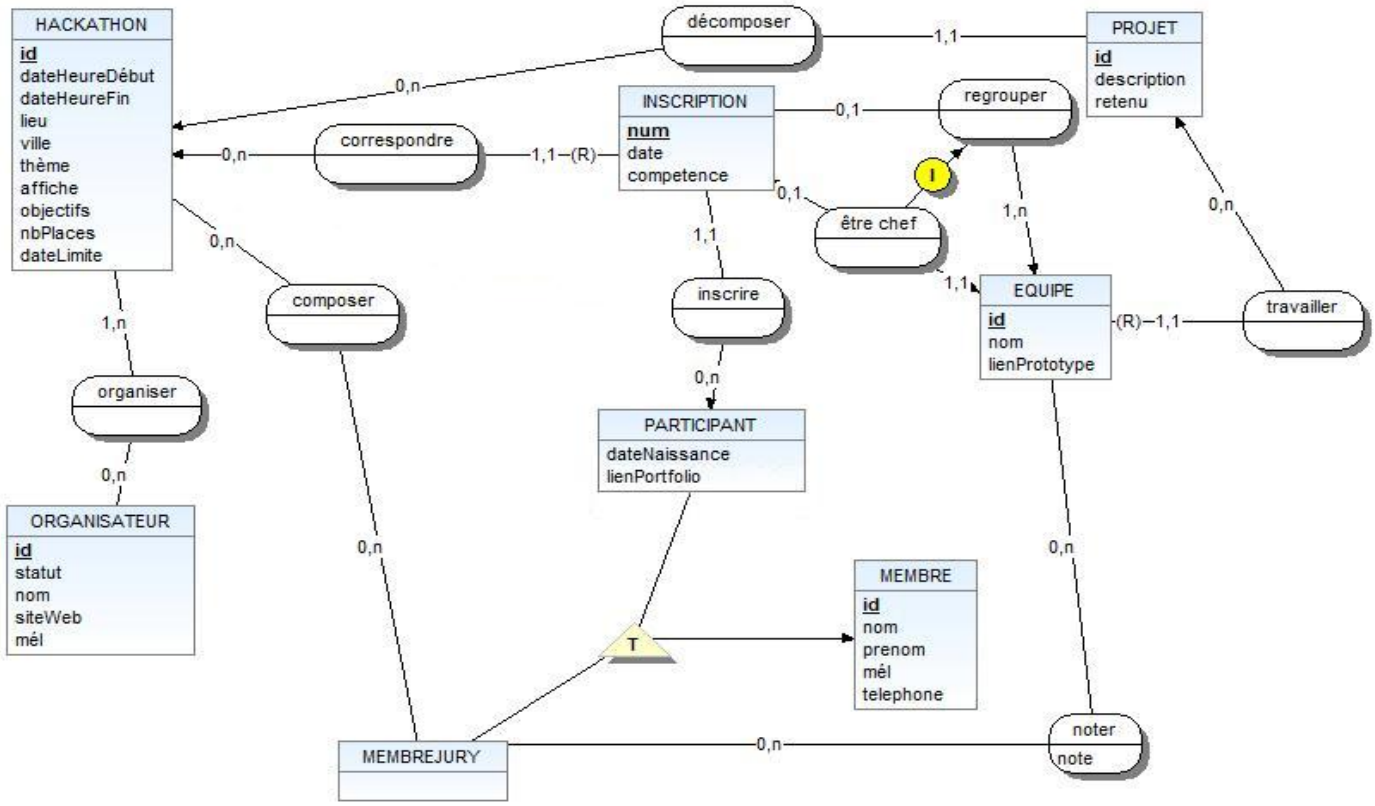
Éléments de correction : HACKAT'INNOV

Dossier A – Gestion des participants (30 points)

Mission A.1 – Évolution de la base de données pour la gestion des hackathons

Question A.1
Proposer une modification de la base de données utilisée par l'application existante *Hackat'Orga* prenant en compte les inscriptions, la gestion des équipes et le vote final.

Schéma Entité-Association :



L'entité faible EQUIPE est spécifiée par « Plusieurs équipes peuvent choisir le même projet, elles seront alors identifiées relativement au projet choisi. »

Contraintes (Les contraintes peuvent être exprimées soit graphiquement directement sur le schéma, soit textuellement.) :

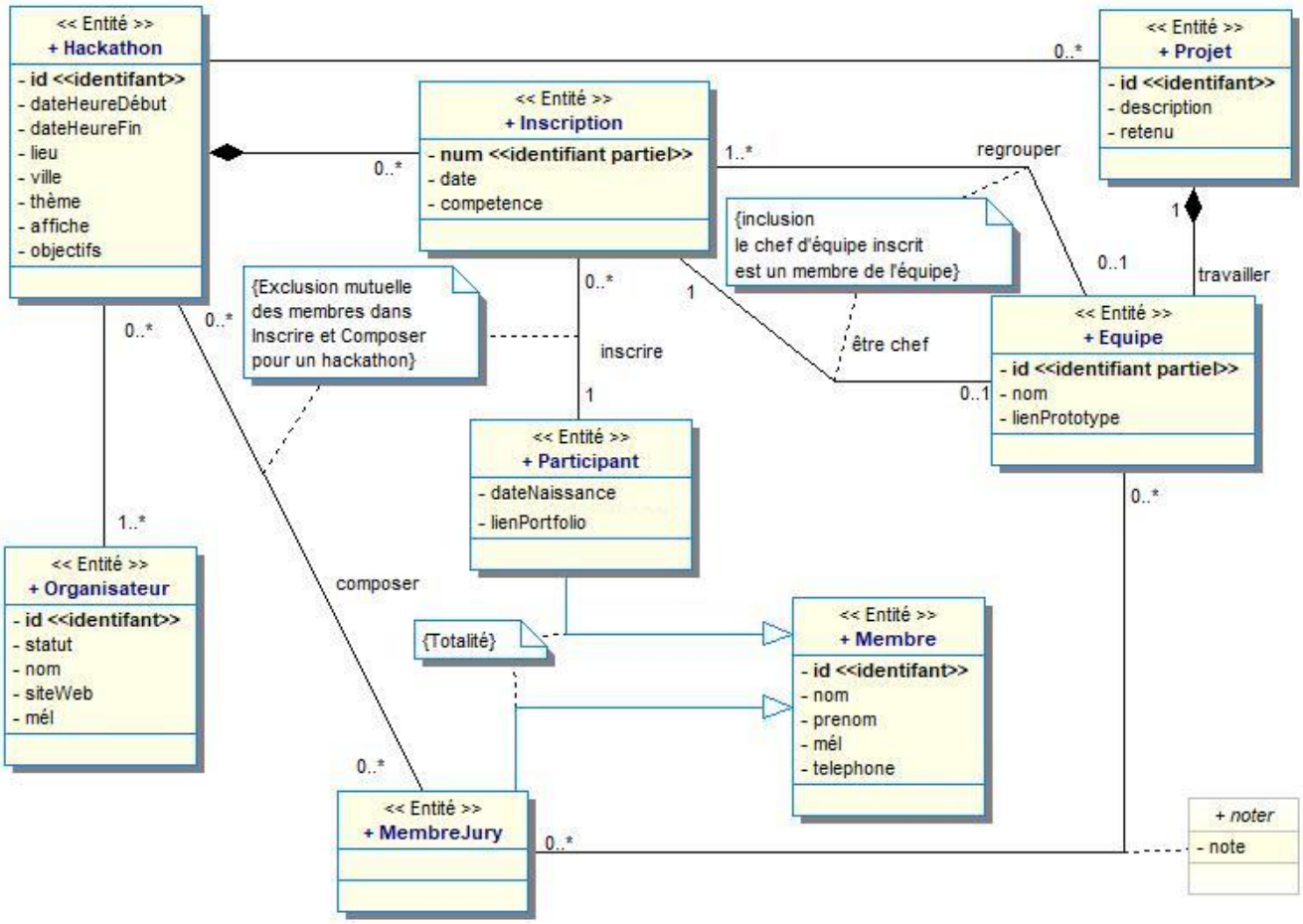
- Bloc 1 : La contrainte d'héritage de TOTALITE sans EXCLUSION exprime qu'un membre du JURY peut aussi être PARTICIPANT.
- Bloc 3 : l'inclusion entre « être chef » et « regrouper » permet de préciser que « Un chef de projet est défini dans chaque équipe parmi ses participants ». La notion de CHEF ne peut pas être indiquée dans PARTICIPANT puisqu'un participant peut utiliser son même profil pour plusieurs hackathons mais ne sera pas forcément chef à chaque fois.

- Une contrainte permettant de traduire le fait qu'on ne peut pas être à la fois PARTICIPANT et JURY pour un même hackathon n'est pas exigée. La tentative de représenter une telle contrainte (par exemple par une EXCLUSION entre composer et inscrire) est valorisée par compensation d'une erreur ou omission qui aurait été faite ailleurs dans le schéma.

Variante acceptée : La notion de CHEF peut être indiquée dans INSCRIPTION par un booléen bien qu'elle ne soit pas optimale.

Une réponse à l'aide du modèle relationnel est acceptée. La traduction de l'héritage par une seule table est acceptée si elle est argumentée.

Diagramme UML :



Mission A.2 – Édition du planning des phases d'un hackathon

Question A.2.1
Justifier que la structure de la base de données permet d'implémenter cette règle de gestion.

La clé primaire composée de la relation PLANNING exprime qu'**un hackathon** à **une heure et date donnée** ne peut proposer qu'**une seule phase**.
Il y a donc une contrainte d'unicité (ou une agrégation d'associations) sur l'association PLANNING.

Question A.2.2
Écrire l'ordre SQL de création de la table PLANNING

```
CREATE TABLE PLANNING (
    idHackathon int,
    dateHeureDebut dateTime,
    idPhase int NOT NULL,
    duree int,
    CONSTRAINT PK_PLANNING PRIMARY KEY (idHackathon, dateHeureDebut),
    CONSTRAINT FK_Hackathon FOREIGN KEY (idHackathon)
            REFERENCES HACKATHON (id),
    CONSTRAINT FK_Phase FOREIGN KEY (idPhase) REFERENCES PHASE (id)
);
```

Dossier B : Gestion d'évènements organisés autour des *hackathons* (35 points)

Question B.1 – Corrections des erreurs signalées ci-dessus

a) Corriger la méthode *ajouterMateriel.*

```
public function ajouterMateriel($unLibelleMateriel, $uneQuantite) {
    if ($unLibelleMateriel != null )
        if (! array_key_exists($unLibelleMateriel, $this->lesMateriels) && $uneQuantite > 0 ){
        //on vérifie que la clé n'existe pas déjà dans le dictionnaire
        $this->lesMateriels[$unLibelleMateriel] = $uneQuantite;
        /*ajoute dans le dictionnaire la clé de type String correspondant au libelleMateriel et la
            valeur de type entier correspondant à la quantité demandée*/
        }
    }
```

b) Écrire le constructeur de la classe Initiation.

```
public function __construct($libelle,$dateHeure, $duree, $salle, $lAnimateur, $leTypePublic, $nb){
    parent::__construct($libelle,$dateHeure, $duree, $salle, $lAnimateur, $leTypePublic);
    //appel au constructeur de la classe mère
    $this->nbplaces = $nb;
    //instanciation du dictionnaire
    $this->lesMateriels = array();
    // instanciation liste de membre
    $this->lesMembresParticipants = array();
}
```

c) Écrire la méthode *ajouterParticipant.*

```
public function ajouterParticipant($unMembre){
    $result = false;
    if ($unMembre != null   &&
        count($this->lesMembresParticipants ) < $this->nbPlaces){
        $this->lesMembresParticipants[] = $unMembre;
    // autre solution arrayPush($this->lesMembresParticipants, $unMembre);
        $result = true;
    }
    return $result;
}
```

d) Écrire la méthode *lesParticipantsToJson*.

```
private function lesParticipantsToJson(){
        // retourne une chaîne de caractères
        $chaineJson = "\"participants\" : [  \n";
        $debutChaine = true;
        foreach ($this->lesMembresParticipants as $unMembre) {
                if ($debutChaine == false) {
                        // on ajoute un élément supplémentaire, on le sépare par une virgule
                        $chaineJson .= ",\n";
                }
                else {
                        $debutChaine = false ;
                }
                $chaineJson .= $unMembre->toJson();
        }
        $chaineJson .= "]  \n";
        return $chaineJson;
}
```

e) Écrire la méthode *toJson*.

```
public function toJson(){
        //Ecrire la méthode qui retourne une chaîne au format JSON
        //contenant les différentes caractéristiques avec la liste des matériels
        //nécessaires à apporter.Cette méthode servira aussi à afficher la liste des
        //participants quand il y en aura.
        $chaineJson = parent::toJson();
        $chaineJson .= ", \n \"nbPlaces\" : \"" . $this->nbplaces . "\", \n";
        $chaineJson .= $this->lesMaterielsToJson();
        if (count($this->lesMembresParticipants) > 0) {
                $chaineJson .= ", \n".$this->lesParticipantsToJson();
        }
        $chaineJson .= "}";
        return $chaineJson;
}
```

Question B.2 – Gestion du planning des évènements des membres au sein de l'hackathon.

Décrire la structure du dictionnaire retourné par la méthode *planningParParticipant* de la classe Hackathon.

La méthode *planningParParticipant*() de la classe est un tableau associatif (ou toute autre solution présentant un mécanisme clé-valeur) associant comme clé une chaîne qui contient la date et l'heure, et comme valeur un objet de type Initiation.

Le dictionnaire retourné contient la liste des événements de type initiation (valeur) du participant dont l'adresse mél est passée en paramètre. Cette liste est indexée par la "dateHeure" d'événement (clé).

Dossier C : Gestion des recherches pour le mobile (20 points)

Question C.1

Expliquer par une phrase l'objectif du code encadré qui vient d'être rajouté dans le fichier *liste.vue* (document 10).

Il permet d'afficher directement dans la vue le nombre d'hackathons (*length*) qui s'appuie sur la liste lesHackathons chargée depuis l'interface de programmation (*API*).

Question C.2

Compléter le code du fichier *liste.vue* (document 10) et *l'API* correspondante en indiquant les lignes concernées.

Rajouter dans le fichier *liste.vue* (*javascript*) : l'entête et les données dans le tableau

Modifications après la ligne 22 :

```
            <th>Date</th>
            <th>Ville</th>
            <th>Thème</th>
        </tr>
        <tr v-for="hackathon in lesHackathons" :key="hackathon.id">
            <td>{{hackathon.dateDebut}}</td>
            <td>{{hackathon.ville}}</td>
            <td>{{hackathon.theme}}</td>
        </tr>
</table>
```

Fichier *api-rechercher.php* (*API*) : modification de la requête

```
// Requete de recherche.
$stmt = $pdo->prepare('
    SELECT DATE_FORMAT(dateHeureDebut, "%d/%m/%Y") AS dateDebut, ville, theme
    FROM HACKATHON
    WHERE ville LIKE :critere
    OR theme LIKE :critere ORDER BY dateDebut, villle');
$stmt->bindValue(':critere', "%$critere%", PDO::PARAM_STR);
$stmt->execute();
```

Question C.3

a) Proposer un commentaire expliquant l'utilité de la jointure de type **LEFT OUTER JOIN** dans cette requête.

La table Évènement est la table maîtresse dans cette jointure. Chaque évènement sera complété par les éléments de conférences et d'initiation s'ils sont présents ou affectés de *NULL*.

Sans LEFT *OUTER*, JOIN ne renverrait aucun résultat car aucun événement ne remplit les deux conditions de jointure.

b) Compléter cette requête *SQL* en ajoutant la date, l'heure et la salle de la conférence avec le nom et le prénom de l'animateur. Les évènements seront triés chronologiquement.

En gras ce qui a été complété :

```
SELECT libelle, dateHeure, salle, nbplaces, theme, nom, prenom
FROM EVENEMENT E
LEFT OUTER JOIN INITIATION I ON E.id = I.idEvenementInit
LEFT OUTER JOIN CONFERENCE C ON E.id = C.idEvenementConf
INNER JOIN MEMBRE M ON M.id = E.idAnimateur
ORDER by dateHeure;
```

Question C.4

Créer une vue en langage *SQL* permettant de retourner les identifiants de tous les évènements de type « initiation » avec, pour chacun, le nombre de membres inscrits.

```
CREATE VIEW V_NBINSCRITS_EVT (idEvtInit, nbInscrits) AS
    SELECT I.idEvenementInit, count(idMembre)
    FROM INITIATION I
    LEFT OUTER JOIN INSCRIRE INS ON I.idEvenementInit = INS.idEvenementInit
    GROUP BY I.idEvenementInit
```

On acceptera toute autre solution SQL qui ramène les bons résultats (UNION de deux requêtes pour simuler LEFT OUTER JOIN).

Dossier D : Gestion des votes (15 points)

Question D.1

Décrire une conséquence d'utilisation malveillante de l'application de gestion des votes.

- Effacer un ou plusieurs votes.
- Modifier un vote.
- Voter plusieurs fois.
- Voter en usurpant l'identité d'un autre participant.
- Voter en usurpant l'identité d'un membre jury.

Une réponse est cohérente si elle concerne bien l'application de gestion des votes. Par exemple, une réponse portant sur l'étape d'initialisation ou de publication d'un hackathon ne peut pas être acceptée. De même une réponse expliquant qu'un membre du jury a été soudoyé n'est pas non plus une utilisation malveillante de l'application de gestion des votes.

Question D.2

Préciser les intérêts de mettre le serveur de bases de données en dehors de la zone démilitarisée (*DMZ)* pour aider Mme Mabille à compléter son étude.

Une zone démilitarisée est un sous-réseau contenant les machines accessibles depuis Internet. Ce sous-réseau est isolé du réseau local et protégé par un pare-feu. Le serveur de base de données n'a pas besoin d'être accessible depuis Internet. Le placer en dehors de la DMZ pour le rendre inaccessible depuis Internet a pour principaux objectifs de :

- **Sécuriser l'accès** à la base de données
- **Protéger les données contre les attaques venant de l'extérieur**
- Participer à la préservation de l'**intégrité** et de la **confidentialité** de la base de données.

Question D.3

Présenter sous forme d'un tableau le ou les éléments de l'architecture applicative concerné(s) par chacun des points de vigilance relevés.

- Sécuriser le service http (https)	- **APACHE**
- Utiliser des requêtes préparées	- **API**
- Se prémunir des attaques par injection SQL	- **API**
- Se connecter via un *Wi-Fi* sécurisé	- ***Android* côté client**
- Préserver l'intégrité de données (droits d'accès – défaut de configuration)	- **BDD + API**
- S'assurer de la mise à jour des librairies et des composants logiciels	- ***Android*, API, Apache, BDD**

On acceptera pour l'item 3 « Se prémunir des attaques par injection SQL » la solution **API** + ***Android* côté client**. Cependant, ***Android* côté client** seul ne suffit pas pour éviter les injections.