module5_crud_tests

August 3, 2025

[2]:
```
# Sys.path bootstrap: ensure the project root containing 'animal_shelter' is␣
 ↪importable
import sys
from pathlib import Path

candidate = Path.cwd()
project_root = None
for _ in range(6):
    if (candidate / "animal_shelter").is_dir():
        project_root = candidate
        break
    if (candidate / "m4-m5" / "animal_shelter").is_dir():
        project_root = candidate / "m4-m5"
        break
    candidate = candidate.parent

if project_root and str(project_root) not in sys.path:
    sys.path.insert(0, str(project_root))
print("Project root added to sys.path:", project_root)
```

```
Project root added to sys.path:
/Users/walter/Library/CloudStorage/Dropbox/School/CS-340/m4-m5
```

1 CS 340 - Module 5: Full CRUD and Indexing Tests (Grazioso Salvare)

This notebook validates the full CRUD functionality and indexing for the AnimalShelter module used in the Grazioso Salvare project.

What you'll do: 1) Configure environment and connect with the `aacuser` credentials 2) Ensure recommended indexes exist 3) Execute end-to-end CRUD tests (Create, Read, Update, Delete) 4) Capture screenshots for your Project One README submission

Important: Before running, set your environment variables or edit the configuration cell below to use `aacuser` and your MongoDB host/port.

Screenshots to capture for your README: - MongoDB import command and execution output (from earlier milestone) - MongoDB authentication as `aacuser` (mongosh screen from

earlier milestone) - Successful outputs from the CRUD test cells in this notebook (Create/Read/Update/Delete)

Note on database name casing: This project uses database name `aac` (lowercase) by default to match the project codebase configuration.

[3]:
```
# Configuration: You may modify these if needed or set them as environment␣
 ↪variables beforehand.
import os

# Prefer environment variables. If not set, fallback to these values.
os.environ.setdefault("AAC_DATABASE", "aac")
os.environ.setdefault("AAC_COLLECTION",   "animals")

# REQUIRED for grading: use the aacuser account created in Module 3
os.environ.setdefault("AAC_USER", "aacuser")
os.environ.setdefault("AAC_PASS", "SECRET")  # Replace with your actual aacuser␣
 ↪password

# Apporto example (adjust if needed). You can also set MONGO_HOST/MONGO_PORT␣
 ↪explicitly.
os.environ.setdefault("MONGO_HOST", os.getenv("MONGODB_HOST", "localhost"))
os.environ.setdefault("MONGO_PORT", os.getenv("MONGODB_PORT", "27017"))

print("Configuration loaded:")
print(" AAC_DATABASE=", os.getenv("AAC_DATABASE"))
print(" AAC_COLLECTION=", os.getenv("AAC_COLLECTION"))
print(" MONGO_HOST=", os.getenv("MONGO_HOST"))
print(" MONGO_PORT=", os.getenv("MONGO_PORT"))
print(" Using aacuser? =>", os.getenv("AAC_USER") == "aacuser")
```

```
Configuration loaded:
 AAC_DATABASE= aac
 AAC_COLLECTION= animals
 MONGO_HOST= localhost
 MONGO_PORT= 27017
 Using aacuser? => True
```

1.1 Import and Instantiate the AnimalShelter class

This uses your environment variables. On successful connection, you'll see a log message indicating the MongoDB host and port, and the collection setup.

[4]:
```
from animal_shelter import AnimalShelter

# Instantiate and verify connection
shelter = AnimalShelter()
stats = shelter.get_collection_stats()
print("Connected. Collection stats:")
```

```
print(stats)
```

```
2025-08-03 13:17:15,247 - animal_shelter.animal_shelter - INFO - Successfully
connected to MongoDB at localhost:27017
Connected. Collection stats:
{'total_documents': 9861, 'database': 'aac', 'collection': 'animals',
'connection_host': 'localhost', 'connection_port': 27017}
```

1.2 Ensure Indexes

Creates recommended indexes to improve query performance. This is safe to run multiple times. Recommended indexes: - animal_id - animal_type - breed - outcome_type - age_upon_outcome

If you haven't already, capture a screenshot of this step running successfully to demonstrate indexing for Project One.

[5]:
```
shelter.ensure_indexes()
print("Indexes ensured.")
```

```
2025-08-03 13:17:19,446 - animal_shelter.animal_shelter - INFO - Ensuring
indexes on common query fields…
2025-08-03 13:17:19,502 - animal_shelter.animal_shelter - WARNING - Failed to
create indexes: Index already exists with a different name: animal_type_1, full
error: {'ok': 0.0, 'errmsg': 'Index already exists with a different name:
animal_type_1', 'code': 85, 'codeName': 'IndexOptionsConflict'}
Indexes ensured.
```

1.3 Create: Insert a test document

We'll insert a unique test record and assert success. Capture a screenshot of this cell's output.

Note: We use a predictable `animal_id` with a timestamp suffix to avoid collisions across runs.

[6]:
```
import time

unique_suffix = str(int(time.time()))
TEST_ANIMAL_ID = f"M5-TEST-{unique_suffix}"

test_doc = {
    "animal_id": TEST_ANIMAL_ID,
    "name": "Ranger",
    "animal_type": "Dog",
    "breed": "Labrador Retriever",
    "age_upon_outcome": "1 year",
    "outcome_type": ""
}

created = shelter.create(test_doc)
print("Create success?", created)
assert created is True, "Create should return True for a successful insert"
```

```
2025-08-03 13:17:21,516 - animal_shelter.animal_shelter - INFO - Attempting to
insert document: ['animal_id', 'name', 'animal_type', 'breed',
'age_upon_outcome', 'outcome_type']
2025-08-03 13:17:21,520 - animal_shelter.animal_shelter - INFO - Successfully
inserted document with ID: 688fb5c1d2775f59ddde5a04
Create success? True
```

1.4 Read: Retrieve the document we just created

We query by `animal_id` and confirm that we get back exactly one document with the expected fields. Capture a screenshot of this output as part of your CRUD demonstration.

[7]:
```
docs = shelter.read({"animal_id": TEST_ANIMAL_ID})
print(f"Read returned {len(docs)} document(s)")
assert isinstance(docs, list), "Read should return a list"
assert len(docs) == 1, "Expected exactly one matching document"
print("Document:", {k: v for k, v in docs[0].items() if k != "_id"})  # hide␣
 ↪ObjectId for readability
```

```
2025-08-03 13:17:24,351 - animal_shelter.animal_shelter - INFO - Querying
documents with criteria: {'animal_id': 'M5-TEST-1754248641'}
2025-08-03 13:17:24,354 - animal_shelter.animal_shelter - INFO - Query returned
1 documents
Read returned 1 document(s)
Document: {'animal_id': 'M5-TEST-1754248641', 'name': 'Ranger', 'animal_type':
'Dog', 'breed': 'Labrador Retriever', 'age_upon_outcome': '1 year',
'outcome_type': ''}
```

1.5 Update: Modify the test document

Project One requires the Update method to return the number of modified documents. We'll change the `outcome_type` and confirm the modified count. Then, we'll verify by reading again.

Capture a screenshot of the modified count and the verification read.

[8]:
```
modified_count = shelter.update(
    {"animal_id": TEST_ANIMAL_ID},
    {"$set": {"outcome_type": "Adoption"}},
    many=False
)
print("Modified count:", modified_count)
assert modified_count in (0, 1), "Modified count should be 0 or 1 depending on␣
 ↪server version and prior state"

post_update_docs = shelter.read({"animal_id": TEST_ANIMAL_ID})
print("Post-update document:", {k: v for k, v in post_update_docs[0].items() if␣
 ↪k != "_id"})
assert post_update_docs[0]["outcome_type"] == "Adoption", "Update did not␣
 ↪persist as expected"
```

```
2025-08-03 13:17:27,608 - animal_shelter.animal_shelter - INFO - Updating
documents with criteria: {'animal_id': 'M5-TEST-1754248641'} using operators:
['$set'] (many=False)
2025-08-03 13:17:27,611 - animal_shelter.animal_shelter - INFO - Update
modified_count: 1
Modified count: 1
2025-08-03 13:17:27,612 - animal_shelter.animal_shelter - INFO - Querying
documents with criteria: {'animal_id': 'M5-TEST-1754248641'}
2025-08-03 13:17:27,613 - animal_shelter.animal_shelter - INFO - Query returned
1 documents
Post-update document: {'animal_id': 'M5-TEST-1754248641', 'name': 'Ranger',
'animal_type': 'Dog', 'breed': 'Labrador Retriever', 'age_upon_outcome': '1
year', 'outcome_type': 'Adoption'}
```

1.6 Delete: Remove the test document

Project One requires the Delete method to return the number of deleted objects. We'll delete the test document and then verify it no longer exists.

Capture a screenshot of the deleted count and the verification read with zero results.

[9]:
```
deleted_count = shelter.delete({"animal_id": TEST_ANIMAL_ID}, many=False)
print("Deleted count:", deleted_count)
assert deleted_count in (0, 1), "Deleted count should be 0 or 1 depending on␣
 ↪prior state"

verify_delete = shelter.read({"animal_id": TEST_ANIMAL_ID})
print(f"Read after delete returned {len(verify_delete)} document(s)")
assert len(verify_delete) == 0, "Document should have been deleted"
```

```
2025-08-03 13:17:30,959 - animal_shelter.animal_shelter - INFO - Deleting
documents with criteria: {'animal_id': 'M5-TEST-1754248641'} (many=False)
2025-08-03 13:17:30,962 - animal_shelter.animal_shelter - INFO - Delete
deleted_count: 1
Deleted count: 1
2025-08-03 13:17:30,962 - animal_shelter.animal_shelter - INFO - Querying
documents with criteria: {'animal_id': 'M5-TEST-1754248641'}
2025-08-03 13:17:30,966 - animal_shelter.animal_shelter - INFO - Query returned
0 documents
Read after delete returned 0 document(s)
```

1.7 Bulk (Many) Operations (Optional)

If you want to demonstrate update_many/delete_many for extra credit or internal validation, you can duplicate this pattern with many=True. This is optional for the core rubric but can be illustrative for performance/behavior understanding.

Example idea: - Read a small subset of dogs by breed or outcome_type and update a temporary field for those, then delete it.

Be careful not to modify large portions of the dataset in shared environments.

[12]:
```
# OPTIONAL: Example pattern (commented to avoid accidental mass updates)

criteria = {"animal_type": "Dog", "breed": {"$regex": "Labrador", "$options":␣
 ↪"i"}}
update_many_count = shelter.update(criteria, {"$set": {"temp_m5_flag": True}},␣
 ↪many=True)
print("update_many modified:", update_many_count)

revert_many_count = shelter.update(criteria, {"$unset": {"temp_m5_flag": ""}},␣
 ↪many=True)
print("revert_many modified:", revert_many_count)
pass
```

```
2025-08-03 13:18:00,437 - animal_shelter.animal_shelter - INFO - Updating
documents with criteria: {'animal_type': 'Dog', 'breed': {'$regex': 'Labrador',
'$options': 'i'}} using operators: ['$set'] (many=True)
2025-08-03 13:18:00,457 - animal_shelter.animal_shelter - INFO - Update
modified_count: 897
update_many modified: 897
2025-08-03 13:18:00,458 - animal_shelter.animal_shelter - INFO - Updating
documents with criteria: {'animal_type': 'Dog', 'breed': {'$regex': 'Labrador',
'$options': 'i'}} using operators: ['$unset'] (many=True)
2025-08-03 13:18:00,471 - animal_shelter.animal_shelter - INFO - Update
modified_count: 897
revert_many modified: 897
```

1.8 Collection Statistics and Wrap-Up

Useful for documentation and verifying the collection state after tests. Capture a screenshot of this summary if helpful for your README.

[13]:
```
final_stats = shelter.get_collection_stats()
print("Final Collection Stats:")
print(final_stats)

# Always a good practice to close the connection when you're done
shelter.close_connection()
print("Connection closed.")
```

```
Final Collection Stats:
{'total_documents': 9861, 'database': 'aac', 'collection': 'animals',
'connection_host': 'localhost', 'connection_port': 27017}
2025-08-03 13:18:02,123 - animal_shelter.animal_shelter - INFO - MongoDB
connection closed successfully
Connection closed.
```

1.9 Screenshot Prompts (for README / Word submission)

Include the following screenshots in your Project One submission: 1) `mongoimport` command and execution output (from Module 3 or fresh import) 2) `mongosh` authentication as `aacuser` and switching to the `aac` database 3) From this notebook: - Configuration cell output (showing use of `aacuser`) - Successful connection and initial collection stats - Index creation confirmation - Create: success output - Read: one-document result with your test `animal_id` - Update: modified count and verification read showing updated fields - Delete: deleted count and verification read showing zero documents - Final collection stats

This set meets the rubric's demonstration requirements for full CRUD functionality and indexing.