/cost savings calculator

---

□ Demo Environment Reference

Database Schema: fitness_db

Business Context: Fitness equipment retail company with sales tracking and customer service management

Tables Overview:

1. **Customers** (Base table)
   - CustomerID, FirstName, LastName, Email, Phone
   - Address, City, State, ZipCode
   - RegistrationDate
   - **Use Cases:** Customer demographics, geographic analysis, customer lifetime value
2. **Products** (Inventory table)

- ProductID, ProductName, ProductType, Brand
- Price, StockQuantity
- **Use Cases:** Product catalog queries, inventory tracking, pricing analysis

3. **Sales** (Transaction header)
   - SaleID, CustomerID, SaleDate, TotalAmount, SalesPersonID
   - **Use Cases:** Revenue trends, sales performance, temporal analysis
4. **SaleDetails** (Transaction line items)
   - SaleDetailID, SaleID, ProductID, Quantity, UnitPrice
   - **Use Cases:** Product-level revenue, quantity analysis, pricing validation
5. **ServiceTickets** (Customer support)
   - TicketID, CustomerID, ProductID, TicketDate
   - IssueDescription, Status, ResolutionDate
   - **Use Cases:** Product quality analysis, customer satisfaction, warranty tracking

Key Relationships: - Sales → Customers (one-to-many) - SaleDetails → Sales (one-to-many) - SaleDetails → Products (many-to-one) - ServiceTickets → Customers (many-to-one) - ServiceTickets → Products (many-to-one)

MCP Tools Configuration

Core MCP Tools (Teradata Server) BASE Category - Database Operations: - `base_databaseList` - List all databases in the system - `base_tableList` - List all tables in a database - `base_tableDDL` - Retrieve table DDL (CREATE TABLE statement) - `base_readQuery` - Execute SELECT queries and return results - `base_databaseBusinessDesc` (prompt) - Get business description of database - `base_tableBusinessDesc` (prompt) - Get business description of table

QUALITY Category - Data Validation: - `qlty_*` tools - Data profiling, validation, anomaly detection - `qlty_databaseQuality` (prompt) - Comprehensive database quality assessment

DBA Category - Administration: - `dba_databaseVersion` - Get Teradata system version - `dba_databaseLineage` (prompt) - Data lineage analysis - `dba_databaseHealthAssessment` (prompt) - System health check - `dba_userActivityAnalysis` (prompt) - User activity tracking - `dba_systemVoice` (prompt) - System status narrative - `dba_tableArchive` (prompt) - Table archiving recommendations - `dba_tableDropImpact` (prompt) - Impact analysis for dropping tables

Client-Side Tools (Built into Uderia) TDA Category - Platform Tools: - `TDA_FinalReport` - Format structured final report for UI - `TDA_ComplexPromptReport` - Execute complex prompt workflows - `TDA_LLMTask` - Delegate tasks to LLM for analysis - `TDA_CurrentDate` - Get current date/time - `TDA_SystemLog` - Log system messages to Live Status Panel - `TDA_Charting` - Generate data visualizations (charts)

MCP Tools by User Persona:

For Business Users (Modules 1-4, 6, 10.1, 10.2): - Primary: `base_readQuery`, `base_tableList`, `base_tableDDL`, `base_databaseList` - Secondary: `TDA_FinalReport`, `TDA_LLMTask`, `TDA_Charting` - Prompts: `base_databaseBusinessDesc`, `base_tableBusinessDesc`

For Administrators (Modules 7, 8, 10.4): - Primary: `dba_databaseVersion`, all `dba_*` prompts - Secondary: All consumption tracking endpoints (via REST API) - Focus: Cost analytics, user

management, system monitoring

For DevOps/Advanced (Modules 9, 10.3): - All tools available: Full system access for integration demonstrations - Focus: REST API automation, Airflow integration, profile switching

Sample Queries by Persona

Knowledge Worker Queries: - "What are the top 5 fitness products by revenue this month?" - "Show me customer demographics for people who bought treadmills" - "Which products have the highest return/service ticket rates?" - "Compare sales performance across different states" - "What's the average time to resolve service tickets by product type?"

Content Developer Queries: - "Analyze the data quality of the Email field in Customers table" - "Create a comprehensive sales pattern for Q4 holiday season" - "What are the most common SQL patterns for revenue analysis?"

Administrator Queries: - "Show me total LLM costs for the past 30 days" - "Which users are consuming the most tokens?" - "What are the top 5 most expensive queries this week?" - "Display current consumption profile allocations"

DevOps Engineer Queries: - "Execute this query asynchronously and poll for results" - "Create an Airflow DAG for daily sales aggregation" - "Test the same query across Claude and GPT-4 for comparison"

Expected Demonstrations

Module 1 (First Conversation): - Query: "Top 5 products by revenue" - **MCP Tools Shown:** `base_readQuery` (executing SQL query) - **Shows:** Strategic planning, SQL generation, table rendering - Follow-up: "Customer demographics for top product" - **Shows:** Context retention, multi-table joins, `TDA_FinalReport` formatting

Module 3.2 (Capabilities Discovery): - **MCP Tools Tab:** Show `base_tableList`, `base_readQuery`, `base_tableDDL`, `dba_databaseVersion` - **MCP Prompts Tab:** Show `base_databaseBusinessDesc`, `base_tableBusinessDesc`, categorized display - **Demonstrate:** Adding new tool to profile, seeing it appear in capabilities

Module 3.4 (System Customization): - **Demonstrate:** Disabling `base_tableDDL` tool and seeing queries fail without it - **Demonstrate:** Re-enabling tool and query succeeding - **Show:** Dynamic capability management (enable/disable specific tools/prompts)

Module 4.2 (RAG Self-Learning): - **Show:** RAG case containing `base_readQuery` tool execution - **Demonstrate:** Similar query retrieving past `base_readQuery` pattern from RAG - **MCP Context:** How RAG captures tool names, arguments, and results

Module 2 (REST API): - Convert above conversational query to cURL command - Show session management and result retrieval

Module 3 (Transparency): - Complex multi-phase query requiring recursive planning - Demonstrate self-correction when initial approach fails

Module 4 (Efficiency): - Show RAG case capture after successful query - Demonstrate plan hydration with follow-up question - Display token savings from RAG-guided planning

Module 6 (Marketplace): - Create template from successful revenue analysis query - Publish to marketplace as “Retail Revenue Patterns” - Show another user deploying and using the pattern

Module 10 (User Journeys): - Knowledge Worker: Full exploratory analysis workflow - Administrator: Cost monitoring and user management - DevOps: Production deployment with Airflow integration

---

□ Tutorial Creation Workflow

For Each Section:

1. **Preparation Phase**
   - ☐ Review learning objectives
   - ☐ Prepare demo environment/data
   - ☐ Test all screen captures
   - ☐ Write and refine narration script
2. **Recording Phase**
   - ☐ Record screen capture (Tight.studio)
   - ☐ Record narration (voice-over or live)
   - ☐ Capture B-roll (UI elements, diagrams)
   - ☐ Ensure 2-5 minute target duration
3. **Post-Production Phase**
   - ☐ Edit for clarity and pacing
   - ☐ Add callouts/highlights
   - ☐ Insert transition slides
   - ☐ Review for technical accuracy
4. **Quality Assurance**
   - ☐ Verify learning objectives achieved
   - ☐ Test with sample audience
   - ☐ Gather feedback
   - ☐ Iterate if needed

□ Success Metrics

Tutorial Effectiveness Indicators: - [ ] Viewers can successfully complete learning objectives - [ ] Clear differentiation from competitors established - [ ] All six core principles comprehensively covered - [ ] Actionable takeaways in every section - [ ] Professional production quality (Tight.studio)

Key Messaging Achieved: - [ ] “Two-in-one” conversation → API paradigm clear - [ ] “Hybrid intelligence” cloud + local positioning understood - [ ] “Self-learning RAG” continuous improvement demonstrated - [ ] “Marketplace” collective intelligence ecosystem showcased - [ ] “Complete transparency” black box elimination proven - [ ] “Financial governance” cost control demonstrated

---

Reference Materials

Primary Sources: - README.md (comprehensive feature documentation) - www.uderia.com (value proposition and messaging) - docs/ directory (technical deep-dives) - Source code (implementation details)

Visual Assets Needed: - Architecture diagrams - UI screenshots (all major views) - Workflow comparison diagrams - Cost savings visualizations - Marketplace ecosystem graphics

Demo Environment Requirements: - Configured MCP servers (database, API) - Sample data for realistic queries - Multiple LLM providers configured - Pre-populated RAG collections - Multiple user accounts for multi-user demos

---

Next Steps

1. **Review & Refine:** Validate this structure with stakeholders
2. **Script Writing:** Fill in narration placeholders section-by-section
3. **Environment Setup:** Prepare demo environment and sample data
4. **Pilot Recording:** Create 2-3 sample sections for feedback
5. **Iteration:** Refine based on pilot feedback
6. **Full Production:** Record all sections systematically
7. **Launch:** Publish to Tight.studio and promote

Document Status: Structure Complete - Ready for Narration Scripting
Next Action Required: Begin filling in narration scripts starting with Module 1