Université / Département d'Informatique — Master 1 IA/RSD

Examen Semestriel

(Durée ⇨ 90 Minutes)

REMARQUES IMPORTANTES :

1. **Les documents, téléphones et smartphones sont strictement interdits.**
2. **Les réponses doivent être portées sur la présente copie.**
3. **La partie 1 (QCM) est indépendante de la partie 2. Toute note inférieure ou égale à zéro en partie 1 est ramenée à zéro, sans incidence sur la note de la partie 2.**
4. **Répartition du temps conseillée : Partie 1 : ~20 minutes, Partie 2 : ~65 minutes et Relecture : 5 minutes.**

Nom :	Prénom :	Spécialité :	Groupe :	Note :
		☐ IA ☐ RSD		/20

Partie 1 : QCM (10 questions = 10 pts)

Consigne : Cocher UNE SEULE réponse par question.

Barème : Réponse correcte +1 pt, fausse -1 pt, absence de réponse 0 pt

1. **Dans un trigger BEFORE UPDATE, si la fonction retourne NULL, que se passe-t-il ?**

   ☐ L'opération UPDATE se poursuit normalement
   ☐ L'opération UPDATE est annulée silencieusement
   ☐ Une erreur est levée automatiquement
   ☐ Seule la première ligne est mise à jour

2. **Que fait l'instruction RAISE EXCEPTION dans une fonction trigger ?**

   ☐ Elle journalise une erreur sans bloquer l'opération
   ☐ Elle annule l'opération en cours et affiche un message d'erreur
   ☐ Elle redémarre automatiquement le trigger
   ☐ Elle désactive temporairement le trigger

3. **Un trigger peut-il modifier d'autres tables que celle sur laquelle il est défini ?**

   ☐ Non, c'est strictement interdit par PostgreSQL
   ☐ Oui, mais uniquement dans un trigger BEFORE
   ☐ Oui, sans restriction particulière
   ☐ Oui, mais uniquement avec des tables temporaires

4. **Dans quelle situation un trigger BEFORE est-il préférable à un trigger AFTER ?**

   ☐ Pour journaliser une opération
   ☐ Pour valider ou modifier des données avant leur enregistrement
   ☐ Pour déclencher des actions en cascade
   ☐ Pour envoyer des notifications

5. Que signifie FOR EACH ROW dans la définition d'un trigger ?

☐ Le trigger s'exécute une fois par transaction
☐ Le trigger s'exécute pour chaque ligne affectée par l'opération
☐ Le trigger ne s'exécute que sur la première ligne
☐ Le trigger s'exécute en parallèle sur toutes les lignes

6. À quoi sert de désactiver temporairement un trigger sans le supprimer ?

☐ Pour tester des modifications sans perdre la définition du trigger
☐ Pour améliorer les performances lors d'imports massifs de données
☐ Pour effectuer une maintenance temporaire sans recréer le trigger
☐ Toutes les réponses ci-dessus

7. Dans un trigger, que contient la variable TG_OP ?

☐ Le nom de la table concernée
☐ Le type d'opération (INSERT, UPDATE, DELETE)
☐ Le nombre de lignes affectées
☐ L'utilisateur qui a déclenché le trigger

8. Dans un trigger BEFORE UPDATE, retourner OLD au lieu de NEW provoque :

☐ L'annulation de l'opération
☐ La restauration des anciennes valeurs
☐ Une erreur de syntaxe
☐ L'exécution normale avec les nouvelles valeurs

9. Dans un trigger AFTER INSERT, que se passe-t-il si vous tentez de modifier NEW.montant := NEW.montant * 1.3 ?

☐ La modification est appliquée et enregistrée dans la base
☐ La modification est ignorée car l'insertion a déjà eu lieu
☐ Une erreur est levée car NEW est en lecture seule dans AFTER
☐ Le trigger est automatiquement converti en BEFORE

10. Vous avez deux triggers sur la même table : un BEFORE INSERT et un AFTER INSERT. Si le trigger BEFORE retourne NULL, que se passe-t-il avec le trigger AFTER ?

☐ Le trigger AFTER s'exécute quand même
☐ Le trigger AFTER ne s'exécute pas car l'insertion a été annulée
☐ Le trigger AFTER génère une erreur
☐ Les deux triggers s'exécutent en parallèle

Partie 2 : Étude de cas - Plateforme API de Services d'IA Générative

(10 points - 0.5 point par vide de code rempli correctement !)

Vous développez une base de données pour une plateforme d'API d'intelligence artificielle générative (type OpenAI, Anthropic Claude, ou Hugging Face). Voici le schéma simplifié :

SCHÉMA DE LA BASE DE DONNÉES

```
CREATE TABLE utilisateur (
  id                        INTEGER PRIMARY KEY,
  email                     VARCHAR(64) UNIQUE NOT NULL,
  credits_disponibles       INTEGER DEFAULT 1000
                            CHECK (credits_disponibles >= 0),
  total_requetes            INTEGER DEFAULT 0 CHECK (total_requetes >= 0),
  total_tokens_utilises     BIGINT DEFAULT 0 CHECK (total_tokens_utilises >= 0),
  type_compte               VARCHAR(20)DEFAULT 'GRATUIT'
                            CHECK (type_compte IN ('GRATUIT','PRO','ENTREPRISE'))
);
CREATE TABLE requete_ia (
  id                        INTEGER PRIMARY KEY,
  id_utilisateur            INTEGER NOT NULL REFERENCES utilisateur(id),
  modele                    VARCHAR(32) NOT NULL,
  tokens_utilises           INTEGER NOT NULL CHECK (tokens_utilises > 0),
  cout_credits              INTEGER NOT NULL CHECK (cout_credits > 0),
  statut                    VARCHAR(32) DEFAULT 'COMPLETE'
                            CHECK (statut IN ('COMPLETE','ERREUR','ARCHIVE'))
);
```

TRIGGER 1 : Validation des crédits et mise à jour

RÈGLES MÉTIER :

1. Une requête doit consommer entre 10 et 50 000 tokens
2. L'utilisateur doit avoir suffisamment de crédits
3. Déduire le coût et mettre à jour les statistiques

```
CREATE OR REPLACE FUNCTION fn_valider_credits()
RETURNS TRIGGER AS $$
DECLARE
  v_credits INTEGER;
BEGIN
  -- Validation du nombre de tokens
  IF ①_____ < ②_____ OR ③_____ > ④_____ THEN
    RAISE EXCEPTION 'Nombre de tokens invalide (min 10, max 50000)';
  END IF;

  -- Vérification des crédits disponibles
  SELECT ⑤_____ INTO v_credits
  FROM utilisateur
  WHERE id = ⑥_____;
```

```
    IF v_credits < ⑦_____ THEN
    RAISE EXCEPTION 'Crédits insuffisants : % disponible(s), % requis',
                    v_credits, ⑧_____;
  END IF;

  -- Mise à jour des statistiques de l'utilisateur
  UPDATE utilisateur
  SET credits_disponibles = credits_disponibles - ⑨_____,
      total_requetes = ⑩_____ + 1,
      total_tokens_utilises = ⑪_____ + ⑫_____
  WHERE ⑬_____;

  RETURN ⑭_____;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER trg_valider_credits
BEFORE INSERT ON requete_ia
FOR EACH ROW
EXECUTE FUNCTION fn_valider_credits();
```

TRIGGER 2 : Protection des comptes ENTREPRISE

RÈGLE MÉTIER : Un compte ENTREPRISE ayant consommé plus de 1 000 000 tokens ne peut pas être supprimé

```
CREATE OR REPLACE FUNCTION fn_proteger_entreprise()
RETURNS TRIGGER AS $$
BEGIN
  IF ⑮_____ = ⑯_____
     AND ⑰_____ > 1000000 THEN
    RAISE EXCEPTION 'Impossible de supprimer : Compte ENTREPRISE avec % tokens',
                    ⑱_____;
  END IF;

  RETURN ⑲_____;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_proteger_entreprise
⑳_____ DELETE ON utilisateur
FOR EACH ROW
EXECUTE FUNCTION fn_proteger_entreprise();
```