

مقارنة أنواع قواعد البيانات الشائعة

MySQL, SQL Server, MongoDB, PostgreSQL, SQLite, Oracle

المجموعة: C

التاريخ: 10 أغسطس 2025

الاسم: زياد عبده احمد محمد ابوراس

الأستاذ: مالك المصنف

وصف التكليف:

مقارنة بين أنواع قواعد البيانات الشائعة (MySQL, SQL Server, MongoDB, PostgreSQL, SQLite, Oracle) من حيث الميزات والعيوب والاستخدامات وطرق الاتصال باستخدام Django



| مقدمة حول قواعد البيانات

? ما هي قاعدة البيانات؟

قاعدة البيانات هي مجموعة منظمة من البيانات المخزنة والمتاح إليها بطريقة إلكترونية.

تعتبر قواعد البيانات أساسية في عصر التقنية الحديثة حيث تقوم بدور المخزن المركزي للمعلومات التي تستخدمها التطبيقات والخدمات.

≡ أنواع قواعد البيانات

قواعد البيانات غير العلائقية (NoSQL)

- مرونة عالية في تخزين البيانات
- لا تتطلب هيكل بيانات ثابت
- مناسبة للبيانات الكبيرة والمعقدة
- أمثلة: MongoDB, Redis, Cassandra, Couchbase

قواعد البيانات العلائقية (SQL)

- تستخدم الجداول وال العلاقات لتخزين البيانات
- هيكل ثابت للبيانات (Schema)
- تدعم العلاقات المعقدة
- أمثلة: MySQL, PostgreSQL, SQL Server, Oracle, SQLite

★ أهمية اختيار قاعدة البيانات المناسبة

يعتمد اختيار قاعدة البيانات المناسبة على متطلبات المشروع، حجم البيانات، نوع البيانات، المرونة المطلوبة، وقابلية التوسيع.



في الشرائح التالية سنناقش ميزات وعيوب كل نوع من أنواع قواعد البيانات الشائعة

MySQL: نظرة عامة



نبذة تعريفية

نظام إدارة قواعد بيانات علائقى (SQL) مفتوح المصدر ومجاني، يعتبر من أكثر أنظمة قواعد البيانات استخداماً في العالم.

المميزات

- تكلفة منخفضة (مجاني ومفتوح المصدر)
- إعداد سريع وتكوين بسيط
- يتواافق مع جميع أنظمة التشغيل

- سهل التعلم والاستخدام مع وثائق ممتازة
- أداء عالي مع تطبيقات الويب الصغيرة والمتوسطة
- مجتمع مستخدمين كبير وداعم

العيوب

- ليس الأنسب للمعاملات المعقدة
- قابلية توسيع محدودة للبيانات الضخمة جداً

- محدود في دعم بعض الميزات المتقدمة
- أقل استقراراً من Oracle للتطبيقات الحرجية

شركات تستخدم MySQL

YouTube

Facebook

Yahoo

Twitter

Shopify

Uber

حالات الاستخدام

- تطبيقات الويب والمواقع الإلكترونية
- أنظمة إدارة المحتوى (CMS)
- التطبيقات التجارية المتوسطة
- مواقع التجارة الإلكترونية
- أنظمة تسجيل المعاملات (OLTP)



قاعدة بيانات علائقية

نظرة عامة: PostgreSQL |

نوع القاعدة والنظرية العامة

قاعدة بيانات علائقية (SQL) مفتوحة المصدر تتميز بأنها متقدمة للغاية، موثوقة، وتدعم ميزات إضافية متقدمة مقارنة بباقي قواعد البيانات العلائقية.

العيوب

- منحنى تعلم أكثر تعقيداً مقارنة بـ MySQL
- يستهلك موارد أكثر للعمليات البسيطة
- إعداد وتكوين أكثر تعقيداً للمبتدئين
- سرعة أقل في عمليات الكتابة البسيطة مقارنة ببعض الخيارات الأخرى
- يحتاج عادة لضبط الأداء (Performance Tuning)

المميزات

- دعم كامل لأنواع المتقدمة من البيانات (JSON, Arrays, UUID)
- أداء ممتاز مع الاستعلامات المعقدة
- نظام إدارة متزامن للمعاملات (MVCC)
- امتثال عالي لمعايير SQL
- قابلية للتوسيع وإضافة امتدادات (Extensions)
- ميزات أمان متقدمة مثل التشفير على مستوى العمود

شركات تستخدم PostgreSQL

- Instagram - لتخزين بيانات المستخدمين والصور
- Spotify - لتخزين البيانات الخاصة بالمستخدمين والموسيقى
- Reddit - لتخزين محتوى المنتديات والبيانات الخاصة بالمستخدمين
- Skype - للرسائل والبيانات الأساسية
- Apple - تستخدمه في بعض أجزاء خدماتها

حالات الاستخدام المثلية

- أنظمة تحتاج لسلامة البيانات (Data Integrity) بشكل صارم
- تطبيقات تحليل البيانات الكبيرة
- أنظمة المعلومات الجغرافية (GIS)
- تطبيقات التمويل والبنوك التي تحتاج لدقة عالية
- أنظمة تخزين البيانات الموزعة

SQLite: نظرة عامة |

نوع قاعدة البيانات

قاعدة بيانات مضمونة (Embedded Database) - تخزن جميع البيانات في ملف واحد

العيوب !

- لا تتناسب التطبيقات الضخمة والبيانات الكبيرة
- محدودة في الاستخدام المتعدد المتزامن
- محدودية في بعض ميزات قواعد البيانات المتقدمة
- أقل أماناً مقارنة بقواعد البيانات المؤسسية
- تقل كفاءتها مع زيادة حجم البيانات

المميزات ✓

- سهلة الاستخدام والتكامل في التطبيقات
- لا تتطلب سيرفر منفصل للتشغيل
- خفيفة الوزن وسريعة الأداء
- مثالية للتطبيقات المحمولة وسطح المكتب
- ملف واحد يحتوي على كل البيانات
- مجانية تماماً - بدون ترخيص

التطبيقات والاستخدامات

- متصفح Firefox
- تطبيقات Apple المختلفة
- غالبية التطبيقات المحمولة

- نظام أندرويد
- متصفح Chrome
- Skype

حالات الاستخدام ?

- تطبيقات الهاتف المحمولة
- التطبيقات المضمنة والأجهزة
- التطبيقات الصغيرة والمتوسطة
- بيئات الاختبار والتطوير
- تخزين التكوينات والإعدادات
- التطبيقات التي لا تحتاج تزامن شبكي



SQLite هي قاعدة البيانات الافتراضية في Django، مثالية للتطوير والاختبار 

SQL Server | نظرة عامة

قواعد بيانات علائقية من مايكروسوفت

SQL Server هي قاعدة بيانات علائقية (SQL) مطورة من قبل شركة مايكروسوف特، تستخدم في تطبيقات المؤسسات والشركات الكبيرة.

العيوب

- تكاليف ترخيص عالية، خاصة للاستخدام المؤسسي
- دعم محدود للمنصات المتعددة (رغم التحسن مع SQL Server على لينكس)
- يتطلب موارد خادم كبيرة
- أقل مرؤنة من بعض المنافسين لبعض أعباء العمل المتخصصة

المميزات

- تكامل قوي مع منتجات وخدمات مايكروسوف特 الأخرى
- ميزات أمان متقدمة مثل التشفير الدائم وأمان مستوى الصفر
- أدوات ذكاء أعمال قوية مع SQL Server Analysis Services
- أداء وموثوقية عالية لتطبيقات المؤسسات
- توثيق ودعم ممتاز من مايكروسوفت

الشركات المستخدمة

- المؤسسات المالية والبنوك
- الوكالات والهيئات الحكومية
- شركات Fortune 500
- المؤسسات التعليمية الكبرى
- شركات التأمين والرعاية الصحية

حالات الاستخدام

- تطبيقات المؤسسات الكبيرة والشركات
- أنظمة ذكاء الأعمال والتحليلات
- الأنظمة المالية وتطبيقات البنوك
- أنظمة البيانات الحكومية والمؤسسية
- البيانات التي تعتمد على تقنيات مايكروسوفت



يدعم Django قواعد بيانات SQL Server عبر مكتبة mssql-django

Oracle Database | نظرة عامة

نوع القاعدة

قاعدة بيانات علائقية (SQL) تعد الأكثر استقراراً واستخداماً في المؤسسات الكبرى والشركات العالمية.

المميزات

- أداء عالي جداً مع الاستعلامات المعقدة والبيانات الضخمة
- ميزات متقدمة للغاية في الأمان والترخيص والتحكم في الصلاحيات
- دعم متخصص للمعاملات ومعالجة البيانات المتزامنة (Transactions)
- قابلية توسيع عالية جداً مع دعم توزيع البيانات والنسخ الاحتياطي Oracle
- أدوات إدارة متكاملة ودعم فني متخصص من شركة Oracle

العيوب

- تكلفة مرتفعة جداً مقارنة بالحلول الأخرى (تراخيص وصيانة)
- تعقيد كبير في الإعداد والتكوين والإدارة يتطلب خبرات متخصصة
- متطلبات أجهزة عالية لضمان الأداء الأمثل
- منحنى تعلم حاد للمطوريين ومسؤولي قواعد البيانات الجدد

حالات الاستخدام والشركات المستخدمة

الشركات المستخدمة

- 98% من شركات Fortune 500
- شركات مالية: JPMorgan, Bank of America
- شركات تقنية: Amazon, Cisco, Salesforce
- قطاع حكومي: وزارات، جيوش، أجهزة أمنية

حالات الاستخدام المثلية

- أنظمة المؤسسات الكبيرة والحرجة (Enterprise Systems)
- تطبيقات تخطيط موارد المؤسسات (ERP)
- البنوك والمؤسسات المالية والبورصات
- تحليل البيانات الضخمة والذكاء الاصطناعي



MongoDB | نظرة عامة

نوع القاعدة: غير علائقية (NoSQL)

MongoDB هي قاعدة بيانات وثائقية غير علائقية (NoSQL) تخزن البيانات في وثائق بتنسيق شبيه بـ JSON.

- العيوب

- لا يدعم المعاملات المعقدة بنفس كفاءة قواعد SQL
- استهلاك أعلى للمساحة التخزنية مقارنة بالقواعد العلائقية
- أقل أماناً في الإعدادات الافتراضية
- منحنى تعلم للمطورين المعتادين على قواعد SQL
- محدودية في العلاقات المعقدة بين البيانات

+ المميزات

- مرونة كبيرة في هيكل البيانات (Schema-less)
- أداء عالي في عمليات القراءة والكتابة
- سهولة التوسيع الأفقي عبر التجزئة (Sharding)
- دعم لاستعلامات المعقدة والتجميعات
- سهولة التطوير والتكامل مع تقنيات الويب الحديثة

MongoDB شركات تستخدم

- Uber - لمعالجة بيانات الرحلات والموقع الجغرافية
- Meta - لتخزين بيانات المستخدمين والتحليلات
- eBay - للمنتجات وبيانات البحث
- Forbes - لإدارة المحتوى والمقالات
- CISCO - لمنصات إدارة المعلومات

حالات الاستخدام المثلية

- تطبيقات الويب والهاتف المحمول سريعة التطوير
- أنظمة تحليل البيانات الضخمة (Big Data)
- التطبيقات التي تتطلب بيانات متنوعة غير منتظمة
- أنظمة تخزين المحتوى والوثائق
- تطبيقات IoT وأنظمة الزمن الحقيقي



MongoDB هي الخيار الأمثل للتطبيقات التي تتطلب مرونة عالية وسرعة في التطوير 

جدول المقارنة بين قواعد البيانات

MongoDB	Oracle	SQL Server	SQLite	PostgreSQL	MySQL	معايير المقارنة
نوع	غير علائقية NoSQL	علائقية SQL	علائقية SQL	علائقية SQL	علائقية SQL	علائقية SQL
مرونة كبيرة، قابلية توسيع عالية، سريعة، دعم البيانات غير المنظمة	استقرار ممتاز، دعم للإنتاج على نطاق واسع، أمان عالي	تكامل مع منتجات مايكروسوفت، أدوات تحليل قوية، أداء عالي	خفيفة الوزن، ملف واحد، لا تحتاج سيرفر، مثالية للتطبيقات الصغيرة	ميزات متقدمة، استقرار، دعم البيانات المعقدة، توافق مع ACID	سهولة استخدام، شائعة، دعم مجتمعي كبير، أداء جيد	أبرز المميزات
لا تناسب البيانات العلائقية، ضعف في ACID، تستهلك مساحة تخزين أكبر	تكلفة مرتفعة جداً، تعقيد الإدارة، متطلبات خبرة عالية	متطلبات موارد عالية، مرتبطة بمايكروسوفت	لا تناسب التطبيقات الكبيرة، محدودية التزامن، محدودية دعم وظائف SQL	منحنى تعلم أكبر، موارد أكثر من MySQL، إعداد أكثر تعقيداً	أقل تطوراً من PostgreSQL، مشاكل أداء مع البيانات الكبيرة جداً	العيوب
سرع جداً في قراءة/كتابة البيانات غير المنظمة	الأفضل للتطبيقات المؤسسية الكبيرة	ممترز للمؤسسات الكبيرة	سرع جداً للقراءة، بطيء للكتابة المتزامنة	ممترز مع الاستعلامات المعقدة	جيد في معظم الحالات	الأداء
ممترزة (أفقياً)	عالية جداً	عالية	منخفضة	عالية جداً	متوسطة	قابلية التوسيع
Uber, Meta, eBay	98% من شركات Fortune 500	شركات Stack Overflow مالية	تطبيقات الموبايل، Firefox	Instagram, Spotify, Reddit	Facebook, Twitter, YouTube	أمثلة شركات
دعم من خلال djongo/mongoengine	دعم رسمي كامل	دعم من خلال مكتبات خارجية	دعم رسمي كامل (افتراضي)	دعم رسمي كامل (موصى به)	دعم رسمي كامل	دعم Django



ملاحظة: يعتمد اختيار قاعدة البيانات المثلية على متطلبات المشروع المحددة والميزانية المتوفرة.



الاتصال بـ MySQL من Django

إعدادات ملف settings.py

إعدادات قاعدة بيانات MySQL في ملف settings.py

```
DATA BASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'mydatabase',  
        'USER': 'myuser',  
        'PASSWORD': 'mypassword',  
        'HOST': 'localhost', # الخادم IP أو عنوان  
        'PORT': '3306', # المنفذ الافتراضي لـ MySQL  
        'OPTIONS': {  
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",  
            # الخيارات الإضافية هنا  
        }  
    }  
}  
  
# استبدال mysqlclient إذا لم يتم استخدام PyMySQL كبدائل  
# أضف هذه الأسطر في ملف __init__.py الخاص بمشروعك
```

ملاحظات هامة:

- تأكد من إنشاء قاعدة البيانات قبل تشغيل الأمر migrate
- قد تحتاج إلى تعديل الإعدادات بناءً على بيئتك التشغيل الخاصة بك
- يُفضل استخدام mysqlclient للأداء الأفضل، لكن PyMySQL هو بديل جيد

المطلوبات الأساسية

تثبيت المكتبات المطلوبة:

```
pip install mysqlclient
```

أو استخدام البديل:

```
pip install PyMySQL
```

خطوات الإعداد

- تثبيت وإعداد خادم MySQL
- إنشاء قاعدة بيانات وحساب مستخدم
- منح الصلاحيات للمستخدم
- تعديل ملف settings.py كما هو موضح
- تشغيل أمر migrate لإنشاء الجداول

```
python manage.py migrate
```



الاتصال بـ PostgreSQL من Django

إعدادات ملف settings.py

إعدادات قاعدة بيانات PostgreSQL في ملف settings.py

```
DATA BASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'mydatabase',  
        'USER': 'postgres_user',  
        'PASSWORD': 'postgres_password',  
        'HOST': 'localhost', # الخادم IP أو عنوان #  
        'PORT': '5432' # المنفذ الافتراضي لـ PostgreSQL  
    },  
    'OPTIONS': {  
        # إعدادات متقدمة مثل:  
        'sslmode': 'require', # لاتصال الآمن #  
    }  
}  
  
# إعدادات إضافية مفيدة:  
TIME_ZONE = 'UTC' # مهم للتعامل مع التواريخ والأوقات #
```

المطلوبات الأساسية

تثبيت المكتبات المطلوبة:

```
pip install psycopg2
```

أو استخدام البديل البحث:

```
pip install psycopg2-binary
```

خطوات الإعداد

1. تثبيت PostgreSQL على النظام
2. إنشاء قاعدة بيانات جديدة
3. إنشاء مستخدم ومنح الصلاحيات
4. تعديل ملف settings.py كما هو موضح
5. تشغيل أمر migrate لإنشاء الجداول

```
python manage.py migrate
```

ملاحظات هامة:

- يفضل استخدام psycopg2 للأداء الأفضل مع PostgreSQL
- يدعم PostgreSQL أنواع المتقدمة مثل UUID و JSON و HStore
- عند استخدام أنواع بيانات خاصة تأكد من تسجيلها في Django
- يمكن استخدام PostgreSQL مع Geodjango للتطبيقات المكانية



الاتصال بـ SQLite من Django |

إعدادات ملف settings.py </>

المطلبات الأساسية

الإعدادات الافتراضية في ملف settings.py في SQLite هي إعدادات قاعدة بيانات # Django)

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}
```

: القديمة (قبل 3.1)، قد تكون الإعدادات كالتالي Django في إصدارات #

```
# 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
```

لا يحتاج إلى المزيد من الإعدادات مثل اسم المستخدم أو كلمة المرور # SQLite لأن نظام قاعدة بيانات مضمونة ولا يحتاج إلى خادم منفصل #

لا حاجة لتنصيب مكتبات إضافية:

SQLite مدمج مع Django و Python بشكل افتراضي

لا تحتاج لتنصيب أي مكتبات خارجية ✓

خطوات الإعداد

1. لا حاجة لتنصيب خادم قاعدة بيانات منفصل

2. التأكد من وجود الإعدادات الافتراضية في settings.py

3. التأكد من وجود حقوق كتابة في مجلد المشروع

4. تشغيل أمر migrate لإنشاء ملف قاعدة البيانات

```
python manage.py migrate
```

ملاحظات هامة:

- SQLite هو الخيار الافتراضي في Django ومناسب للتطوير
- غير مناسب للتطبيقات كبيرة الحجم أو موقع الويب ذات الحركة العالية
- لا يدعم SQLite الوصول المتزامن من عدة عمليات (مشكلة مع خوادم متعددة)
- لتحويل المشروع إلى قاعدة بيانات أخرى في المستقبل، يمكن استخدام dumpdata/loaddata

إعداد متقدم:

إعدادات متقدمة يمكن إضافتها #

```
'OPTIONS': {  
    'timeout': 20,  
}
```

مميزات SQLite مع Django

- سهولة الإعداد والتكون
- مثالي لبيئة التطوير والاختبار
- لا يتطلب تشغيل خادم منفصل
- ملف واحد يحتوي كامل قاعدة البيانات
- مناسب للتطبيقات الصغيرة والمتوسطة

الاتصال بـ Oracle من Django |

إعدادات ملف settings.py </>

إعدادات قاعدة بيانات Oracle في ملف settings.py

```
DATA BASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.oracle',  
        'NAME': 'xe', # اسم خدمة TNS أو SID  
        'USER': 'username', # اسم مستخدم Oracle  
        'PASSWORD': 'password',  
        'HOST': 'localhost', # لخادم IP أو عنوان  
        'PORT': '1521', # المنفذ الافتراضي لـ Oracle  
        'OPTIONS': {  
            'threaded': True, # تعيين الدعم للـ threads  
            'purity': 'default',  
        },  
        'TEST': {  
            'USER': 'test_user',  
            'TBLSPACE': 'test_tblspace',  
            'TBLSPACE_TMP': 'test_tblspace_tmp'  
        }  
    }  
}
```

ملاحظات هامة:

- يجب تثبيت Oracle Client Library على جهازك أو الخادم
- قد تحتاج إلى إعداد متغير البيئة ORACLE_HOME
- Oracle يفرض قيود على طول أسماء الجداول والأعمدة (30 حرفاً)
- نوصي باستخدام python-oracledb مع Django 5.0 وما بعده

المطلوبات الأساسية

تثبيت المكتبات المطلوبة:

```
pip install python-oracledb
```

الإصدار 1.3.2 أو أحدث مطلوب لـ Django 5

```
pip install cx_Oracle # للإصدارات القديمة من Django
```

خطوات الإعداد

1. تثبيت Oracle Client على الخادم

2. الحصول على TNS Service Name

3. إعداد المستخدم وكلمة المرور

4. تعديل ملف settings.py كما هو موضح

5. تشغيل أمر migrate لإنشاء الجداول

```
python manage.py migrate
```



يتوفر Django رسمياً لـ Oracle ويستخدم بشكل كبير في المؤسسات الكبيرة

Made with Genspark

الاتصال بـ SQL Server | Django

إعدادات ملف settings.py

إعدادات قاعدة بيانات SQL Server في ملف settings.py

```
 DATABASES = {  
    'default': {  
        'ENGINE': 'mssql',  
        'NAME': 'mydatabase',  
        'USER': 'myuser',  
        'PASSWORD': 'mypassword',  
        'HOST': 'localhost\\SQLEXPRESS', # IP اسم السيرفر أو عنوان#  
        'PORT': '', # اترك فارغاً للمنفذ الافتراضي 1433  
  
        'OPTIONS': {  
            'driver': 'ODBC Driver 17 for SQL Server', # اسم السائق  
            'Trusted_Connection': 'no', # للمصادقة بالويندوز استخدم yes  
            'connection_timeout': 30,  
        },  
    }  
}
```

ملاحظات هامة:

- تأكد من تثبيت سائق ODBC (ODBC Driver for SQL Server) على نظامك
- تدعم مكتبة mssql-django مصادقة Windows و Azure Active Directory و SQL Server
- ضبط الإعدادات قد يختلف حسب نسخة SQL Server المستخدمة
- للاتصال بـ Azure SQL Database، استخدم خيارات إضافية في الإعدادات

المطلوبات الأساسية

تثبيت المكتبات المطلوبة:

```
pip install mssql-django
```

أو تثبيته مباشرة من GitHub

```
pip install  
git+https://github.com/microsoft/mssql-django
```

خطوات الإعداد

- تثبيت خادم SQL Server (أو استخدام خادم موجود)
- إنشاء قاعدة بيانات وحساب مستخدم
- منح الصلاحيات للمستخدم
- تعديل ملف settings.py كما هو موضح
- تشغيل أمر migrate لإنشاء الجداول

```
python manage.py migrate
```



الدعم الرسمي لـ SQL Server في Django متاح عبر مكتبة mssql-django من Microsoft

Made with Genspark

الاتصال بـ MongoDB من Django

إعدادات ملف settings.py

```
# الطريقة الأولى: استخدام djongo
# لاتصال بخادم بعيد مع مصادقة CLIENT
# لـ MongoDB
# djongo
# mongoengine
# في settings.py
```

ملاحظات هامة:

- يسمح باستخدام ORM القياسي في Django مع MongoDB
- يوفر ORM مخصص لـ MongoDB مع مزايا إضافية لنمذجة البيانات
- لا يدعم العلاقات كما في قواعد البيانات العلائقية، لذا قد تحتاج إلى تعديل نماذج البيانات
- لا يعتبر Django متوافق رسمياً مع MongoDB، لذا قد تواجه بعض التحديات

المطلوبات الأساسية

تثبيت أحد المكتبات المطلوبة:

```
pip install djongo
```

أو استخدام بديل:

```
pip install mongoengine django-mongoengine
```

خطوات الإعداد

1. تثبيت وتشغيل MongoDB Server

2. إنشاء قاعدة بيانات في MongoDB

3. تثبيت المكتبة المناسبة (djongo أو mongoengine)

4. تعديل ملف settings.py وفقاً للمكتبة المختارة

5. تعديل نماذج Django للتواافق مع MongoDB

6. تنفيذ الهجرات (في حالة djongo)

```
python manage.py makemigrations
python manage.py migrate
```



| التوصيات والخلاصة

النقاط الرئيسية

: الخيار الأمثل للمشاريع المتوسطة والكبيرة والتي تتطلب استقراراً وميزات متقدمة.

: مناسب للتطبيقات الويب الشائعة مع سهولة الاستخدام والدعم الواسع.

: مثالي للتطبيقات الصغيرة والمحمولة والتجربة السريعة.

: الأفضل للبيانات غير المنظمة والتطبيقات التي تتطلب مرونة عالية في البنية.

: للمؤسسات الكبيرة التي تحتاج أقصى درجات الأمان والاستقرار.

❖ أفضل الممارسات مع Django

- استخدم **PostgreSQL** كخيار أول مع Django (موصى به رسميًا)
- استخدم **SQLite** للتطوير فقط، وليس للإنتاج
- تأكد من تثبيت مكتبات الاتصال المناسبة لكل قاعدة بيانات
- استخدم متغيرات البيئة لتخزين معلومات الاتصال بقاعدة البيانات
- قم بتنفيذ فحوصات الاتصال قبل نشر التطبيق

💡 توصيات اختيار قاعدة البيانات

- اختر **PostgreSQL** للمشاريع التي ستتوسع مستقبلاً
- اختر **MySQL** للتطبيقات البسيطة والسريعة
- استخدم **SQLite** للتطوير المحلي والتطبيقات الصغيرة
- اختر **MongoDB** إذا كانت بياناتك متغيرة الهيكل
- استثمر في **Oracle** إذا كان أمان البيانات هو الأولوية القصوى

❖ الخلاصة

لا توجد قاعدة بيانات "مثالية" لكل الحالات. اختيار قاعدة البيانات المناسبة يعتمد على متطلبات المشروع، حجم البيانات، نوع التطبيق، الميزانية المتاحة، والخبرة التقنية لفريق التطوير. من الأفضل الاستفادة من نقاط القوة في كل نظام و اختيار ما يناسب احتياجاتك.

