Comprehensive Database Test Plan

PSScript Platform - January 15, 2026

Executive Summary

This test plan provides a systematic approach to verify database integrity, API integration, and end-to-end data flow for the PSScript PowerShell analysis platform. All tests are designed to validate the current configuration as of January 15, 2026.

Test Environment

Infrastructure

Component	Technology	Port	Test Coverage
Database	PostgreSQL 15 + pgvector	5432	Unit, Integration
Cache	Redis 7.0	6379	Integration
Backend	Express/TypeScript	4000	Unit, Integration, E2E
Frontend	React/Vite	3000	E2E
AI Service	FastAPI/Python	8000	Integration

Prerequisites

```
 # Start services
docker-compose up -d postgres redis

# Verify connections
psql -h localhost -p 5432 -U postgres -d psscript -c "SELECT 1"
redis-cli -p 6379 ping
```

Test Categories

1. Unit Tests (Database Layer)

Location: `src/backend/src/__tests__/database.test.ts`

Test ID	Description	Priority	Status
DB-U-001	PostgreSQL connection	Critical	✅ Created
DB-U-002	Health check query	Critical	✅ Created
DB-U-003	pgvector extension verification	High	✅ Created
DB-U-004	Connection pool settings	High	✅ Created
DB-U-005	User CRUD operations	Critical	✅ Created
DB-U-006	Script CRUD operations	Critical	✅ Created
DB-U-007	Category CRUD operations	High	✅ Created
DB-U-008	Tag associations	Medium	✅ Created
DB-U-009	Script versioning	High	✅ Created
DB-U-010	Script analysis storage	High	✅ Created
DB-U-011	Execution logging	Medium	✅ Created
DB-U-012	Index verification	High	✅ Created
DB-U-013	Cascade delete behavior	High	✅ Created
DB-U-014	Transaction rollback	Critical	✅ Created
DB-U-015	Data integrity (Unicode, special chars)	Medium	✅ Created

Run Command:

```
cd src/backend && npm test -- --testPathPattern=database.test.ts
```

---

2. Integration Tests (API → Database)

Test ID	Description	Endpoint	Method
DB-I-001	User registration stores in DB	/auth/register	POST
DB-I-002	Login validates password hash	/auth/login	POST
DB-I-003	Script upload with deduplication	/scripts/upload	POST
DB-I-004	Script list with pagination	/scripts	GET
DB-I-005	Script search with filtering	/scripts/search	GET
DB-I-006	Analysis results storage	/scripts/:id/analyze	POST
DB-I-007	Version history retrieval	/scripts/:id/versions	GET
DB-I-008	Execution log creation	/scripts/:id/execute	POST
DB-I-009	Category CRUD via API	/categories	ALL
DB-I-010	Chat history persistence	/chat/history	POST
DB-I-011	Analytics aggregation queries	/analytics/*	GET
DB-I-012	Bulk delete with transaction	/scripts/delete	POST

Test Script:

```bash
#!/bin/bash
# Integration test runner

BASE_URL="http://localhost:4000/api"
TOKEN="" # Set after login

# Test DB-I-001: User Registration
echo "Testing user registration..."
REGISTER_RESPONSE=$(curl -s -X POST "$BASE_URL/auth/register" \
  -H "Content-Type: application/json" \
  -d '{"username":"test_'$(date +%s)'","email":"test_'$(date +%s)
echo "$REGISTER_RESPONSE" | jq .

# Test DB-I-002: Login
echo "Testing login..."
LOGIN_RESPONSE=$(curl -s -X POST "$BASE_URL/auth/login" \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@psscript.com","password":"admin123"}')
TOKEN=$(echo "$LOGIN_RESPONSE" | jq -r '.token')
echo "Token: ${TOKEN:0:20}..."

# Test DB-I-004: Script list
echo "Testing script list with pagination..."
curl -s "$BASE_URL/scripts?page=1&limit=5" \
  -H "Authorization: Bearer $TOKEN" | jq '.data | length'

# Test DB-I-011: Analytics
echo "Testing analytics..."
curl -s "$BASE_URL/analytics/security" \
  -H "Authorization: Bearer $TOKEN" | jq '.summary'

echo "Integration tests complete!"
```

3. End-to-End Tests (Frontend → API → Database)

Test ID	Description	User Flow
DB-E-001	Complete script upload flow	Upload → Validate → Store → Display
DB-E-002	Script analysis workflow	Select → Analyze → Save → Show Results
DB-E-003	User authentication flow	Register → Login → Access Protected
DB-E-004	Category filtering	Select Category → Filter Scripts → Verify
DB-E-005	Version history navigation	View Script → Show Versions → Revert
DB-E-006	Search functionality	Enter Query → Search → Display Results
DB-E-007	Dashboard statistics	Load → Aggregate → Display Charts
DB-E-008	Chat persistence	Send Message → Store → Reload → Verify

Playwright Test Example:

```
// e2e/database-flow.spec.ts
import { test, expect } from '@playwright/test';

test.describe('Database E2E Tests', () => {
  test('DB-E-001: Complete script upload flow', async ({ page }) =
    // Login
    await page.goto('/login');
    await page.fill('[name="email"]', 'admin@psscript.com');
    await page.fill('[name="password"]', 'admin123');
    await page.click('button[type="submit"]');
    await expect(page).toHaveURL('/dashboard');

    // Navigate to upload
    await page.click('text=Upload Script');

    // Upload a script
    const scriptContent = 'Get-Process | Select-Object Name, CPU'
    await page.fill('[name="title"]', 'E2E Test Script');
    await page.fill('[name="content"]', scriptContent);
    await page.click('button:has-text("Upload")');

    // Verify script appears in list
    await expect(page.locator('text=E2E Test Script')).toBeVisible

    // Verify in database (via API)
    const response = await page.request.get('/api/scripts?search=
    const data = await response.json();
    expect(data.data.length).toBeGreaterThan(0);
  });

  test('DB-E-007: Dashboard statistics from database', async ({ pa
    await page.goto('/login');
    await page.fill('[name="email"]', 'admin@psscript.com');
    await page.fill('[name="password"]', 'admin123');
    await page.click('button[type="submit"]');

    // Wait for dashboard to load
    await page.waitForSelector('[data-testid="total-scripts"]');
    // Verify stats are displayed (pulled from database)
    const totalScripts = await page.textContent('[data-testid="to
    expect(parseInt(totalScripts || '0')).toBeGreaterThanOrEqual(
  });
});
```

---

4. Performance Tests

Test ID	Description	Threshold	Query
DB-P-001	Simple SELECT	< 50ms	`SELECT * FROM scripts LIMIT 10`
DB-P-002	JOIN query	< 100ms	Scripts with User, Category
DB-P-003	COUNT aggregation	< 30ms	`SELECT COUNT(*) FROM scripts`
DB-P-004	ILIKE search	< 200ms	`WHERE content ILIKE '%Get-%'`
DB-P-005	Vector similarity	< 500ms	pgvector cosine distance
DB-P-006	Paginated list	< 100ms	LIMIT/OFFSET with ORDER
DB-P-007	Analytics aggregation	< 300ms	GROUP BY with calculations

Performance Test Script:

```
 -- Run in psql with \timing on
\timing

-- DB-P-001: Simple SELECT
SELECT * FROM scripts LIMIT 10;

-- DB-P-002: JOIN query
SELECT s.*, u.username, c.name as category_name
FROM scripts s
LEFT JOIN users u ON s.user_id = u.id
LEFT JOIN categories c ON s.category_id = c.id
LIMIT 10;

-- DB-P-003: COUNT aggregation
SELECT COUNT(*) FROM scripts;

-- DB-P-004: ILIKE search
SELECT * FROM scripts WHERE content ILIKE '%Get-%' LIMIT 10;

-- DB-P-005: Vector similarity (if embeddings exist)
SELECT s.title, 1 - (e.embedding <=> '[0.1, 0.2, ...]'::vector) a
FROM script_embeddings e
JOIN scripts s ON e.script_id = s.id
ORDER BY e.embedding <=> '[0.1, 0.2, ...]'::vector
LIMIT 5;

-- DB-P-006: Paginated list
SELECT * FROM scripts ORDER BY created_at DESC LIMIT 20 OFFSET 40

-- DB-P-007: Analytics aggregation
SELECT
  DATE_TRUNC('day', created_at) as day,
  COUNT(*) as script_count,
  AVG(sa.security_score) as avg_security
FROM scripts s
LEFT JOIN script_analysis sa ON s.id = sa.script_id
```

```
WHERE s.created_at >= NOW() - INTERVAL '30 days'
GROUP BY DATE_TRUNC('day', created_at)
ORDER BY day;
```

---

5. Cache Integration Tests

Test ID	Description	Expected Behavior
DB-C-001	Cache hit on script list	Second request returns cached data
DB-C-002	Cache invalidation on create	New script clears `scripts:*` pattern
DB-C-003	Cache invalidation on update	Updated script clears specific key
DB-C-004	Cache invalidation on delete	Deleted script clears cache
DB-C-005	Redis fallback to memory	If Redis down, use in-memory cache
DB-C-006	TTL expiration	Cache entries expire after TTL

Cache Test Script:

```
#!/bin/bash
# Cache integration tests

# Monitor Redis commands
redis-cli MONITOR &
MONITOR_PID=$!

# Make first request (cache miss)
echo "First request (should be cache miss)..."
curl -s "http://localhost:4000/api/scripts?page=1&limit=5" > /dev/

# Make second request (cache hit)
echo "Second request (should be cache hit)..."
curl -s "http://localhost:4000/api/scripts?page=1&limit=5" > /dev/

# Check cache keys
echo "Cache keys:"
redis-cli KEYS "scripts:*"

# Clear cache
echo "Clearing cache..."
curl -s "http://localhost:4000/api/scripts/clear-cache"

# Verify cleared
echo "Cache keys after clear:"
redis-cli KEYS "scripts:*"

kill $MONITOR_PID 2>/dev/null
```

6. Data Integrity Tests

Test ID	Description	Validation
DB-D-001	Foreign key constraints	Delete parent fails if children exist (without CASCADE)
DB-D-002	Unique constraints	Duplicate username/email rejected
DB-D-003	NOT NULL constraints	Required fields enforced
DB-D-004	File hash deduplication	Same content returns same hash
DB-D-005	Password hashing	Plaintext never stored
DB-D-006	JSONB structure	Invalid JSON rejected
DB-D-007	Version uniqueness	Same script+version rejected

Integrity Test Queries:

```sql
-- DB-D-001: Foreign key test
-- Try to delete user with scripts (should fail without CASCADE o
BEGIN;
DELETE FROM users WHERE id = 1;
ROLLBACK;

-- DB-D-002: Unique constraint test
INSERT INTO users (username, email, password_hash, role)
VALUES ('admin', 'new@test.com', 'hash', 'user');
-- Should fail: duplicate username

-- DB-D-003: NOT NULL test
INSERT INTO scripts (description, user_id)
VALUES ('No title', 1);
-- Should fail: title is required

-- DB-D-004: File hash check
SELECT file_hash, COUNT(*)
FROM scripts
WHERE file_hash IS NOT NULL
GROUP BY file_hash
HAVING COUNT(*) > 1;
-- Should return 0 rows (no duplicates)

-- DB-D-005: Password storage verification
SELECT id, username
FROM users
WHERE password_hash NOT LIKE '$2%';
-- Should return 0 rows (all bcrypt hashed)

-- DB-D-007: Version uniqueness
SELECT script_id, version, COUNT(*)
FROM script_versions
GROUP BY script_id, version
HAVING COUNT(*) > 1;
-- Should return 0 rows
```

7. Security Tests

Test ID	Description	Attack Vector
DB-S-001	SQL injection prevention	`'; DROP TABLE users; --`
DB-S-002	Password timing attack	Constant-time comparison
DB-S-003	Account lockout	Brute force protection
DB-S-004	JWT validation	Invalid/expired tokens
DB-S-005	Role-based access	Non-admin accessing admin routes

Security Test Commands:

```
# DB-S-001: SQL injection test (should be safe)
curl -X POST "http://localhost:4000/api/auth/login" \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@test.com'; DROP TABLE users; --","password"
# Should return 401, not execute DROP

# DB-S-003: Account lockout test
for i in {1..10}; do
  curl -s -X POST "http://localhost:4000/api/auth/login" \
    -H "Content-Type: application/json" \
    -d '{"email":"admin@psscript.com","password":"wrongpassword"}
done
# After 5 attempts, should return 423 Locked

# DB-S-004: Invalid JWT test
curl -s "http://localhost:4000/api/scripts" \
  -H "Authorization: Bearer invalid.token.here"
# Should return 401 Unauthorized

# DB-S-005: Non-admin accessing admin route
USER_TOKEN="..." # Regular user token
curl -s -X POST "http://localhost:4000/api/users" \
  -H "Authorization: Bearer $USER_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"username":"hacker","email":"hack@test.com","password":"ha
# Should return 403 Forbidden
```

Test Execution Schedule

Daily (Automated CI/CD)

- [ ] Unit tests (database.test.ts)
- [ ] Basic integration tests (health checks)
- [ ] Cache connectivity

Weekly

- [ ] Full integration test suite
- [ ] Performance benchmarks
- [ ] Security scan

Monthly

- [ ] End-to-end tests
- [ ] Load testing
- [ ] Backup/restore verification

Quarterly

- [ ] Database schema audit
- [ ] Index optimization review
- [ ] Migration verification

Test Results Template

```
## Test Run: [DATE]

### Environment
- PostgreSQL Version: [VERSION]
- Node.js Version: [VERSION]
- Test Runner: Jest/Playwright

### Summary
| Category | Passed | Failed | Skipped |
|----------|--------|--------|---------|
| Unit | X | Y | Z |
| Integration | X | Y | Z |
| E2E | X | Y | Z |
| Performance | X | Y | Z |

### Failed Tests
1. [TEST_ID]: [Description]
   - Error: [Error message]
   - Fix: [Recommended action]

### Performance Results
| Query | Avg Time | Threshold | Status |
|-------|----------|-----------|--------|
| Simple SELECT | Xms | 50ms | ✅/❌ |

### Notes
[Any observations or recommendations]
```

Troubleshooting Guide

Connection Issues

```
# Check PostgreSQL is running
docker-compose ps postgres

# Check connection from host
psql -h localhost -p 5432 -U postgres -d psscript -c "SELECT 1"

# Check from within Docker network
docker exec -it psscript-backend-1 \
  psql -h postgres -p 5432 -U postgres -d psscript -c "SELECT 1"
```

Test Failures

```
# Run specific test with verbose output
cd src/backend && npm test -- --testPathPattern=database.test.ts -

# Check database logs
docker-compose logs postgres | tail -100

# Reset test database
docker-compose down -v
docker-compose up -d postgres redis
```

Performance Issues

```
# Check slow queries
docker exec -it psscript-postgres-1 psql -U postgres -d psscript -
SELECT query, calls, mean_time, total_time
FROM pg_stat_statements
ORDER BY mean_time DESC
LIMIT 10;
"

# Check index usage
docker exec -it psscript-postgres-1 psql -U postgres -d psscript -
SELECT schemaname, tablename, indexname, idx_scan
FROM pg_stat_user_indexes
ORDER BY idx_scan;
"
```

Appendix: Quick Commands

```
# Run all database tests
cd src/backend && npm test -- --testPathPattern=database

# Run specific test section
cd src/backend && npm test -- --testNamePattern="Connection"

# Check test coverage
cd src/backend && npm test -- --coverage --testPathPattern=databas

# Generate HTML report
cd src/backend && npm test -- --reporters=default --reporters=jes
```

Test Plan Version: 1.0 Created: January 15, 2026 Next Review: April 15, 2026

Generated 2026-01-16 23:34 UTC