تقرير المحاضرة الرابعة

جامعة إب / كلية العلوم التطبيقية / قسم تقنية المعلومات

المقرر: هندسة البرمجيات

الطالب: أمين الصلاحي

المدرس: م/ مالك المصنف

دليل استراتيجي لأنظمة قواعد البيانات في إطار عمل Django

الجزء الأول: المفاهيم الأساسية ونظرة عامة استراتيجية

ملخص تنفيذي

في عالم تطوير تطبيقات الويب باستخدام إطار عمل Django، يعد اختيار نظام قاعدة البيانات قراراً استراتيجياً يؤثر على الأداء، وسلامة البيانات، وقابلية التوسع على المدى الطويل. يقدم هذا التقرير تحليلاً شاملاً لأكثر قواعد البيانات شيوعاً ودعماً رسمياً من Django: PostgreSQL، وMySQL، وSQLite.

خلص التحليل إلى أن PostgreSQL يعتبر الخيار الأفضل والأكثر موثوقية للتطبيقات الجادة والمصممة للإنتاج. فهو يقدم مجموعة غنية من الميزات المتقدمة التي تتكامل بسلاسة مع Django، مما يجعله الخيار الاستراتيجي للمشاريع التي تتوقع نمواً أو تتطلب معالجة معقدة للبيانات. من ناحية أخرى، يعد MySQL حلاً موثوقاً به للغاية، خاصة في التطبيقات التي تعتمد بشكل كبير على عمليات القراءة وتتطلب معالجة سريعة لعدد كبير من المعاملات. أما SQLite، فبفضل طبيعته البسيطة وغير المركزية، فهو خيار ممتاز للتطوير المحلي السريع والاختبار، ولكن لا ينصح به على الإطلاق لبيئات الإنتاج بسبب قيوده الحرجة في التزامن وقابلية التوسع.[1]

مقدمة: نظام ORM في Django كطبقة تجريدية

يعتمد إطار عمل Django على فلسفة "البطاريات المدمجة" (batteries-included)، حيث يوفر مجموعة متكاملة من الأدوات التي تسهل عملية التطوير. من أبرز هذه الأدوات هو نظام ORM (Object-Relational Mapper) المدمج.[4] يعمل هذا النظام كطبقة تجريدية موحدة تسمح للمطورين بالتعامل مع البيانات في قاعدة البيانات باستخدام كائنات Python بدلاً من كتابة استعلامات SQL خام.[5] بفضل هذه الطبقة، يمكن للمطور استخدام نفس الواجهة البرمجية (API) للعمل مع قواعد بيانات مختلفة، مما يقلل من التعقيد ويجعل تغيير قاعدة البيانات أمراً ممكناً إلى حد كبير.[1]

يدعم Django رسمياً مجموعة من قواعد البيانات الأكثر شيوعاً[6]:

- PostgreSQL

- MariaDB
- MySQL
- Oracle
- SQLite

بالإضافة إلى ذلك، توفر جهات خارجية واجهات برمجية تسمح لـ Django بالتكامل مع قواعد بيانات أخرى مثل Sybase SQL Anywhere، وIBM DB2، وMicrosoft SQL Server، وحتى قواعد بيانات NoSQL مثل MongoDB.[7]

على الرغم من أن نظام ORM في Django مصمم لتوحيد تجربة التعامل مع البيانات، إلا أن التوثيق الرسمي نفسه يقر بأن "قواعد البيانات ليست متماثلة".[6] هذا يعني أن طبقة التجريد، على الرغم من قوتها، لا يمكنها أن تخفي بشكل كامل الاختلافات الجوهرية في سلوك قواعد البيانات الأساسية، مثل نماذج الأمان، أو آليات التزامن، أو خصائص الأداء الفريدة لكل منها. لفهم هذه الفروق، يجب على المطورين الخبراء امتلاك معرفة عميقة بالقاعدة التي يختارونها. يسعى هذا التقرير إلى توضيح هذه الفروقات الدقيقة لتمكين اتخاذ قرار مستنير.

الجزء الثاني: مقارنة متعمقة بقواعد البيانات الأساسية المدعومة

2.1. PostgreSQL: القوة المؤسسية والتميز في الميزات

يُعتبر PostgreSQL الخيار المفضل والقياسي للتطبيقات الجادة التي تتطلب موثوقية وقابلية للتوسع.[5] من أبرز نقاط قوته:

- **الالتزام بـ ACID:** يشتهر PostgreSQL بالالتزام الصارم بخصائص ACID (الذرية، الاتساق، العزل، والموثوقية)، مما يضمن سلامة البيانات حتى في حالات فشل النظام.[3]
- **الميزات المتقدمة:** يتميز بميزات فريدة يدعمها Django بشكل أصلي، مثل حقول JSONB لتخزين البيانات غير المنظمة، وحقول `ArrayField` لتخزين المصفوفات، وقدرات البحث الكامل عن النصوص (Full-Text Search).[1]
- **الأداء:** يتفوق PostgreSQL في التعامل مع الاستعلامات المعقدة والتحليلات البيانية، مما يجعله مثالياً لتطبيقات تحليل البيانات والذكاء الاصطناعي.[2]

التكامل مع Django

يتطلب PostgreSQL وجود أحد مشغلات Python مثل psycopg (أو psycopg2.6 الموصى به حالياً). للمطورين تحسين أداء تطبيقاتهم عبر إعدادات متقدمة في قاموس `OPTIONS` ضمن إعدادات `DATABASES`. على سبيل المثال، يمكن التحكم في مستوى العزل للمعاملات (isolation level) ليصبح `SERIALIZABLE` للحالات المتقدمة.[6] كما أن استخدام خاصية

`CONN_MAX_AGE` في إعدادات الاتصال يقلل من الحمل الزائد الناتج عن إعادة إنشاء اتصال مع قاعدة البيانات في كل طلب.[6]

يُعد اختيار PostgreSQL قراراً استراتيجياً من شأنه أن يجهز التطبيق للمستقبل، لأنه يسمح للمطورين بإضافة هياكل بيانات مرنة مثل حقول JSONB دون الحاجة إلى التبديل إلى قاعدة بيانات NoSQL مستقلة.[5] هذا يخفف من التعقيد المعماري ويقلل من الحاجة لإعادة هندسة التطبيق لاحقاً. ومع ذلك، يجب الانتباه إلى أن هذه الإعدادات المتقدمة، مثل رفع مستوى العزل، تتطلب أن يكون التطبيق مستعداً للتعامل مع الاستثناءات التي قد تحدث.[6] هذا يؤكد أن نظام ORM لا يعفي المطور من فهم سلوكيات قاعدة البيانات الأساسية لضمان عمل التطبيق بشكل صحيح.

مثال عملي: استخدام حقول JSONField والبحث الكامل

يمكن تعريف نموذج Django يستخدم حقل JSONField للاستفادة من قدرات PostgreSQL المتقدمة:

Python

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=255)
    author = models.CharField(max_length=100)
    metadata = models.JSONField()
```

ولتنفيذ بحث كامل عن النصوص، يمكن استخدام الأدوات المدمجة في Django كما يلي[5]:

Python

```
from django.contrib.postgres.search import SearchQuery, SearchRank, SearchVector
from myapp.models import Book

# Combine multiple fields into a single search vector
vector = SearchVector('title', 'author')

# Create a search query for 'Django'
query = SearchQuery('Django')

# Perform the search and order results by relevance
books = Book.objects.annotate(rank=SearchRank(vector, query)).order_by('-rank')
```

2.2. MySQL / MariaDB: حصان العمل الموثوق به

يُعتبر MySQL، وشقيقه مفتوح المصدر MariaDB، من أكثر قواعد البيانات العلائقية شيوعاً، ويتمتعان بسمعة ممتازة كخيار موثوق به وذو أداء عالٍ، خاصة في التطبيقات التي تتضمن عدداً كبيراً من عمليات القراءة (read-heavy) مثل المنصات الاجتماعية ومواقع التجارة الإلكترونية.[2] يتميز MySQL بسرعته العالية في معالجة المعاملات البسيطة، كما أن نظامه البيئي الواسع والمجتمع الداعم يوفران مصادر وأدوات وفيرة للمطورين.[2]

التكامل مع Django

هو الخيار الموصى به.6 من أهم النقاط الواجب mysqlclient حيث يُعتبر ،MySQL مشغلاً لتوصيله بـ Django يتطلب مراعاتها هي استخدام محرك التخزين

InnoDB، الذي يضمن سلامة المعاملات (ACID) وقدرات استعادة البيانات بعد الأعطال.[11] يمكن تحديد محرك التخزين الافتراضي باستخدام

init_command في إعدادات OPTIONS للتأكد من استخدام InnoDB.[6]

من النقاط الدقيقة التي يجب فهمها هي إعدادات الترتيب اللغوي (Collation) في MySQL.[6] يستخدم Django ترميز UTF-8 افتراضياً، ويستخدم MySQL في المقابل ترميز

utf8mb4_0900_ai_ci افتراضياً، وهو ترميز غير حساس لحالة الأحرف. هذا يعني أن قيمتين مثل "Fred" و "freD" تعتبران متساويتين من وجهة نظر قاعدة البيانات. هذه الميزة يمكن أن تتسبب في مشكلات خفية، حيث قد يفشل قيد التفريد (unique constraint) عند محاولة إدخال قيمتين تختلفان فقط في حالة الأحرف.[6] هذا الموقف مثال جيد يوضح أن طبقة التجريد في Django لا تستطيع إخفاء كل سلوكيات قاعدة البيانات، ويجب على المطور أن يدرك أهمية إعداد

db_collation بشكل صريح على مستوى النموذج عند الحاجة إلى حساسية حالة الأحرف.[6]

مثال عملي: إعداد settings.py

على النحو التالي: 4 settings.py في ملف DATABASES يجب تعديل قاموس ،MySQL بـ Django لتوصيل

Python

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'your_database_name',
        'USER': 'your_username',
        'PASSWORD': 'your_password',
        'HOST': 'localhost',
```

```
        'PORT': '3306',
    }
}
```

2.3. SQLite: الخيار الافتراضي للتطوير والاختبار

يُعد SQLite هو قاعدة البيانات الافتراضية المضمنة في Django.[2] يتميز بسهولة إعداده الفائقة وعدم حاجته إلى خادم منفصل، حيث يتم تخزين قاعدة البيانات بالكامل في ملف واحد. هذه الطبيعة الخفيفة والمدمجة تجعله الخيار المثالي للبدء في المشاريع، أو للتطبيقات الفردية، أو للتطبيقات التي لا تتطلب اتصالاً بالشبكة.[3]

التكامل مع Django

ولا يتطلب أي إعدادات إضافية باستثناء تحديد مسار ملف قاعدة البيانات في Django، مهيئاً مسبقاً في SQLite يأتي حيث يمكن تكوين إطار العمل Django، أداة أساسية لتشغيل الاختبارات في SQLite الإعدادات.13 بالإضافة إلى ذلك، يعد لاستخدام قاعدة بيانات

`in-memory` (في الذاكرة)، مما يجعل الاختبارات سريعة جداً ويضمن بيئة نظيفة في كل مرة يتم فيها التشغيل.[14]

مثال عملي: استخدام قاعدة بيانات في الذاكرة للاختبارات

في الذاكرة لتسريع الاختبارات، وهي ممارسة موصى بها بشدة في SQLite لاستخدام قاعدة بيانات Django يمكن تكوين بيئات التطوير 14:

Python

```
# settings.py
import sys

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'your_production_database_name',
        #... Other production settings
    }
}

# Override database for testing to use in-memory SQLite
if 'test' in sys.argv:
    DATABASES['default'] = {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': ':memory:',
    }
```

القيود الحرجة: لماذا ليس للإنتاج؟

غير مناسب تماماً للاستخدام في بيئات الإنتاج.3 SQLite على الرغم من مميزاته، يجب التأكيد على أن

- **مشاكل التزامن:** يفرض SQLite قفلاً على مستوى قاعدة البيانات بالكامل أثناء عمليات الكتابة.[3] هذا يعني أنه لا يمكن لأكثر من مستخدم واحد الكتابة في قاعدة البيانات في نفس الوقت، مما يجعله نقطة اختناق فورية في أي تطبيق ويب متعدد المستخدمين. قد يؤدي ذلك إلى أخطاء "Database is Locked" المتكررة.[6]
- **قيود الترحيلات (Migrations):** تفتقر SQLite إلى الدعم المدمج لتعديل المخطط (schema). لذا، عندما يقوم Django بتطبيق الترحيلات، فإنه يحاكي هذه العملية عن طريق إنشاء جدول جديد، ونسخ البيانات، ثم حذف الجدول القديم وإعادة تسمية الجدول الجديد. هذه العملية بطيئة وقد تكون عرضة للخطأ.[15]
- **اعتبارات النشر:** عند نشر تطبيق يستخدم SQLite على منصات مثل Heroku، والتي تستخدم أنظمة ملفات سريعة الزوال (ephemeral filesystems)، فإن ملف قاعدة البيانات سيُفقد بالكامل عند كل إعادة تشغيل للتطبيق.[1]

الجزء الثالث: التحليل المقارن وإطار صنع القرار

3.1. تحليل مقارن متعدد العوامل

يقدم الجدول التالي مقارنة شاملة بين قواعد البيانات الثلاث بناءً على عوامل رئيسية تؤثر على اختيار المطور:

الميزة (Feature)	SQLite	MySQL	PostgreSQL
التزامن (Concurrency)	**محدود للغاية.** قفل على مستوى قاعدة البيانات، غير مناسب للكتابة المتعددة.[3]	**جيد جداً.** قفل على مستوى الصفوف، أداء ممتاز في المعاملات.[3]	**ممتاز.** تحكم متقدم في التزامن (MVCC)، أداء عالي في عمليات الكتابة والقراءة المتزامنة.[2]
الميزات المتقدمة	**أساسي.** لا يدعم العديد من الميزات	**قياسي.** يدعم معظم الميزات العلائقية.[2]	**غني جداً.** يدعم أنواع البيانات المتقدمة (JSONB، ArrayField) والبحث الكامل

	المتقدمة.[1]		عن النصوص.[1]
أداء القراءة	**سريع.** أداء ممتاز في عمليات القراءة البسيطة.[1]	**سريع جداً.** أداء متفوق في التطبيقات التي تعتمد على القراءة.[2]	**جيد.** أسرع من MySQL في الاستعلامات المعقدة.[2]
أداء الاستعلامات المعقدة	**ضعيف.** لا يدعم بعض الاستعلامات المتقدمة.[1]	**جيد.** يمكن أن يتأثر أداؤه بالاستعلامات المعقدة.[2]	**متفوق.** الأفضل في التعامل مع الاستعلامات المعقدة والتحليلات.[2]
سهولة الإعداد	**الأسهل.** لا يتطلب أي إعداد أو خادم.[2]	**سهل نسبياً.** يتطلب تثبيت خادم ومشغل.[10]	**يتطلب جهداً أكبر.** يتطلب تثبيت خادم وتهيئة أولية.[5]

3.2. إطار عمل لصنع القرار: اختيار قاعدة البيانات المناسبة

يعتمد قرار اختيار قاعدة البيانات بشكل أساسي على حجم المشروع ونوع استخدامه. يُوصى بالاعتماد على الإطار التالي لاتخاذ قرار مستنير:

- **مشروع صغير/شخصي أو مرحلة التطوير:**
  - **حالة الاستخدام:** مدونة شخصية، موقع Portfolio، أو نموذج أولي لتطبيق.
  - **قاعدة البيانات الموصى بها:** SQLite.
  - **التبرير:** سهولة الإعداد الفائقة، وعدم وجود حاجة إلى خادم، وسرعة التطوير والاختبار. لا يتطلب أي تكلفة أو وقت إضافي للإعداد.[1]
- **تطبيق ويب متوسط الحجم/تجارة إلكترونية:**
  - **حالة الاستخدام:** تطبيق يحتاج إلى عدد كبير من المستخدمين، وعدد هائل من عمليات القراءة، ومعاملات متكررة.
  - **قاعدة البيانات الموصى بها:** MySQL.
  - **التبرير:** أداء ممتاز في التطبيقات التي تعتمد على القراءة، وقدرة عالية على التعامل مع المعاملات.[2] يوفر

MySQL أيضاً نظاماً بيئياً ناضجاً وأدوات مساعدة.[2]

- **تطبيق مؤسسي/تحليل بيانات أو نمو متوقع:**
  - **حالة الاستخدام:** نظم معلومات معقدة، تطبيقات مصرفية أو صحية، تطبيقات تحليل البيانات، أو مشاريع تتوقع نمواً كبيراً في المستقبل.
  - **قاعدة البيانات الموصى بها:** PostgreSQL.
  - **التبرير:** موثوقية عالية، التزام بسلامة البيانات، ودعم لميزات متقدمة (مثل `JSONField` و`PostGIS` الجغرافي) التي تجعل التطبيق قابلاً للتوسع ومعالجة هياكل البيانات المعقدة.[2]

الجزء الرابع: الخاتمة والتوصيات العملية

4.1. أفضل الممارسات لإدارة قواعد البيانات في Django

بغض النظر عن قاعدة البيانات المختارة، هناك ممارسات أساسية يجب اتباعها لضمان الأداء الجيد واستقرار التطبيق:

- **الاتصالات الدائمة:** يُنصح بضبط متغير `CONN_MAX_AGE` على قيمة موجبة (بالثواني) في إعدادات `DATABASES`. هذا يسمح لـ Django بإعادة استخدام اتصالات قاعدة البيانات القائمة بدلاً من إنشاء اتصال جديد لكل طلب HTTP، مما يقلل من الحمل الزائد ويحسن الأداء بشكل ملحوظ.[6]
- **إدارة الترحيلات:** يعد استخدام أوامر Django للترحيلات (`makemigrations` و `migrate`) حجر الزاوية في إدارة مخطط قاعدة البيانات بشكل احترافي. هذه الأوامر توفر طريقة متسقة وآمنة لتعديل الجداول، بغض النظر عن قاعدة البيانات المستخدمة.[15]

4.2. الحكم النهائي

بعد تحليل شامل للمزايا والعيوب، والاستخدامات، وأداء قواعد البيانات الرئيسية، يمكن القول بثقة أن **PostgreSQL هو الخيار الافتراضي الموصى به لأي تطبيق Django جاد مصمم للإنتاج**. إن استثمار الوقت في الإعداد الأولي لـ PostgreSQL هو ثمن بسيط مقابل المكاسب الكبيرة في الموثوقية، وقوة الميزات، وقابلية التوسع التي يقدمها على المدى الطويل. بالنسبة لأي مشروع يتوقع نمواً أو قد يحتاج إلى التعامل مع هياكل بيانات معقدة في المستقبل، فإن PostgreSQL يوفر الأساس الأكثر استراتيجية ومتانة.