DATA MODEL DOCTOR

IDEO-LAB

jANUARY 29TH 2026

VERSION 1.55

AGENDA DATAMODEL DOCTOR

0) Objectif & sortie

1) Modes d’exécution

2) Pipeline d’analyse (ordre logique)

3) Checklists (catégories de diagnostics)

4) Système de règles (extensible)

5) Reporting (bilan “très poussé”)

6) Intégration projet (Ideo-Lab)

7) MVP vs V2

0) Objectif & sortie

- **But** : analyser un modèle Django (ou une app) *avant même les migrations* et produire un **bilan complet** : erreurs probables, dettes d'architecture, risques DB, incohérences FK, perf, compat.
- **Sorties** :
  1. **Rapport console** (lisible, résumé + détails)
  2. **Rapport JSON** (pour cron/CI, dashboards)
  3. (option) **Rapport HTML** façon Ideo-Lab (cards + modals)

1) Modes d’exécution

- `--model app_label.ModelName` : audit ciblé (ton besoin principal)
- `--app app_label` : audit tous les modèles d’une app
- `--all-local` : audit global (toutes apps locales)
- `--diff` (optionnel) : compare deux snapshots (avant/après patch)

2) Pipeline d'analyse (ordre logique)

1. **Introspection Django**
   - chargement `apps.get_models()`, champs, `Meta`, managers
   - dépendances : parents (inheritance), M2M through, proxy, abstract
2. **Graph des relations**
   - graphe FK/OneToOne/M2M (direction, cardinalité, `related_name`)
   - détection cycles et "goulots" (tables centrales)
3. **Compat DB / backend**
   - règles spécifiques `mysql/mariadb` vs `postgres` (types, index length, text/blob index, collations)
   - compat migration (opérations "dangereuses")
4. **Audit de schéma réel (optionnel)**
   - si `--with-db` : introspection tables/colonnes/index/fk existants
   - diff modèles ↔ DB (mais sans faire de migrate)
5. **Scoring & recommandations**
   - score global + scores par catégorie (FK, indexes, perf, sécurité, data integrity)

3) Checklists (catégories de diagnostics)

A) Erreurs "bloquantes" (probables crash)

- collision de noms de table/colonne (db_column, db_table)
- champs invalides selon backend (ex: `JSONField` selon version/DB)
- `unique_together` / `constraints` contradictoires
- `related_name` en collision (reverse accessor)
- `on_delete=DO_NOTHING` sans contrainte réelle et sans stratégie
- M2M implicite vs `through` mal défini
- `null=True` + `unique=True` (pièges MySQL/NULL)
- champs `CharField(max_length=…)` incompatibles index (MySQL index l

B) Problèmes Foreign Keys / intégrité

- FK vers modèle `managed=False` sans garde-fous
- FK vers table qui n'existe pas (si `--with-db`)
- `db_constraint=False` : avertissement + justification attendue
- cascades dangereuses (CASCADE sur tables volumineuses)
- `SET_NULL` alors que champ `null=False`
- cycles de cascade (risque suppression en chaîne inattendue)
- OneToOne utilisé là où FK + unique ferait mieux (ou inverse)

C) Architecture & maintainability

- modèle "god object" (trop de champs, trop de relations)
- champs redondants / dérivables (risque de divergence)
- `TextField` là où `CharField` suffirait (index, perf)
- `choices` non stables / générés dynamiquement (risque churn migrations)
- champs calculés mis en DB sans mécanisme d'update (signals/cron)
- noms / conventions incohérents (verbose_name, plural, related_name)

D) Performance (DB & ORM)

- index manquants sur :
  - FK
  - champs filtrés fréquemment (`status`, `created_at`, `project_id`, etc.)
  - champs utilisés en `order_by`
- indexes inutiles / redondants
- composite indexes recommandés (patterns `WHERE a=? AND b=?`)
- risques `N+1` "structurels" (relations trop profondes)
- `select_for_update`/verrous potentiels (si patterns détectables via Meta / usage connu)
- `auto_now_add/auto_now` + filtrage temps (index timestamp)

E) Migration-safety (même si tu ne migres pas)

- champs avec `default=` dynamique (datetime.now() au import) → churn / migrations fantômes
- changement de type "dangereux" (ex int→char, char→text)
- `null` → `not null` sans default/data migration
- suppressions de champs sans stratégie de backfill/archive
- renommages sans `RenameField` (risque data loss)

F) Sécurité & data correctness

- PII détectée (email, phone, ip) → recommandations (hash, encrypt, retention)
- champs "secrets" en clair (token, secret_key) → warning
- audit des champs "logs" volumineux (risque inflation DB)
- audit `IPAddressField` / stockage IPv6 / compat

4) Système de règles (extensible)

- règles codées type `Rule` :
  - `id`, `severity` (INFO/WARN/ERROR/CRIT), `category`, `evidence`, `recommendation`
- moteur :
  - passe 1 : règles locales sur champ
  - passe 2 : règles sur modèle
  - passe 3 : règles sur graphe multi-modèles
- allowlist / suppressions :
  - `# dmd: ignore RULE_ID reason="..."` (commentaire dans model)
  - config `.yml` / `settings.DATA_MODEL_DOCTOR = {...}`

5) Reporting (bilan “très poussé”)

- résumé en tête :
  - nb ERROR/WARN/INFO
  - top 5 risques
  - score par catégorie
- détails :
  - par modèle → par champ → preuves + recommandations
- export JSON :
  - parfait pour cron : “diff” entre runs + alerting

6) Intégration projet (Ideo-Lab)

- commande Django : `manage.py data_model_doctor`
- cron :
  - daily `--all-local --check --json --store`
- option stockage DB (comme migration doctor) :
  - `DoctorRun`, `ModelSnapshot`, `RuleHit` (option)
- page HTML dashboard : cards + modals (pattern Ideo-Lab)

---

7) MVP vs V2

MVP (rapide, utile tout de suite)

- introspection Django + graphe relations
- règles A/B/C + churn defaults
- report console + JSON

V2

- `--with-db` (introspection schema)
- scoring avancé + suggestions indexes composites
- page dashboard + historique des runs
- auto “fix proposals” (patch suggestions, pas d’écriture automatique)

OPTIONS EXACTES

OPTIONS EXACTES

- 🎯 Ciblage
- 🔍 Modes d’analyse
- ❓ Analyse avancée
- 🚫 Contrôle des règles

🎯 Ciblage

Option	Description
`--model app.Model`	Audit **un seul modèle**
`--app app_label`	Audit tous les modèles d'une app
`--apps a1,a2,a3`	Audit plusieurs apps
`--all-local`	Audit toutes les apps locales (ignore django/contrib & site-packages)

🔍 Modes d'analyse

Option	Description
`--check`	Lance l'audit complet (par défaut si rien d'autre)
`--with-db`	Compare modèles ↔ schéma DB réel
`--json`	Sortie JSON (stdout)
`--store`	Sauvegarde rapport en DB (si app doctor installée)
`--fail-on-error`	Exit code 1 si erreurs CRIT/ERROR (CI/CD)

🧪 Analyse avancée

Option	Description
`--graph`	Affiche graphe des relations FK
`--perf`	Active règles performance (indexes manquants etc.)
`--integrity`	Active règles FK / on_delete / nullability
`--architecture`	Active règles design modèle
`--migration-safety`	Active règles risques migrations futures

➡ Par défaut **TOUT est activé**

🚫 Contrôle des règles

Option	Description
`--ignore RULE_ID`	Ignore une règle spécifique
`--only RULE_ID`	N'exécute qu'une règle
`--min-severity WARN`	Affiche seulement WARN/ERROR/CRIT

🏗️ 2) ARCHITECTURE CODE

powershell

Copier le code

```
data_model_doctor/
 ├── management/commands/data_model_doctor.py   ← CLI
 ├── engine.py                                  ← moteur d'audit
 ├── graph.py                                   ← graphe relations
 ├── rules/
 │     ├── base.py                              ← classe Rule
 │     ├── fields.py                            ← règles champs
 │     ├── fks.py                               ← règles relations
 │     ├── architecture.py                      ← règles design
 │     ├── performance.py                       ← règles index/perf
 │     ├── migrations.py                        ← risques migrations
 ├── reporters/
 │     ├── console.py                           ← sortie lisible
 │     ├── json_report.py                       ← export JSON
 └── utils.py
```

ANALYSE DETAILLEE

🧩 Classe Rule (base)

python

Copier le code

```python
class Rule:
    id = "GEN001"
    severity = "WARN"  # INFO/WARN/ERROR/CRIT
    category = "architecture"

    def check_model(self, model, context):
        return []  # list of findings
```

Chaque finding :

python

Copier le code

```python
{
    "rule": "FK001",
    "severity": "ERROR",
    "model": "weglot.Translation",
    "field": "project",
    "message": "ForeignKey uses DO_NOTHING",
    "hint": "Use PROTECT or CASCADE with justification"
}
```

🧠 Engine

python

Copier le code

```python
class ModelDoctorEngine:
    def run(models, options):
        graph = build_relation_graph(models)
        findings = []

        for rule in enabled_rules:
            findings += rule.run(models, graph, options)

        return findings
```

🕸️ Graph Builder

Détecte :

- cycles FK
- tables “centrales” (trop de dépendances)
- cascades dangereuses

📢 Reporters

Console (lisible humain)

```
[ERROR][FK001] weglot.Translation.project
    ForeignKey uses DO_NOTHING
    → Use PROTECT or CASCADE

[WARN][PERF003] weglot.Translation.created_at
    Field used for ordering but not indexed
```

MVP REGLES INCLUSES

🔴 FK / INTÉGRITÉ

ID	Description
FK001	`on_delete=DO_NOTHING`
FK002	`SET_NULL` mais `null=False`
FK003	FK sans index (perf)
FK004	related_name collision
FK005	cascade chain dangereuse

🟠 CHAMPS

ID	Description
FLD001	`CharField` sans max_length correct
FLD002	`null=True` + `unique=True`
FLD003	default dynamique (datetime.now())
FLD004	TextField indexé (MySQL danger)

🟡 ARCHITECTURE

ID	Description
ARC001	Modèle > 25 champs
ARC002	Trop de FK (>10)
ARC003	Champs redondants suspects (name + title + label etc.)

🟢 PERFORMANCE

ID	Description
PERF001	FK non indexée
PERF002	Champ `created_at/updated_at` non indexé
PERF003	order_by sur champ non indexé

MIGRATION SAFETY

ID	Description
MIG001	default dynamique = churn migrations
MIG002	null=False sans default
MIG003	type risqué pour MySQL index

⚙ 4) EXEMPLES UTILISATION

Audit complet projet :

bash Copier le code

```
python manage.py data_model_doctor --all-local --check
```

Audit modèle unique :

bash Copier le code

```
python manage.py data_model_doctor --model weglot.TranslationProject
```

CI strict :

bash Copier le code

```
python manage.py data_model_doctor --all-local --fail-on-error --json
```

Audit + DB compare :

bash Copier le code

```
python manage.py data_model_doctor --app weglot --with-db
```

🚀 PROCHAINE ÉTAPE

Je te génère maintenant :

1. Le squelette Django complet
2. Le moteur MVP fonctionnel
3. Les règles de base implémentées
4. Le report console clair

📊 SCORE INTERPRÉTATION

Score	État
90-100	Modèle sain
70-89	Quelques dettes techniques
40-69	Problèmes sérieux
<40	Architecture dangereuse

NOUVELLES FONCTIONNALITES

NOUVELLES CAPACITÉS AJOUTÉES

- Graphe complet des relations FK
- Détection de cycles (boucles FK dangereuses)
- Analyse du schéma DB réel
- Score de santé du modèle
- Rapport exploitable pour futur dashbc

✅ CE MVP FAIT DÉJÀ

✓ FK dangereuses

✓ erreurs null/unique

✓ défauts migration churn

✓ modèle trop gros

✓ perf basique datetime

✓ textfield index dangereux