TECHEVENT Romeo Croquelois

Partie 2 :

I. Réaliser et tester les requêtes SQL

1. Liste des participants avec nombre d'événements

```
SELECT participant.*, COUNT(evenement.id_evenement) AS NbEvenements
FROM participant
INNER JOIN inscription ON participant.id_personne = inscription.id_participant
INNER JOIN evenement ON inscription.id_evenement = evenement.id_evenement
GROUP BY participant.id_personne
```

2. Chiffre d'affaires par événement

```
SELECT evenement.id_evenement, evenement.nom, SUM(lignefacture.quantite *
lignefacture.prix_unitaire) AS chiffre_affaires
FROM evenement
INNER JOIN inscription ON evenement.id_evenement = inscription.id_evenement
INNER JOIN facture ON facture.id_participant = inscription.id_participant
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
GROUP BY evenement.id_evenement, evenement.nom
ORDER BY chiffre_affaires DESC;
```

3. Employés sans supérieur

```
SELECT *
FROM employe
WHERE id_superieur IS NULL
```

4. Hiérarchie employé → supérieur

```
SELECT emp.nom AS employe,
      sup.nom AS superieur
FROM Employe e
JOIN Personne emp ON e.id_personne = emp.id_personne
LEFT JOIN Employe es ON e.id_superieur = es.id_personne
LEFT JOIN Personne sup ON es.id_personne = sup.id_personne;
```

5. Participants VIP uniquement

```
SELECT *
FROM participant
WHERE categorie = "VIP";
```

6. Événements sans inscription

```
SELECT evenement.*
FROM evenement
LEFT JOIN inscription ON evenement.id_evenement = inscription.id_evenement
WHERE inscription.id_evenement IS NULL
```

7. Top 5 participants les plus rentables

```
SELECT participant.id_personne, participant.entreprise, SUM(lignefacture.quantite *
lignefacture.prix_unitaire) AS total_facture
FROM participant
INNER JOIN facture ON facture.id_participant = participant.id_personne
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
GROUP BY participant.id_personne, participant.entreprise
ORDER BY total_facture DESC
LIMIT 5;
```

8. Moyenne des salaires

```
SELECT AVG(salaire) AS MoyenneSalaire
FROM employe
```

9. Matériel par technicien

```
SELECT employe.*, materiel.*
FROM employe
INNER JOIN materiel ON employe.id_personne = materiel.id_technicien
WHERE employe.poste = "Technicien"
ORDER BY employe.id_personne
```

10. Total facturé par facture

```
SELECT facture.id_facture, facture.date_facture, participant.entreprise, SUM(lignefacture.quantite
* lignefacture.prix_unitaire) AS total_facture
FROM facture
INNER JOIN participant ON participant.id_personne = facture.id_participant
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
GROUP BY facture.id_facture, facture.date_facture, participant.entreprise
ORDER BY total_facture DESC;
```

11. Participants inscrits à plus de 2 événements

```
SELECT id_participant, COUNT(id_evenement) AS NbEvenement
FROM inscription
GROUP BY id_participant
HAVING NbEvenement >= 2
```

12. Événements avec CA supérieur à la moyenne

```
SELECT  evenement.id_evenement, evenement.nom, SUM(lignefacture.quantite *
lignefacture.prix_unitaire) AS chiffre_affaires
FROM evenement
INNER JOIN inscription ON evenement.id_evenement = inscription.id_evenement
INNER JOIN facture ON facture.id_participant = inscription.id_participant
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
GROUP BY evenement.id_evenement, evenement.nom
HAVING SUM(lignefacture.quantite * lignefacture.prix_unitaire) >
( SELECT AVG(total_event)
 FROM (
    SELECT SUM(lignefacture.quantite * lignefacture.prix_unitaire) AS total_event
    FROM evenement
    INNER JOIN inscription ON evenement.id_evenement = inscription.id_evenement
    INNER JOIN facture ON facture.id_participant = inscription.id_participant
    INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
    GROUP BY evenement.id_evenement
   ) AS sous_requete
)
ORDER BY chiffre_affaires DESC;
```

13. Employés embauchés après 2022

```
SELECT *
FROM employe
WHERE date_embauche > "2022-12-31"
```

14. Nombre de sessions par salle

```
SELECT id_salle, COUNT(*) AS nombre_sessions
FROM session
GROUP BY id_salle;
```

15. Événement avec plus grande capacité totale

```
SELECT evenement.id_evenement, evenement.nom, salle.capacite
FROM evenement
INNER JOIN salle ON salle.id_evenement = evenement.id_evenement
GROUP BY evenement.id_evenement, evenement.nom
ORDER BY salle.capacite DESC
LIMIT 1;
```

16. Participants sans facture

```
SELECT participant.id_personne, participant.entreprise, participant.entreprise, participant.fonction
FROM participant
INNER JOIN facture ON facture.id_participant = participant.id_personne
WHERE facture.id_facture IS NULL;
```

17. Montant total facturé en 2025

```
SELECT SUM(lignefacture.quantite * lignefacture.prix_unitaire) AS total_2025
FROM facture
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
WHERE YEAR(facture.date_facture) = 2025;
```

18. Classement des événements par rentabilité

```
SELECT evenement.id_evenement, evenement.nom AS nom_evenement,
SUM(lignefacture.quantite * lignefacture.prix_unitaire) AS chiffre_affaires
FROM evenement
INNER JOIN inscription ON inscription.id_evenement = evenement.id_evenement
INNER JOIN facture ON facture.id_participant = inscription.id_participant
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
GROUP BY evenement.id_evenement, evenement.nom
ORDER BY chiffre_affaires DESC;
```

19. Matériel valeur > 5000€

```
SELECT materiel.id_materiel, materiel.designation, materiel.type, materiel.etat,
materiel.valeur_achat
FROM materiel
WHERE materiel.valeur_achat > 5000;
```

20. Nombre d’inscriptions par catégorie

```
SELECT participant.categorie, COUNT(inscription.id_inscription) AS nombre_inscriptions
FROM participant
INNER JOIN inscription ON inscription.id_participant = participant.id_personne
GROUP BY participant.categorie
ORDER BY nombre_inscriptions DESC;
```

21. Participants ayant annulé

```
SELECT id_participant
FROM inscription
WHERE statut = "Annulée"
```

22. Employés ayant au moins 3 subordonnés

```
SELECT superieur.id_personne, superieur.poste, superieur.salaire, superieur.date_embauche,
COUNT(subordonne.id_personne) AS nombre_subordonnes
FROM employe AS superieur
INNER JOIN employe AS subordonne ON subordonne.id_superieur = superieur.id_personne
GROUP BY superieur.id_personne, superieur.poste, superieur.salaire, superieur.date_embauche
HAVING COUNT(subordonne.id_personne) >= 3
ORDER BY nombre_subordonnes DESC;
```

23. Taux de confirmation (%)

```
SELECT id_evenement, (COUNT(CASE WHEN statut = 'Confirmée' THEN 1 END) * 100.0 /
COUNT(*)) AS taux_confirmation
FROM inscription
GROUP BY id_evenement;
```

24. Plus ancienne date d’embauche

```
SELECT employe.id_personne, employe.date_embauche, employe.poste, employe.salaire
FROM employe
WHERE employe.date_embauche = (
   SELECT MIN(date_embauche)
   FROM employe
);
```

25. Factures sans ligne

```
SELECT facture.id_facture, facture.date_facture, facture.id_participant
FROM facture
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
WHERE lignefacture.id_ligne IS NULL;
```

26. Sessions par intervenant

```
SELECT
   id_intervenant,
   COUNT(*) AS nombre_sessions
FROM session
GROUP BY id_intervenant;
```

27. Événement avec le plus de sessions

```
SELECT evenement.id_evenement, evenement.nom, COUNT(session.id_session) AS
nombre_sessions
FROM evenement
INNER JOIN salle ON salle.id_evenement = evenement.id_evenement
INNER JOIN session ON session.id_salle = salle.id_salle
GROUP BY evenement.id_evenement, evenement.nom
ORDER BY nombre_sessions DESC
LIMIT 1;
```

28. CA moyen par participant

```
SELECT AVG(total_par_participant) AS ca_moyen_par_participant
FROM (
   SELECT participant.id_personne, SUM(lignefacture.quantite * lignefacture.prix_unitaire) AS
total_par_participant
   FROM participant
   INNER JOIN facture ON facture.id_participant = participant.id_personne
   INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
   GROUP BY participant.id_personne
) AS sous_requete;
```

29. Participants multi-catégories (anomalie)

```
SELECT participant.id_personne, participant.entreprise, COUNT(DISTINCT participant.categorie)
AS nombre_categories
FROM participant
GROUP BY participant.id_personne, participant.entreprise
HAVING COUNT(DISTINCT participant.categorie) > 1;
```

30. Total global entreprise

```
SELECT participant.entreprise, SUM(lignefacture.quantite * lignefacture.prix_unitaire) AS
total_global
FROM participant
INNER JOIN facture ON facture.id_participant = participant.id_personne
INNER JOIN lignefacture ON lignefacture.id_facture = facture.id_facture
GROUP BY participant.entreprise
ORDER BY total_global DESC;
```

II. Réaliser et tester les triggers

1. Empêcher salaire négatif

```
CREATE TRIGGER verif_salaire_non_negatif
BEFORE INSERT ON employe
FOR EACH ROW
BEGIN
   IF NEW.salaire < 0 THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Erreur : le salaire ne peut pas être négatif.';
   END IF;
END;
```

2. Empêcher suppression d’un employé ayant du matériel

```
CREATE TRIGGER verif_suppression_employe_materiel
BEFORE DELETE ON employe
FOR EACH ROW
BEGIN
   IF EXISTS (
      SELECT 1
      FROM materiel
      WHERE materiel.id_employe = OLD.id_personne
   ) THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Impossible de supprimer cet employé : du matériel lui est
attribué.';
   END IF;
END;
```

3. Calcul automatique total facture

```
CREATE TRIGGER maj_total_facture_insert
AFTER INSERT ON lignefacture
FOR EACH ROW
BEGIN
   UPDATE facture
   SET total_facture = (
      SELECT SUM(quantite * prix_unitaire)
      FROM lignefacture
      WHERE id_facture = NEW.id_facture
   )
   WHERE id_facture = NEW.id_facture;
END;
```

4. Empêcher inscription si événement complet

```
CREATE TRIGGER verif_evenement_complet
BEFORE INSERT ON inscription
FOR EACH ROW
BEGIN
   DECLARE nb_inscriptions INT;
   SELECT COUNT(*) INTO nb_inscriptions
   FROM inscription
   WHERE id_evenement = NEW.id_evenement;
   IF nb_inscriptions >= (
      SELECT capacite
      FROM evenement
      WHERE id_evenement = NEW.id_evenement
   ) THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Impossible : cet événement est complet.';
   END IF;
END;
```

5. Empêcher doublon inscription

```
CREATE TRIGGER verif_doublon_inscription
BEFORE INSERT ON inscription
FOR EACH ROW
BEGIN
   IF EXISTS (
      SELECT 1
      FROM inscription
      WHERE id_participant = NEW.id_participant
        AND id_evenement = NEW.id_evenement
   ) THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Impossible : ce participant est déjà inscrit à cet événement.';
   END IF;
END;
```

6. Statut par défaut

```
CREATE TRIGGER defaut_statut_inscription
BEFORE INSERT ON inscription
FOR EACH ROW
BEGIN
   IF NEW.statut IS NULL OR NEW.statut = '' THEN
      SET NEW.statut = 'En attente';
   END IF;
END;
```

7. Historisation suppression

```
CREATE TABLE employe_historique (
    id_historique INT AUTO_INCREMENT PRIMARY KEY,
    id_personne INT,
    date_embauche DATE,
    salaire DECIMAL(10,2),
    poste VARCHAR(100),
    id_superieur INT,
    date_suppression DATETIME
);


CREATE TRIGGER historisation_employe_suppression
BEFORE DELETE ON employe
FOR EACH ROW
BEGIN
    INSERT INTO employe_historique (
        id_personne,
        date_embauche,
        salaire,
        poste,
        id_superieur,
        date_suppression
    ) VALUES (
        OLD.id_personne,
        OLD.date_embauche,
        OLD.salaire,
        OLD.poste,
        OLD.id_superieur,
        NOW()
    );
END;
```

8. Vérifier dates événement

```
CREATE TRIGGER verif_dates_evenement
BEFORE INSERT ON evenement
FOR EACH ROW
BEGIN
    IF NEW.date_debut > NEW.date_fin THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Erreur : la date de début ne peut pas être après la date de fin.';
    END IF;
END;
```

9. Empêcher auto-hiérarchie

```
CREATE TRIGGER verif_auto_hierarchie_employe
BEFORE INSERT ON employe
FOR EACH ROW
BEGIN
   IF NEW.id_superieur = NEW.id_personne THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Erreur : un employé ne peut pas être son propre supérieur.';
   END IF;
END;
```

10. Bloquer suppression facture payée

```
CREATE TRIGGER verif_suppression_facture_inscription_confirmee
BEFORE DELETE ON facture
FOR EACH ROW
BEGIN
   IF EXISTS (
      SELECT 1
      FROM inscription
      WHERE id_participant = OLD.id_participant
        AND statut = 'Confirmée'
   ) THEN
      SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Impossible de supprimer cette facture : une inscription associée
est Confirmée.';
   END IF;
END;
```

III. Réaliser et tester les procédures stockées

1. Ajouter un participant

```
CREATE PROCEDURE ajouter_participant (
   IN p_entreprise VARCHAR(100),
   IN p_fonction VARCHAR(100),
   IN p_categorie VARCHAR(50)
)
BEGIN
   INSERT INTO participant (entreprise, fonction, categorie)
   VALUES (p_entreprise, p_fonction, p_categorie);
END;
```

2. Inscrire un participant

```
CREATE PROCEDURE inscrire_participant (
   IN p_id_participant INT,
   IN p_id_evenement INT,
   IN p_statut VARCHAR(50)
)
BEGIN
   IF p_statut IS NULL OR p_statut = '' THEN
      SET p_statut = 'En attente';
   END IF;
   INSERT INTO inscription (id_participant, id_evenement, statut)
   VALUES (p_id_participant, p_id_evenement, p_statut);
END;
```

3. CA par événement

```
CREATE PROCEDURE ca_par_evenement (
   IN p_id_evenement INT,
   OUT p_CA DECIMAL(10,2)
)
BEGIN
   DECLARE nb INT DEFAULT 0;
   DECLARE prix DECIMAL(10,2) DEFAULT 0;
   SELECT tarif INTO prix
   FROM evenement
   WHERE id_evenement = p_id_evenement;
   SELECT COUNT(*) INTO nb
   FROM inscription
   WHERE id_evenement = p_id_evenement
     AND statut = 'Confirmée';
   SET p_CA = prix * nb;
END;
```

4. Liste participants événement

```
CREATE PROCEDURE liste_participants_evenement (
   IN p_id_evenement INT
)
BEGIN
   SELECT
      p.id_personne,
      p.entreprise,
      p.fonction,
      p.categorie,
      i.statut
   FROM participant p
   INNER JOIN inscription i
      ON p.id_personne = i.id_participant
   WHERE i.id_evenement = p_id_evenement;
END;
```

5. Total facturé participant

```
CREATE PROCEDURE total_facture_participant (
   IN p_id_participant INT,
   OUT p_total DECIMAL(10,2)
)
BEGIN
   SELECT SUM(l.quantite * l.prix_unitaire)
   INTO p_total
   FROM facture f
   INNER JOIN lignefacture l
      ON f.id_facture = l.id_facture
   WHERE f.id_participant = p_id_participant;
END;
```

6. Matériel par employé

```
CREATE PROCEDURE materiel_par_employe (
   IN p_id_employe INT
)
BEGIN
   SELECT
      m.id_materiel,
      m.designation,
      m.type,
      m.etat,
      m.valeur_achat,
      m.id_technicien,
      a.date_attribution,
      a.date_retour
   FROM materiel m
   INNER JOIN attribution a
      ON m.id_materiel = a.id_materiel
   WHERE a.id_employe = p_id_employe;
END;
```

7. Augmenter salaires %

```
CREATE PROCEDURE augmenter_salaires (
   IN p_pourcentage DECIMAL(5,2)
)
BEGIN
   UPDATE employe
   SET salaire = salaire + (salaire * p_pourcentage / 100);
END;
```

8. Statistiques annuelles

```
CREATE PROCEDURE statistiques_annuelles (
   IN p_annee INT,
   OUT p_CA DECIMAL(10,2)
)
BEGIN
   SELECT SUM(l.quantite * l.prix_unitaire)
   INTO p_CA
   FROM facture f
   INNER JOIN lignefacture l
      ON f.id_facture = l.id_facture
   WHERE YEAR(f.date_facture) = p_annee;
END;
```

9. Rapport complet événement

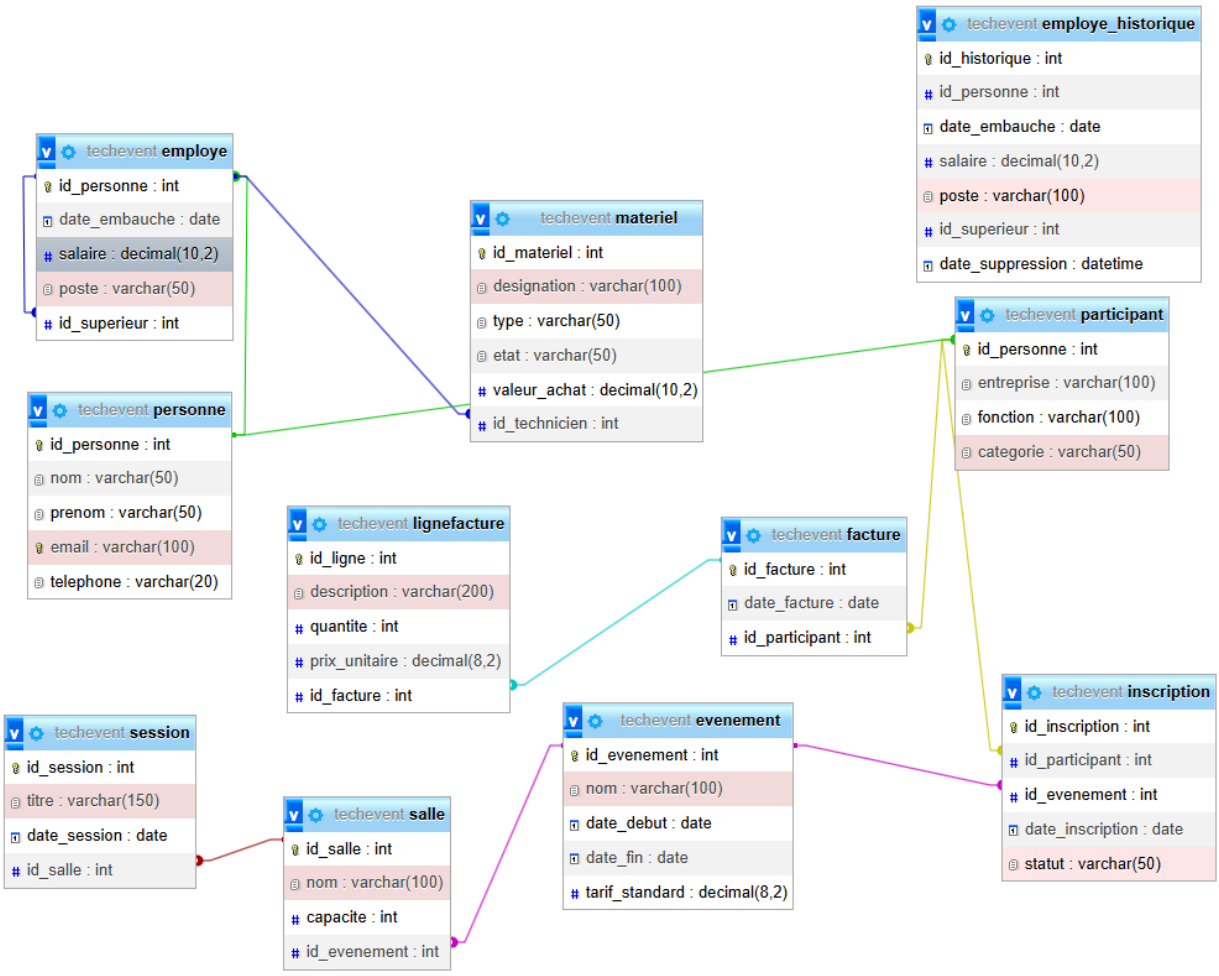
```
CREATE PROCEDURE rapport_evenement (
   IN p_id_evenement INT
)
BEGIN
   SELECT
      e.id_evenement,
      e.nom,
      e.date_debut,
      e.date_fin,

      -- Nombre total d'inscriptions
      (SELECT COUNT(*)
       FROM inscription
       WHERE id_evenement = p_id_evenement) AS total_inscriptions,

      -- Nombre d'inscriptions Confirmées
      (SELECT COUNT(*)
       FROM inscription
       WHERE id_evenement = p_id_evenement
         AND statut = 'Confirmée') AS inscriptions_confirmees,

      -- CA total (simple)
      (SELECT SUM(l.quantite * l.prix_unitaire)
       FROM facture f
       INNER JOIN lignefacture l ON f.id_facture = l.id_facture
       WHERE f.id_participant IN (
          SELECT id_participant
          FROM inscription
          WHERE id_evenement = p_id_evenement
       )
      ) AS chiffre_affaires

   FROM evenement e
   WHERE e.id_evenement = p_id_evenement;
END;
```


techevent employe_historique
id_historique : int
id_personne : int
date_embauche : date
salaire : decimal(10,2)
poste : varchar(100)
id_superieur : int
date_suppression : datetime
techevent employe
id_personne : int
date_embauche : date
salaire : decimal(10,2)
poste : varchar(50)
id_superieur : int
techevent materiel
id_materiel : int
designation : varchar(100)
type : varchar(50)
etat : varchar(50)
valeur_achat : decimal(10,2)
id_technicien : int
techevent participant
id_personne : int
entreprise : varchar(100)
fonction : varchar(100)
categorie : varchar(50)
techevent personne
id_personne : int
nom : varchar(50)
prenom : varchar(50)
email : varchar(100)
telephone : varchar(20)
techevent lignefacture
id_ligne : int
description : varchar(200)
quantite : int
prix_unitaire : decimal(8,2)
id_facture : int
techevent facture
id_facture : int
date_facture : date
id_participant : int
techevent inscription
id_inscription : int
id_participant : int
id_evenement : int
date_inscription : date
statut : varchar(50)
techevent session
id_session : int
titre : varchar(150)
date_session : date
id_salle : int
techevent salle
id_salle : int
nom : varchar(100)
capacite : int
id_evenement : int
techevent evenement
id_evenement : int
nom : varchar(100)
date_debut : date
date_fin : date
tarif_standard : decimal(8,2)