

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО  
КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ЗВІТ З ЛАБОРАТОРНИХ РОБІТ  
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ  
**«СУЧАСНІ ПАРАДИГМИ ЗБЕРЕЖЕННЯ ДАНИХ»**

Виконав: студент групи КН-22-1(а)

Панцюк К.В.

Перевірив: асист. каф. Андреев П.І.

КРЕМЕНЧУК 2025

## Лабораторна робота №7

### Тема. Транзакції в MongoDB

**Мета роботи:** навчитися створювати, виконувати та керувати транзакціями в MongoDB.

#### Порядок виконання роботи

- 1.Отримати індивідуальний варіант завдання.
- 2.Реалізувати CRUD-операції.
- 3.Використати транзакції до створених CRUD-операцій.
- 4.Обґрунтувати використання транзакцій.

#### Виконання лабораторної роботи

1.Відповідно до бази даних розробленої у 2-ї лабораторній роботі продовжуємо роботу з нею

2.Оскільки раніше були реалізовані CRUD-операцій можна було продовжити роботу з нею проте було прийняте рішення створити нові операції у межах транзакції.

Нижче наведено код для потрібних файлів:

#### **lab7\_transactions.js:**

```
const { MongoClient, ObjectId } = require('mongodb');  
const uri = 'mongodb://127.0.0.1:27017/electronicsStore';
```

```
const productRepository = require('./productRepository');  
const userRepository = require('./userRepository');  
const orderRepository = require('./orderRepository');
```

```
// === ОСНОВНА ФУНКЦІЯ ТРАНЗАКЦІЇ ===
```

```
async function createOrderWithTransaction(userId, productId, quantity) {  
  const client = new MongoClient(uri, {  
    writeConcern: { w: "majority" },  
    readConcern: { level: "snapshot" },  
  });
```

```

try {
  await client.connect();
  const db = client.db();
  const session = client.startSession();

  try {
    // === ПОЧАТОК ТРАНЗАКЦІЇ ===
    session.startTransaction();

    // === READ: Отримання продукту (всередині транзакції) ===
    const product = await db.collection('products').findOne(
      { _id: new ObjectId(productId) },
      { session }
    );

    if (!product || product.stock < quantity) {
      throw new Error('Недостатньо товару на складі');
    }

    const initialStock = product.stock; // зберігаємо початкову кількість

    // === READ: Отримання користувача (всередині транзакції) ===
    const user = await db.collection('users').findOne(
      { _id: new ObjectId(userId) },
      { session }
    );

    if (!user) {
      throw new Error('Користувача не знайдено');
    }
  }
}

```

```
}
```

```
// === CREATE: Створення замовлення (всередині транзакції) ===
```

```
const orderResult = await orderRepository.createOrder(  
  db,  
  userId,  
  product,  
  quantity,  
  session  
);
```

```
// === UPDATE: Зменшення запасу товару (всередині транзакції) ===
```

```
const updateResult = await db.collection('products').updateOne(  
  { _id: new ObjectId(productId) },  
  { $inc: { stock: -quantity } },  
  { session }  
);
```

```
if (updateResult.modifiedCount === 0) {  
  throw new Error('Не вдалося оновити запас товару');  
}
```

```
// === ПІДТВЕРДЖЕННЯ ТРАНЗАКЦІЇ ===
```

```
await session.commitTransaction();  
console.log('Транзакція успішна. Замовлення створено:',  
orderResult.insertedId);
```

```
// === READ: Перевірка оновленого запасу товару після транзакції ===
```

```
const updatedProduct = await db.collection('products').findOne(  
  { _id: new ObjectId(productId) }
```

```
);
```

```
const finalStock = updatedProduct.stock;
```

```
// === Вивід порівняння запасу ===
```

```
console.log(`Було на складі: ${initialStock}`);
```

```
console.log(`Замовлено: ${quantity}`);
```

```
console.log(`Залишилось після транзакції: ${finalStock}`);
```

```
if (finalStock === initialStock - quantity) {
```

```
    console.log('Підтверджено: запас товару зменшено коректно.');
```

```
} else {
```

```
    console.warn('Увага: запас товару не відповідає очікуваному  
значенню.');
```

```
}
```

```
return { success: true, orderId: orderResult.insertedId };
```

```
} catch (error) {
```

```
    console.error('Помилка транзакції:', error.message);
```

```
    await session.abortTransaction();
```

```
    return { success: false, error: error.message };
```

```
} finally {
```

```
    await session.endSession();
```

```
}
```

```
} finally {
```

```
    await client.close();
```

```
}
```

```
}
```

```
// === ФУНКЦІЯ ДЕМОНСТРАЦІЇ З ТЕСТОВИМИ ДАНИМИ ===
```

```

async function demonstrateTransaction() {
  console.log('=== Демонстрація транзакції ===');

  const testUser = await userRepository.create({
    firstName: 'Тестовий',
    lastName: 'Користувач',
    email: 'test@example.com',
    password: 'password123',
    createdAt: new Date(),
    lastLogin: new Date()
  });

  const testProduct = await productRepository.create({
    name: 'Тестовий продукт',
    description: 'Опис тестового продукту',
    sku: 'TEST-PROD-001',
    categoryId: new ObjectId(),
    brand: 'Тестовий бренд',
    price: 1000,
    stock: 10,
    isActive: true,
    createdAt: new Date(),
    updatedAt: new Date()
  });

  if (!testUser.success || !testProduct.success) {
    console.error('Помилка при створенні тестових даних');
    return;
  }
}

```

```

const result = await createOrderWithTransaction(
  testUser.insertedId,
  testProduct.insertedId,
  2
);

if (result.success) {
  console.log('Замовлення створено з ID:', result.orderId);
} else {
  console.error('Помилка при створенні замовлення:', result.error);
}
}

// === ГОЛОВНА ФУНКЦІЯ ЗАПУСКУ ===
async function main() {
  try {
    await demonstrateTransaction();
    console.log('\n=== ДЕМОНСТРАЦІЯ ЗАВЕРШЕНА ===');
  } catch (error) {
    console.error('Помилка демонстрації:', error);
  }
}

module.exports = {
  createOrderWithTransaction,
  demonstrateTransaction,
  main
};

if (require.main === module) {

```

```
main().catch(console.error);  
}
```

### **orderRepository.js**

```
const { ObjectId } = require('mongodb');  
  
async function createOrder(db, userId, product, quantity, session) {  
  const order = {  
    userId: new ObjectId(userId),  
    items: [  
      {  
        productId: new ObjectId(product._id),  
        name: product.name,  
        price: product.price,  
        quantity: quantity,  
      },  
    ],  
    totalAmount: product.price * quantity,  
    finalAmount: product.price * quantity,  
    status: 'processing',  
    createdAt: new Date(),  
    updatedAt: new Date(),  
    paymentStatus: 'pending',  
    paymentMethod: 'cash_on_delivery',  
    shippingAddress: {  
      street: 'вул. Прикладна, 1',  
      city: 'Кременчук',  
      postalCode: '39600',  
      country: 'Україна'  
    }  
  };  
};
```



```
const result = await db.collection('orders').insertOne(order, { session });  
return result;  
}
```

```
module.exports = {  
  createOrder  
};
```

### **productRepository.js**

```
const BaseRepository = require('./baseRepository');  
const { ObjectId } = require('mongodb');
```

```
class ProductRepository extends BaseRepository {  
  constructor() {  
    super('products');  
  }
```

```
  async findByCategory(categoryId, page = 1, limit = 10) {  
    return this.paginate(  
      { categoryId: new ObjectId(categoryId), isActive: true },  
      page,  
      limit,  
      { name: 1 }  
    );  
  }
```

```
  async searchProducts(searchTerm, page = 1, limit = 10) {  
    const filter = {  
      $and: [  
        { isActive: true },
```

```

    {
      $or: [
        { name: { $regex: searchTerm, $options: 'i' } },
        { description: { $regex: searchTerm, $options: 'i' } },
        { brand: { $regex: searchTerm, $options: 'i' } },
        { tags: { $in: [new RegExp(searchTerm, 'i')] } }
      ]
    }
  ]
};

```

```

    return this.paginate(filter, page, limit, { score: { $meta: "textScore" } });
  }

```

```

async updateStock(productId, quantity) {
  const collection = await this.getCollection();
  return collection.updateOne(
    { _id: new ObjectId(productId) },
    { $inc: { stock: quantity } }
  );
}
}

```

```

module.exports = new ProductRepository();

```

### **userRepository.js**

```

const BaseRepository = require('./baseRepository');
const { ObjectId } = require('mongodb');

```

```

class UserRepository extends BaseRepository {
  constructor() {

```

```

    super('users');
  }

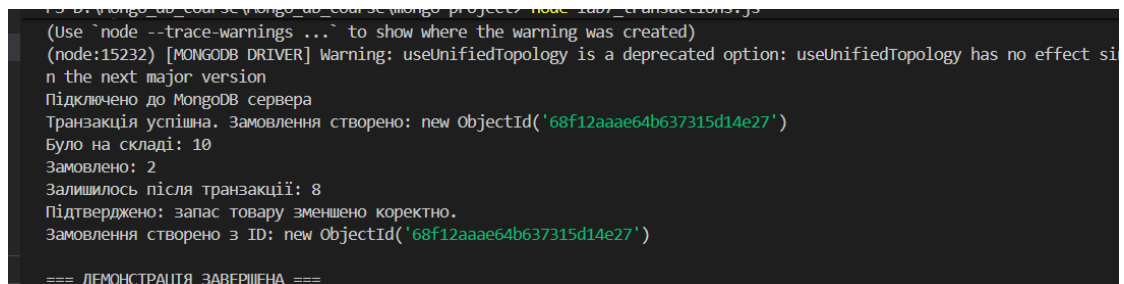
  async findByEmail(email) {
    const collection = await this.getCollection();
    return collection.findOne({ email });
  }

  async addToWishlist(userId, productId) {
    const collection = await this.getCollection();
    return collection.updateOne(
      { _id: new ObjectId(userId) },
      { $addToSet: { wishlist: new ObjectId(productId) } }
    );
  }

  async updateLastLogin(userId) {
    return this.update(userId, { lastLogin: new Date() });
  }
}

module.exports = new UserRepository();

```



```

C:\Users\user> node --trace-warnings ... (Use `node --trace-warnings ...` to show where the warning was created)
(node:15232) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect si
n the next major version
Підключено до MongoDB сервера
Транзакція успішна. Замовлення створено: new ObjectId('68f12aaae64b637315d14e27')
Було на складі: 10
Замовлено: 2
Залишилось після транзакції: 8
Підтверджено: запас товару зменшено коректно.
Замовлення створено з ID: new ObjectId('68f12aaae64b637315d14e27')

=== ДЕМОНСТРАЦІЯ ЗАВЕРШЕНА ===

```

Рисунок 7.1–Результат виводу

## Обґрунтування використання транзакцій

У MongoDB транзакція дозволяє виконати кілька дій як одну операцію. Це потрібно, коли важливо, щоб усі зміни відбулися разом або не відбулися взагалі.

У нашому випадку:

- Ми створюємо замовлення.
- Одночасно зменшуємо кількість товару на складі.

Якщо щось піде не так (наприклад, товару недостатньо або користувача не знайдено), усі зміни скасовуються, і база даних залишається в правильному стані. Це захищає нас від помилок і забезпечує надійність.

Контрольні питання

1. Що таке транзакція в MongoDB?

Транзакція – це набір дій, які виконуються разом. Якщо одна з них не спрацює, всі інші теж скасовуються.

2. З яких версій MongoDB підтримуються багатодокументні транзакції?

- З версії 4.0 – для replica set
- З версії 4.2 – для sharded cluster

3. В яких середовищах MongoDB працюють транзакції?

Транзакції працюють у:

- replica set (сервери з копіями даних)
- sharded cluster (розподілені дані по кількох серверах)

У звичайному (standalone) режимі — не працюють.

4. Який рівень ізоляції транзакцій використовується в MongoDB?

MongoDB використовує Snapshot Isolation — це означає, що всі дії в транзакції бачать дані такими, якими вони були на початку транзакції, навіть якщо інші користувачі щось змінюють.

5. Як створити транзакцію у mongosh?

// Створення сесії

```
session = db.getMongo().startSession();
```

// Початок транзакції

```
session.startTransaction();
```

```
// Дії в межах транзакції  
db.orders.insertOne(..., { session });  
  
// Завершення транзакції  
session.commitTransaction();  
  
// Якщо щось пішло не так  
session.abortTransaction();
```

6. Які обмеження і недоліки має використання транзакцій у MongoDB?

- Транзакції можуть бути повільнішими, ніж звичайні операції.
- Вони мають обмеження по часу виконання.
- Не працюють у standalone-сервері.
- Потрібно уважно обробляти помилки.

7. Які приклади застосування транзакцій у MongoDB?

- Створення замовлення + оновлення товару
- Переказ грошей між рахунками
- Видалення користувача + його замовлень
- Масове оновлення даних, яке має бути узгодженим