

1. Update Database Settings

In your Django settings.py, you can add configuration for multiple databases. Here's an example of how to set up two databases (default and tenant_db):

```
# settings.py
```

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql', # Use 'postgresql', 'mysql', etc.  
        'NAME': 'default_db',  
        'USER': 'your_default_db_user',  
        'PASSWORD': 'default_db_password',  
        'HOST': 'localhost', # Or your database server  
        'PORT': '5432', # Default PostgreSQL port  
    },  
    'tenant_db': {  
        'ENGINE': 'django.db.backends.postgresql', # Use 'postgresql', 'mysql', etc.  
        'NAME': 'tenant_db',  
        'USER': 'your_tenant_db_user',  
        'PASSWORD': 'tenant_db_password',  
        'HOST': 'localhost',  
        'PORT': '5432',  
    }  
}
```

2. Configure Database Routing

Django allows you to define database routers to determine which database should be used for specific models or operations. Create a new file, db_router.py, and add routing logic for tenant and default databases:

```
# db_router.py
```

```
class TenantRouter:  
    """  
    A router to control all database operations on models related to tenants.
```

```
"""
```

```
def db_for_read(self, model, **hints):
```

```
    """Direct reads for tenant models to the tenant_db."""
```

```
    if model._meta.app_label == 'tenant_app': # Replace with your tenant app name
```

```
        return 'tenant_db'
```

```
    return 'default'
```

```
def db_for_write(self, model, **hints):
```

```
    """Direct writes for tenant models to the tenant_db."""
```

```
    if model._meta.app_label == 'tenant_app':
```

```
        return 'tenant_db'
```

```
    return 'default'
```

```
def allow_relation(self, obj1, obj2, **hints):
```

```
    """Allow relations only within the same database."""
```

```
    if obj1._state.db == obj2._state.db:
```

```
        return True
```

```
    return False
```

```
def allow_migrate(self, db, app_label, model_name=None, **hints):
```

```
    """Make sure the tenant app only appears in the tenant_db database."""
```

```
    if app_label == 'tenant_app':
```

```
        return db == 'tenant_db'
```

```
    return db == 'default'
```

In settings.py, add the database router:

```
# settings.py
```

```
DATABASE_ROUTERS = ['path.to.db_router.TenantRouter']
```

Replace 'tenant_app' with the name of the Django app containing your tenant-specific models.

3. Use Tenant-Specific Tables (Optional)

If using PostgreSQL, you can leverage schemas to separate tenant data logically. The `django-tenants` package or `django-multitenant` can help streamline this. These packages allow you to set up separate schemas for each tenant with shared tables for common data.

- **django-tenants:** Recommended for PostgreSQL, provides tenant schemas.
- **django-multitenant:** Works with various databases, useful for multi-tenant setups without schemas.

4. Modify Management Commands (e.g., Migrations)

If you need to run migrations on the `tenant_db`, you can specify the database with:

```
python manage.py migrate --database=tenant_db
```

Or to target a specific app:

```
python manage.py migrate tenant_app --database=tenant_db
```

5. Accessing Specific Databases in Code

To explicitly use a database when querying models, specify the database like this:

```
# Query from the tenant database
```

```
ModelName.objects.using('tenant_db').all()
```