21-05 Cours: Modélisation NoSQL et Sécurité des Systèmes d’Information — schémas, index, chiffrement, audit, monitoring

Modélisation NoSQL et Sécurité des Systèmes d’Information: schémas, accès, chiffrement, audit et monitoring

Introduction générale: une approche intégrée, du modèle de données à la sécurité opérationnelle

La réussite d’un projet data moderne repose sur deux piliers complémentaires: une modélisation NoSQL adaptée

aux usages réels et une sécurité “en couches” couvrant l’accès, le chiffrement, l’audit et le monitoring. Contrairement à la normalisation relationnelle “standardisée”, la modélisation NoSQL (documents, colonnes, clé/valeur) se décide au regard des patrons d’accès, de la volumétrie et des contraintes du moteur choisi (MongoDB, Elasticsearch, DynamoDB, etc.). En parallèle, la sécurité n’est jamais une mesure unique: elle s’organise comme un oignon, par couches successives (réseau, transport, authentification, autorisation, données, opérations). Ce chapitre adopte un ton didactique et pragmatique: il développe les choix de schéma (embedding vs références), les limites physiques (taille des documents, I/O), la gestion des binaires, l’indexation et la performance; puis relie ces décisions au chiffrement (en transit et au repos), aux contrôles d’accès (moindre privilège), à l’audit et au monitoring, jusqu’aux sauvegardes testées et à la posture professionnelle.

Partie I — Modélisation NoSQL: principes, choix et bonnes pratiques

1. Pourquoi “ça dépend” en NoSQL

En NoSQL, le “bon” modèle est celui qui justifie ses compromis au regard:

- Des données: forme, taille, hétérogénéité, volatilité.
- Des patrons d’accès: lecture, écriture, filtres, tri, pagination.
- Des contraintes opérationnelles: latence, réplication, limites de taille, ressources.
- Du moteur: capacités d’indexation, mapping, agrégation, limites de document. Idée clé: on choisit le modèle physique pour simplifier les requêtes critiques, respecter les limites techniques et rester malléable.

2. Schéma-less, mais pas sans discipline

Schema-less n’est pas l’absence totale de structure: c’est la capacité à accepter des documents hétérogènes. Exemple: une spec “couleur” en string, une “taille” en entier/float, ou l’absence de certaines clés d’un produit à l’autre. Cette flexibilité:

- Facilite l’évolution du domaine (ajout de nouvelles specs sans migration lourde).
- Implique une gouvernance applicative (conventions de nommage et types).
- Doit composer avec le mapping des moteurs (Elasticsearch: inflation du mapping si trop de variété). Bonne pratique: définir des conventions internes et des index ciblés sur les attributs réellement exploités.

3. Embedding vs Références: la décision structurante

Deux approches physiques:

- Embedding (imbriquer les sous-documents dans le parent):
  - Avantages: lecture “one-shot”, cohérence locale, simplicité applicative.
  - Limites: taille du document, mises à jour ciblées plus coûteuses, filtrage/pagination compliqués sur de gros tableaux.
- Références (collections séparées + clés étrangères applicatives):
  - Avantages: scalabilité, indexation spécifique, accès ciblé/paginé, contrôle du poids.
  - Limites: jointures applicatives, complexité opérationnelle, cohérence à gérer. Règle d’or:
- Sous-éléments nombreux, évolutifs, accédés indépendamment → collection séparée.
- Petites listes, fortement liées au parent, lues ensemble → embedding.

4. Cardinalité, attributs multi-valués et index

- Relations 1–N: préférer une collection séparée où chaque enfant porte la clé du parent (productId), plutôt qu'une liste d'IDs dans le parent (complexifie requêtes et maintenance).
- Attributs multi-valués: les tableaux peuvent être indexés mais restent coûteux si volumineux; attention aux limites du moteur (multi-keys, nested).
- Choix des types: pour les prix, privilégier des entiers en centimes ou des décimaux avec précision gérée; pour les catégories, enum si stable, string si flexible (standardiser côté appli).

5. La taille contre le nombre: mesurez en octets et en I/O

"Beaucoup" ne signifie pas "mille éléments"—ce qui importe, c'est la taille totale et l'empreinte I/O:

- Plus un document est gros, plus il est long à transférer et traiter (mémoire, cache, réplication).
- Les limites de taille (ex. 16 Mo) protègent la stabilité des moteurs; éviter d'approcher ces seuils.
- Favoriser les accès via index sélectifs plutôt que le chargement massif. Point de vigilance: un tableau de 100 objets avec corps texte volumineux peut suffire à pénaliser les lectures; séparer et paginer.

6. Données binaires: éviter l'inline

Stocker des binaires en base JSON via base64 gonfle la taille et dégrade les performances.

- Alternatives: stockage objet (S3, GCS, Azure Blob) référencé par URL; GridFS (MongoDB) si besoin côté moteur.
- Pratique: conserver les métadonnées en base et pointer vers le binaire.

7. Indexation et performance: filtrer ≠ charger

- Un index peut cibler des champs imbriqués, mais si la requête n'est pas sélective, le moteur chargera quand même des blocs volumineux.
- Les index multi-clés sur des tableaux aident, au prix d'un coût d'écriture augmenté.
- Design des index dès le départ: sur (productId, nom, valeur) pour specs; sur (productId, date) ou (productId, note) pour avis.

8. Étude de cas: Produits, Specs, Avis

- Produits: id, nom, prix, catégorie, métadonnées; éventuellement champs dérivés (dénomralisation).
- Specs:
  - Embedding si peu nombreuses et descriptives (lecture atomique).
  - Collection séparée si filtrage intensif par attributs (couleur, taille, matière) avec index dédiés.
- Avis:
  - Avoid embedding massif: risque de dépasser la taille et surcoût I/O.
  - Collection “Avis” séparée: pagination, index sur date/note, chargements partiels.
  - Dénormalisation partielle dans le produit: moyenne, compteur, derniers N avis “légers” (snippet), pour accélérer l’affichage. Stratégies de bascule: monitorer la taille des documents; au-delà d’un seuil (N avis ou taille), passer à un modèle référencé.

9. Impact du moteur choisi: MongoDB vs Elasticsearch

- MongoDB: schema-less tolérant, indexes sur champs imbriqués, pipeline d’agrégation (match, unwind, group, project); attention à la limite de 16 Mo par document.
- Elasticsearch: orienté recherche, mapping nécessaire; sous-documents “nested” avec requêtes dédiées; trop de variété de clés dégrade le mapping. Conclusion: le moteur dicte aussi le schéma—adapter, tester, mesurer.

10. Exemples de designs

- Design A (embedding modéré):
  - Produit: specs courtes embedded; avisRésumé (moyenne, count, derniersN).
  - Avis complets séparés (pagination).
- Design B (références fortes):
  - Produit minimal; specs et avis en collections dédiées; index composés ciblés.
- Design C (hybride):
  - Quelques specs “pivots” embedded (taille, couleur); reste séparé.
  - Avis séparés + résumé dans le produit.

11. Analogies et cas applicatifs

- Blog et commentaires: embedding pour petite audience; séparation quand le volume explose.

- Catalogue multimédia: métadonnées très variées (codec, durée, résolution) = force du schema-less.
- Commerce à forte charge: séparation fine des avis/specs pour indexation, sharding et mise à l’échelle.

12. Erreurs fréquentes à éviter

- Listes d’IDs multi-valuées au lieu d’une relation 1–N classique.
- Embedding indiscriminé des avis volumineux.
- Oublier les index sur champs de filtrage.
- Croire que schema-less dispense de discipline.
- Ignorer les coûts d’I/O et la pagination réelle.

Partie II — Sécurité des systèmes d’information et des bases: principes, couches et pratiques

1. Défense en profondeur et moindre privilège

La sécurité est une chaîne; elle cède au maillon le plus faible. Approche par couches:

- Réseau: segmentation, allowlists, pare-feu.
- Transport: chiffrement TLS/mTLS.
- Authentification: comptes individuels, MFA, rotation.
- Autorisation: rôles, droits CRUD limités, RLS si disponible.
- Données: chiffrement au repos, politiques de purge maîtrisées.
- Opérations: journalisation, audit externe, sauvegardes testées, procédures d’incident.
  Principe du moindre privilège: ne donner que les droits nécessaires; limiter l’impact d’une compromission.

2. Contrôles d’accès: rôles, permissions, soft delete

- Gérer via rôles (lecteur, éditeur, service, admin); rattacher les comptes aux rôles, pas de droits ad hoc.
- Politique de soft delete: retirer DELETE; marquer “archivé = true”; prévoir vues et audits; respecter RGPD (droit à l’effacement). Modèles d’identité:
- Compte technique unique (simple, mais audit détaillé côté BD plus difficile).
- Propagation d’identité (SSO/Kerberos/JWT) jusque dans la BD (audit fin, complexité accrue).

3. Authentification et hygiène

- Changer impérativement les comptes par défaut (admin/admin).
- Mots de passe forts, rotation, gestionnaire de secrets; offboarding rigoureux.
- Activer et configurer l’authentification/autorisation côté BD (ne pas laisser en mode “ouvert”). Curiosité historique: de nombreuses compromissions “simples” exploitent des identifiants par défaut non modifiés.

4. Contrôle d’accès réseau

- Restreindre aux hôtes connus; différencier dev/test/prod.
- Éviter les connexions directes depuis postes utilisateurs vers BD de production.
- Segmenter (VLAN), ACL, bastion administrateur. Exemple MySQL: lier les comptes à “utilisateur@hôte” avec hôte restreint (éviter “%”).

5. Journalisation et audit

Objectifs: forensique, conformité, accounting.

- Journaliser connexions, DDL, DML critiques, requêtes lentes.
- Protéger les journaux (intégrité; stockage externe immuable/WORM).
- Corréler dans un SIEM (ELK/Opensearch, Splunk). Curiosité: journaux “tamper-evident” enchaînés par hachages, inspirés des blockchains.

6. Chiffrement: en transit et au repos

- En transit (TLS/SSL): prévenir interception et modification; gérer certificats, rotation; mTLS pour microservices.
- Au repos:
  - Chiffrement de disque (FDE: BitLocker, LUKS, FileVault): protection contre vol physique; transparent pour l'application.
  - TDE au niveau moteur (Oracle, SQL Server, extensions PostgreSQL): fichiers de base et backups chiffrés; nécessite KMS/HSM.
  - Chiffrement applicatif (colonnes/blobs): protection forte même avec accès moteur; perte de requêtabilité et complexité de clés. Pragmatisme PKI interne: commencer par la périphérie (reverse proxy/API gateway, VPN), puis étendre le chiffrement interne avec automation (ACME/step-ca/Vault) ou service mesh (Istio, Linkerd).

7. Sauvegardes, transactions et reprise

- Transactions: BEGIN/COMMIT/ROLLBACK; éviter auto-commit pour opérations massives.
- Sauvegardes: full/incrémentales/différentielles/snapshots; isoler et durcir les dépôts; chiffrer les backups.
- Tester les restaurations régulièrement: une sauvegarde non testée n'est pas une sauvegarde; valider RPO/RTO, intégrité, runbooks. Cas réel: backups "réussis" mais vides (0 octet) faute de droits; jour J, rien à restaurer.

8. Décommissionnement des supports

- Effacement cryptographique si disque chiffré (crypto-erase).
- Déchiquetage industriel (DIN 66399), Secure Erase/PSID pour SSD, percage/déformation des plateaux pour HDD.
- Documenter la procédure, séparer HDD/SSD, prestataires certifiés. Curiosité: l'aimant "courant" n'a pas d'effet sur HDD modernes; nul pour SSD.

Partie III — Sécurité des applications web: injections SQL, XSS, validation et WAF

1. Injections SQL: nature et prévention

Définition: injection lorsque des données utilisateur deviennent du code SQL et modifient la requête.

- Exemples: ' OR 1=1 --, UNION, blind/time-based.
- Impacts: bypass d’authentification, exfiltration, modification, DROP selon droits. Prévention incontournable: requêtes préparées (prepared statements).
- Séparent code et données; les paramètres ne sont pas interprétés comme SQL.
- Bénéfices de performance: plan cache, stabilité, moins de parsing. Bonnes pratiques:
- APIs paramétrées (PDO, PreparedStatement, psycopg2/sqlalchemy).
- Typage explicite des paramètres; encodage cohérent (UTF-8).
- Messages d’erreur non verbeux côté utilisateur; journaux détaillés côté SIEM.
- Moindre privilège: même si injection, les dégâts sont limités.

2. Validation des entrées et échappement des sorties

- Input validation: whitelist, longueur, jeu de caractères, normalisation (NFKC); utile mais ne remplace pas les prepared statements.
- Output escaping: protéger contre le XSS (HTML, attribut, JS, URL); outils et CSP; ne pas confondre avec protection SQL.

3. WAF et détection

- WAF niveau 7 (OWASP CRS): détecter patterns d’attaques, journaliser, bloquer/quarantaine.
- Limites: complément, pas substitut au code sécurisé.
- Réponse aux incidents: playbooks, corrélations SIEM, durcissement des règles.

Partie IV — Monitoring, opérations et posture de conseil

1. Logging et transactions: éviter le piège du rollback des logs

- Anti-pattern: logguer “en base” dans la même transaction que le métier — le rollback efface aussi les logs.
- Solutions:
  - Transaction séparée et commit immédiat pour la ligne de log.
  - Logging hors base (stdout, fichiers, bus → collecte centralisée).
  - Audit natif BD pour événements sensibles (pgaudit, profiler). Pratiques:
- Corrélation: request_id/session_id dans chaque log et write BD.
- Séparer log d’événement vs log d’audit; niveaux (ERROR/WARN/INFO); rétention.

2. Observabilité: métriques, traces et alertes

- Time-series: Prometheus, InfluxDB; dashboards (Grafana).
- Logs/traces: ELK/Opensearch, Loki, OpenTelemetry.
- Métriques critiques: connexions, latence, locks, I/O disque, buffer cache, CPU/RAM, taille des index/tables, WAL/redo.
- Alertes: disque >80%, dérive latences, locks longs, croissance anormale des journaux. Checklist santé par SGBD:
- PostgreSQL: pg_stat_activity, pg_stat_statements, autovacuum, pgaudit.
- MySQL/MariaDB: performance_schema, information_schema.
- Oracle: AWR/ASH, OEM.
- MongoDB: profiler, Compass.

3. Accès et réseau: segmentation, bastion, certificats

- Segmentation stricte; deny-by-default; bastion avec MFA pour admins.
- TLS/mTLS entre application et BD; PKI interne; automatiser renouvellement (ACME/Step CA).
- Argumentaire: coût des certificats négligeable face au coût d’une fuite.

4. Sauvegardes et restaurations: vérité opérationnelle

- Stratégie: full + incrémentales; isoler, immutabilité (WORM).
- Tests de restauration périodiques; validation d’intégrité (hash), contenu, performances de reprise.
- Documentation: runbooks d’incident; responsabilités claires.

5. Scénarios d’attaque et erreurs systémiques

- Brute force sur admin évident; élévation aux hyperviseurs; destruction des backups; rançon.
  - Mesures: MFA, segmentation, immutabilité des backups, détection brute force, moindre privilège.
- Risque interne: mécontentement, négligence, collusion.
  - Mesures: audit immuable, revue d’accès périodique, séparation des tâches, culture de sécurité.

6. Posture de conseil et responsabilité

- Devoir de conseil: exposer risques et options; adapter au contexte; formuler les compromis.
- Limites: refuser un engagement manifestement non sécurisé; décharge écrite si nécessaire.
- Traçabilité: ordres de changement signés pour actions sensibles; journaliser qui/quoi/quand/pourquoi.
- Éthique: environnement professionnel respectueux; évolution de responsabilités (de savoir-faire à pilotage des choix).

Interdisciplinarité et liens utiles

- Génie logiciel: DDD pour déterminer agrégats (embedding) vs bounded contexts (références).
- Architecture systèmes: impact schéma sur réplication, sharding, tolérance aux pannes, coûts cloud.
- Sécurité réseau: Zero Trust, mTLS, bastions, segmentation dynamique.
- Data engineering: pipeline ETL, indexation vers moteurs de recherche (Elasticsearch), caches (Redis), data lakes (Parquet/Avro, Spark/Flink).
- Conformité: RGPD (minimisation, anonymisation, droit à l’effacement), PCI-DSS.
- SRE/Opérations: CI/CD avec scanners SAST/DAST, runbooks, game days. Curiosités historiques:
- Essor du NoSQL avec la scalabilité web; MongoDB a popularisé documents JSON-like et pipelines d’agrégation.
- Elasticsearch/Lucene: index inversés, analyzers; mapping strict pour performance de recherche.

- Limites de taille des documents: corrélées aux pages/blocs internes et aux coûts de réplication; compromis de stabilité.

Exemples pratiques intégrés

Cas 1: E-commerce avec filtrage massif de specs et avis nombreux

- Specs: collection séparée avec index sur (productId, nom, valeur) pour “couleur/taille/matière”.
- Avis: collection dédiée, index (productId, date) pour “les 4 plus récents”; produits dénormalisés avec moyenne et compteur.
- Monitoring: latences par requête, taille des collections, croissance index.

Cas 2: Binaires (PDF de factures)

- Métadonnées en base; fichiers en stockage objet (URL, hash, MIME); GridFS si moteur document imposé.
- Sécurité: TDE/FDE pour backups; chiffrement en transit; accès restreints.

Cas 3: Petites listes fortement liées (adresses client)

- Embedding si 1–2 adresses statiques; collection séparée si historique riche (validation, audit, pagination).

Cas 4: Web App et SQLi/XSS

- Prepared statements partout; validation en entrée; escaping en sortie (HTML/JS/URL); CSP; WAF en couche supplémentaire; moindre privilège sur comptes BD.

Pistes méthodologiques: concevoir, tester, ajuster

1. Modèle conceptuel: entités/relations (produits, specs, avis, clients).
2. Patrons d’accès et SLA: requêtes fréquentes, volumes, filtres/tri/pagination.
3. Modèle physique et index:
   - Embedding vs référenciation par cardinalité et volumétrie.
   - Dimensionner la taille des documents; planifier index et partitions.
   - Prévoir pipelines (ETL, caches, search), résilience et resynchronisation.
4. Sécurité et opérations:
   - Rôles et moindres privilèges; TLS/mTLS; audit externe; sauvegardes testées.
   - Observabilité: métriques, logs, traces; alertes actionnables.
5. Itérations: mesurer, profiler, corriger (coût des requêtes, taille des payloads, plans d’exécution).

Points clés et mots-clés (mini-listes contextualisées)

- Modélisation:
  - Embedding vs Références
  - Cardinalité
  - Schema-less discipliné
  - Index multi-clés
  - I/O et taille de document
- Sécurité:
  - Défense en profondeur
  - Moindre privilège
  - TLS/mTLS, TDE/FDE
  - Audit immuable, SIEM
  - Soft delete, RLS
- Web:
  - Prepared statements
  - Input validation, Output escaping
  - XSS, WAF, CSP
- Opérations:
  - Logging hors transaction
  - Monitoring (Prometheus/Grafana)

    - Sauvegardes testées, RPO/RTO
    - Runbooks, game days
- Architecture:
    - DDD, bounded contexts
    - ETL, caches, search
    - PKI interne, service mesh

Schéma riassuntivo: concepts principaux et mots-clés

Concepts principaux:

- Modèle NoSQL centré usages: choisir embedding ou références selon cardinalité, filtrage, volumétrie et limites du moteur (MongoDB, Elasticsearch).
- Taille > nombre: mesurer en octets et en I/O; éviter les documents lourds; indexer les champs de filtrage.
- Binaires hors base: stockage objet + métadonnées; GridFS si nécessaire.
- Sécurité en couches: réseau (segmentation), transport (TLS/mTLS), identité (MFA, rotation), autorisation (rôles, moindre privilège), données (TDE/FDE, chiffrement applicatif), opérations (audit, sauvegardes testées).
- Prévention des injections: requêtes préparées, validation en entrée, escaping en sortie pour XSS, WAF en complément.
- Observabilité: logging hors transaction, audit BD, métriques/traces, alertes; tests de restauration réguliers.
- Gouvernance et posture: devoir de conseil, traçabilité des actions sensibles, refus de risque non maîtrisé. Mots-clés:
- NoSQL, embedding, références, schema-less, cardinalité, index, I/O, dénormalisation, MongoDB, Elasticsearch, mapping/nested, agrégation.
- Défense en profondeur, moindre privilège, rôles, CRUD, soft delete, RLS, TLS/mTLS, TDE/FDE, KMS/HSM.
- Audit, journalisation, WORM, SIEM, prepared statements, plan cache, input validation, output escaping, XSS, WAF, CSP.
- Prometheus, Grafana, ELK/Opensearch, OpenTelemetry, pgaudit, performance_schema.
- Sauvegardes, restauration, RPO/RTO, runbooks, game days, PKI interne, service mesh, ETL, caches, search, RGPD.

https://archive.qoyri.fr/mindmaps/cours21.11.25.2.html

Revision #4
Created 1 December 2025 12:34:28 by qoyri

Updated 1 December 2025 12:40:27 by qoyri