SÉCURISATION DES BASES DE DONNÉES

BTS SIO Option SLAM — Épreuve E7

La sécurisation d'une BDD opère à trois niveaux : la structure (contraintes d'intégrité), le comportement (triggers, transactions) et les droits d'accès (DCL). Ces trois niveaux sont complémentaires et constituent la défense en profondeur de la couche données.

1. Contraintes d'Intégrité (DDL)

Les contraintes d'intégrité sont déclarées au niveau du schéma (DDL - Data Definition Language). Elles sont vérifiées automatiquement par le SGBD à chaque INSERT, UPDATE ou DELETE, indépendamment de l'application appelante.

1.1 Catalogue des contraintes

Contrainte	Rôle	Exemple SQL	Erreur si violation
PRIMARY KEY	Unicité + NOT NULL de l'identifiant principal	id INT PRIMARY KEY AUTO_INCREMENT	Duplicate entry / NULL not allowed
UNIQUE	Unicité d'une colonne ou combinaison sans être PK	UNIQUE (email)	Duplicate entry pour la colonne
NOT NULL	Interdit les valeurs NULL	nom VARCHAR(100) NOT NULL	Column cannot be null
FOREIGN KEY	Intégrité référentielle entre tables	FOREIGN KEY (user_id) REFERENCES users(id)	Cannot add or update a child row
CHECK	Valeur dans un intervalle ou respectant une condition	CHECK (age BETWEEN 0 AND 150)	CHECK constraint failed
DEFAULT	Valeur par défaut si non fournie	created_at DATETIME DEFAULT NOW()	(aucune — remplit silencieusement)

1.2 Intégrité référentielle - Options ON DELETE / ON UPDATE

Quand une ligne parent est supprimée ou modifiée, le SGBD applique l'action définie sur les enfants :

Action	Comportement	Usage recommandé
CASCADE	Propage la suppression/modification aux enfants	Relations fortes — ex : commande supprimée → lignes supprimées
SET NULL	Met la FK à NULL dans les enfants	Relations optionnelles — ex : auteur supprimé, articles conservés
RESTRICT / NO ACTION	Bloque l'opération si des enfants existent	Protection des données critiques — comportement par défaut
SET DEFAULT	Met la valeur par défaut dans les enfants	Rare — peu supporté (PostgreSQL uniquement)

SQL

```
CREATE TABLE commandes (
  id          INT PRIMARY KEY AUTO_INCREMENT,
  user_id     INT NOT NULL,
  montant     DECIMAL(10,2) NOT NULL CHECK (montant > 0),
  statut      ENUM('en_attente','validee','annulee') DEFAULT 'en_attente',
  created_at  DATETIME DEFAULT NOW(),
  FOREIGN KEY (user id) REFERENCES utilisateurs(id)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
);
```

> ✓*Les contraintes BDD sont la dernière ligne de défense — même si l'application présente un bug ou qu'un script direct est exécuté sur la BDD, elles s'appliquent toujours.*

2. Triggers - Automatisation et Traçabilité

Un trigger (déclencheur) est une procédure stockée exécutée automatiquement par le SGBD en réponse à un événement DML (INSERT, UPDATE, DELETE) sur une table. Il s'exécute côté serveur BDD, indépendamment de l'application.

2.1 Anatomie d'un trigger

Paramètre	Valeurs possibles	Signification
Moment	BEFORE / AFTER	BEFORE : modifier les données avant insertion | AFTER : réagir après l'opération
Événement	INSERT / UPDATE / DELETE	Opération qui déclenche le trigger
Références	OLD.colonne / NEW.colonne	OLD = valeur avant modif | NEW = valeur après | INSERT : NEW seulement | DELETE : OLD seulement
Granularité	FOR EACH ROW	En MySQL/MariaDB : toujours ligne par ligne

2.2 Trigger de traçabilité (table d'audit)

Le cas d'usage principal en E7 : journaliser automatiquement toute modification sensible dans une table d'audit immuable.

SQL

```
-- Table d'audit (append-only — pas de UPDATE ni DELETE autorisés)
CREATE TABLE audit_utilisateurs (
  id          INT PRIMARY KEY AUTO_INCREMENT,
  user_id     INT NOT NULL,
  action      ENUM('INSERT','UPDATE','DELETE') NOT NULL,
  ancien_email VARCHAR(255),
  nouveau_email VARCHAR(255),
  modifie_par  VARCHAR(100) DEFAULT USER(),
  modifie_le   DATETIME     DEFAULT NOW()
);

-- Trigger AFTER UPDATE sur la table utilisateurs
DELIMITER //
CREATE TRIGGER trg_audit_email
  AFTER UPDATE ON utilisateurs
  FOR EACH ROW
BEGIN
  IF OLD.email <> NEW.email THEN
    INSERT INTO audit_utilisateurs
      (user_id, action, ancien_email, nouveau_email)
    VALUES
      (OLD.id, 'UPDATE', OLD.email, NEW.email);
  END IF;
END //
DELIMITER ;
```

2.3 Trigger de validation métier (BEFORE)

Un trigger BEFORE peut bloquer une opération non conforme en levant une erreur — complément aux contraintes CHECK pour les règles complexes.

SQL

```
-- Interdire un montant de commande supérieur au crédit disponible
DELIMITER //
CREATE TRIGGER trg_check_credit
  BEFORE INSERT ON commandes
  FOR EACH ROW
BEGIN
  DECLARE credit_dispo DECIMAL(10,2);
  SELECT credit_max - encours
    INTO credit_dispo
    FROM clients WHERE id = NEW.client_id;

  IF NEW.montant > credit_dispo THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Crédit insuffisant pour cette commande';
```

```
  END IF;
END //
DELIMITER ;
```

> ⚠ *Un trigger mal écrit peut bloquer silencieusement des opérations légitimes. Toujours tester les cas limites et documenter les triggers dans le schéma BDD.*

2.4 Autres usages des triggers

- **Mise à jour automatique :** recalculer un total, mettre à jour updated_at, dénormaliser une valeur agrégée
- **Synchronisation :** répliquer une modification vers une table d'historique ou d'archive
- **Contrôle de flux :** empêcher la suppression d'un enregistrement référencé par une règle métier (au-delà de la FK)
- **Sécurité renforcée :** déclencher une alerte (via table de notifications) si un compte admin est modifié en dehors des heures ouvrées

3. DCL - Contrôle des Accès (Data Control Language)

Le DCL gère les droits d'accès directement au niveau du SGBD. C'est la mise en œuvre technique du principe du moindre privilège : chaque utilisateur (ou rôle) ne dispose que des droits strictement nécessaires à ses fonctions.

3.1 Commandes DCL fondamentales

SQL

```
-- Créer un utilisateur applicatif (jamais utiliser root en production)
CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'MotDePasseForté!42';

-- Accorder des droits précis sur une BDD
GRANT SELECT, INSERT, UPDATE ON ma_bdd.* TO 'app_user'@'localhost';

-- Accorder sur une table spécifique uniquement
GRANT SELECT ON ma_bdd.produits TO 'reporting_user'@'%';

-- Accorder sur des colonnes spécifiques (granularité maximale)
GRANT SELECT (nom, prenom, email) ON ma_bdd.utilisateurs TO 'support_user'@'localhost';

-- Révoquer un droit
REVOKE DELETE ON ma_bdd.* FROM 'app_user'@'localhost';

-- Appliquer les changements immédiatement
FLUSH PRIVILEGES;

-- Vérifier les droits d'un utilisateur
SHOW GRANTS FOR 'app_user'@'localhost';
```

3.2 Matrice des profils d'accès recommandés

Profil / Rôle	SELECT	INSERT	UPDATE	DELETE	CREATE/DROP	Usage
app_user (applicatif)	✓	✓	✓	✓	✗	Compte utilisé par l'application web
reporting_user	✓	✗	✗	✗	✗	Rapports, tableaux de bord, BI
backup_user	✓	✗	✗	✗	✗	Dumps automatiques (mysqldump)
admin_dba	✓	✓	✓	✓	✓	DBA uniquement — connexion locale, jamais depuis l'app
audit_user	✓	✗	✗	✗	✗	Accès en lecture seule aux tables d'audit

3.3 Vues de sécurité - Masquer les données sensibles

Une vue (VIEW) expose un sous-ensemble contrôlé d'une table. Elle permet de donner accès à certaines colonnes/lignes sans exposer la table complète.

SQL
```sql
-- Vue exposant les utilisateurs sans le hash du mot de passe
CREATE VIEW v_utilisateurs_public AS
  SELECT id, nom, prenom, email, date_inscription
  FROM utilisateurs;  -- colonne 'password_hash' exclue

-- Vue filtrant par département (Row-Level Security partiel)
CREATE VIEW v_employes_rh AS
  SELECT nom, prenom, poste, salaire
  FROM employes
  WHERE departement = 'RH';

-- Accorder accès à la vue uniquement (pas à la table sous-jacente)
GRANT SELECT ON ma_bdd.v_utilisateurs_public TO 'support_user'@'localhost';
REVOKE SELECT ON ma_bdd.utilisateurs FROM 'support_user'@'localhost';
```

4. Transactions et Propriétés ACID

Une transaction est un ensemble d'opérations SQL traitées comme une unité atomique. Si une opération échoue, toutes les autres sont annulées — le principe du « tout ou rien ».

4.1 Propriétés ACID

Propriété	Définition	Exemple concret
A - Atomicité	Toutes les opérations réussissent ou aucune n'est appliquée	Virement : débit ET crédit — si le crédit échoue, le débit est annulé
C - Cohérence	La BDD passe d'un état valide à un autre état valide (contraintes respectées)	Le solde ne peut pas devenir négatif si une contrainte CHECK l'interdit
I - Isolation	Les transactions concurrentes ne s'interfèrent pas	Deux utilisateurs réservant le dernier billet simultanément
D - Durabilité	Une transaction validée (COMMIT) est persistée même en cas de crash	Après COMMIT, les données survivent à une coupure de courant

4.2 Syntaxe et gestion des erreurs

SQL
```sql
-- Exemple : virement bancaire sécurisé
START TRANSACTION;

UPDATE comptes SET solde = solde - 500 WHERE id = 1;
-- Vérification intermédiaire possible
SELECT solde INTO @solde_exp FROM comptes WHERE id = 1;

IF @solde_exp < 0 THEN
  ROLLBACK;  -- Annulation totale (garantie l'atomicité)
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Solde insuffisant';
END IF;

UPDATE comptes SET solde = solde + 500 WHERE id = 2;

COMMIT;  -- Validation définitive et persistance
```

4.3 Niveaux d'isolation

Le niveau d'isolation détermine comment les transactions concurrentes se voient mutuellement. Un niveau élevé = moins de conflits mais plus de verrous (performances réduites).

Niveau	Dirty Read	Non-Repeatable Read	Phantom Read	Usage
READ UNCOMMITTED	Possible	Possible	Possible	Jamais en production - lectures de données non validées
READ COMMITTED	✗	Possible	Possible	Oracle par défaut - bon compromis perf/cohérence
REPEATABLE READ	✗	✗	Possible	MySQL/MariaDB par défaut - sûr pour la plupart des usages
SERIALIZABLE	✗	✗	✗	Maximum de sécurité - transactions séquentielles - coût élevé

SQL
```
-- Définir le niveau d'isolation pour la session
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Ou pour toute la transaction en cours
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
-- ...
COMMIT;
```

5. Procédures Stockées et Fonctions

Une procédure stockée est un programme SQL précompilé et stocké dans le SGBD. Elle constitue une couche d'abstraction entre l'application et les tables, renforçant la sécurité et la maintenabilité.

5.1 Avantages sécurité

- **Élimination des injections SQL :** les paramètres sont automatiquement échappés — l'appel d'une procédure ne peut pas être modifié par injection
- **Séparation des droits :** accorder EXECUTE sur la procédure sans donner SELECT/INSERT sur les tables sous-jacentes
- **Logique métier centralisée :** les règles de validation sont dans la BDD — impossible à contourner depuis l'application
- **Performance :** plan d'exécution précompilé — réduction du temps de traitement pour les opérations répétitives

5.2 Exemple - Procédure d'authentification sécurisée

SQL
```
DELIMITER //
CREATE PROCEDURE sp_authentifier(
  IN  p_email    VARCHAR(255),
  IN  p password VARCHAR(255),
  OUT p_user_id  INT,
  OUT p_resultat VARCHAR(50)
)
BEGIN
  DECLARE v_hash     VARCHAR(255);
  DECLARE v tenta    INT;
  DECLARE v_lock_fin DATETIME;

  -- Vérifier verrouillage du compte
  SELECT tentatives_echec, lock_jusqu_a
    INTO v_tenta, v_lock_fin
    FROM utilisateurs WHERE email = p email;
```

```
  IF v_lock_fin IS NOT NULL AND NOW() < v_lock_fin THEN
    SET p_resultat = 'COMPTE_VERROUILLE';
    LEAVE sp_authentifier;  -- Quitter sans vérifier le mot de passe
  END IF;

  -- Récupérer le hash (l'application vérifie avec password_verify)
  SELECT id, password_hash INTO p_user_id, v_hash
    FROM utilisateurs WHERE email = p_email AND actif = 1;

  -- Note : la vérification Argon2/bcrypt se fait côté application
  -- La procédure retourne le hash pour vérification
  SET p_resultat = IF(p_user_id IS NOT NULL, 'HASH_FOURNI', 'UTILISATEUR_INCONNU');

  -- Log de la tentative
  INSERT INTO audit_connexions (email, succes, ip, tentative_le)
    VALUES (p_email, 0, CONNECTION_ID(), NOW());
END //
DELIMITER ;

-- Appel depuis l'application
CALL sp_authentifier('user@mail.com', 'mdp test', @uid, @res);
SELECT @uid, @res;
```