# TD N°8 : uberisation

**Thème :** modélisation d'une application de transport de personnes

## Contexte

La société de transport de personnes TOPCHRONO souhaite mettre en place une plateforme extranet à destination des particuliers. L'application proposera des services similaires à ceux proposés par la plateforme UBER :

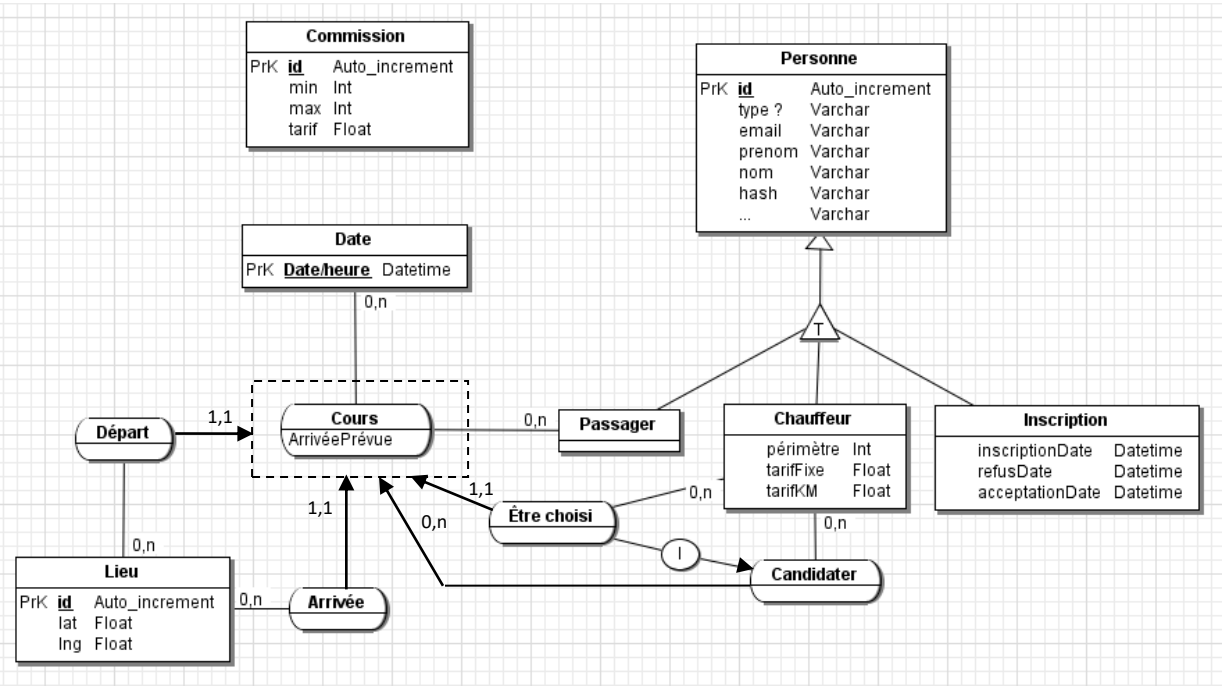
- La plateforme répertorie les chauffeurs et les passagers. Diverses informations sont conservées à leur sujet : adresse email, prénom, nom, « hash » du mot de passe, etc. Rien n'empêche a priori qu'un chauffeur en repos puisse être passager !
- TOPCHRONO effectue une qualification de tous les chauffeurs, à savoir qu'un chauffeur effectue une inscription préalable. Son inscription n'est « convertie » en un compte chauffeur qu'une fois l'inscription validée. Afin de pouvoir afficher le périmètre d'intervention d'un chauffeur, on souhaite pouvoir conserver son adresse principale sous la forme d'un couple longitude et latitude. Il appartient au chauffeur de choisir son périmètre d'intervention. On suppose que le périmètre d'intervention du chauffeur est un cercle.
- Un passager propose une course en partance d'un lieu et à destination d'un autre lieu. Pour des raisons d'affichage encore, les lieux sont stockés sous la forme de couples longitude/latitude. La course intervient à une date et une heure donnée. Le logiciel se charge de fournir une estimation de la date et de l'heure d'arrivée.
- Les chauffeurs peuvent consulter les courses qui sont dans leur périmètre d'intervention. Si la course est dans ce périmètre, ils peuvent proposer leurs services à titre de candidats. Ils peuvent alors consulter les coordonnées du passager et le contacter. Il appartient au passager de choisir le chauffeur qui réalisera sa course. Pour l'accompagner dans sa décision, grâce à la plateforme, le passager peut consulter le profil des chauffeurs et son tarif. La plateforme fournie, pour chaque chauffeur une estimation du tarif pratiqué ;
- Dans la première version, le tarif du chauffeur est tarif forfaitaire (montant fixe) assorti d'un tarif kilométrique ;
- Pour chaque course, la longueur du trajet est stockée. Elle est arrondie au kilomètre le plus proche. Elle est obtenue grâce à l'API Google Maps. Cette longueur sert de base de calcul à l'estimation du tarif pratiqué par le chauffeur.
- Par ailleurs, TOPCHRONO, société soucieuse de respecter ses partenaires, chauffeurs et passages, prélève une commission d'apporteur d'affaire sur chacune des candidatures des chauffeurs. La commission est un montant fixe fonction de la longueur du trajet. Le montant est fixé au moyen d'un barème progressif (exemple : 5€ si le trajet a une longueur comprise entre 0km et 50km exclus, 9€ s'il est compris entre 50km et 100km exclus, etc.).

## Travail à faire

# 1. Modélisation

| | |
|---|---|
| 1.1. | Modéliser la base de données du logiciel sous forme de MCD (ou de diagramme de classes) au regard des informations fournies. |
| 1.2. | Rédiger le schéma relationnel correspondant à votre MCD. |
| 1.3. | Afin de mieux répondre aux attentes des chauffeurs, proposer une évolution leur permettant de fixer un tarif kilométrique dégressif, à savoir diminuant (ou augmentant) en fonction de la longueur du trajet. |

## 1.1. MCD




Ce qu'il fallait voir :

- Une contrainte d'inclusion entre les associations « Candidater » (chauffeur candidat à un course) et « Être choisi » (chauffeur choisi par le passager) où la flèche est orientée vers l'association « Candidater ». Signification : être choisi implique être candidat.
- Un héritage de type « T » (totalité), avec : une entité parente « Personne » et des entités filles « Chauffeur » et « Passager », voire « Inscription » ;
- L'entité « inscription » peut être soit une entité fille de « Personne » soit une entité indépendante reprenant toutes les propriétés du chauffeur, y compris celles héritées de « Personne » ;
- Une représentation convenable d'une « Course », la plus exacte étant un agrégat : une course est définie par un passager donnée pour une date donnée. Le couple (passager, date) suffit à identifier une course. Une alternative : une entité « Course » relative à un passager et une date.

### 1.2. Schéma relationnel

**Personne**(id, email, prénom, nom, hash, type…)
Clef primaire : id
Clef étrangère : ---
*N.B. : on a choisi d'ajouter la propriété discriminante type pouvant prendre une combinaison constituée des valeurs « C » (chauffeur), « P » (passager), « I » (inscription). Exemples : P, CI, PCI.*

**Chauffeur**(id, lat, lng, périmètre, tarifFixe, tarifKM)
Clef primaire : id
Clef étrangère : id en référence à Personne(Id)
*N.B. : les propriété « lat » et « lng » représentent respectivement la latitude et la longitude de l'adresse principale du chauffeur. La propriété périmètre est une distance exprimée en km (entier).*

**Passager**(id)
Clef primaire : id
Clef étrangère : id en référence à Personne(Id)
*N.B. : en pratique, si le passager n'a pas de propriété supplémentaire, on peut donc se passer de l'entité Passager. Un passager est une personne telle que Personne.type LIKE %P%.*

**Inscription**(id, inscriptionDate, refusDate, acceptationDate)
Clef primaire : id
Clef étrangère : id en référence à Personne(Id)

**Course**(passager, départDate, départLat, départLng, arrivéeLat, arrivéeLng, arrivéePrévue, distance, chauffeur)
Clef primaire : passager, departDate
Clefs étrangères :
- passager en référence à Passager(id)
- chauffeur en référence à Chauffeur(id)

*N.B. : le lieu de départ est représenté par le couple (Course.départLat, Course.départLng) et celui d'arrivée par le couple (Course.arrivéeLat, Course.arrivéeLng). La distance est un entier (des kms).*

**Commission**(id, min, max, tarif)
Clef primaire : id
Clefs étrangères : ---

### 1.3. Evolution permettant aux chauffeurs de fixer un tarif dégressif en fonction de la longueur d'un trajet, i.e. d'une course.

**Tarif**(chauffeur, id, min, max, tarifKM)
Clef primaire : chauffeur
Clef étrangère : chauffeur en référence à Chauffeur(id)

*N.B. :*
*- un tarif est relatif à un chauffeur*
*- le couple (min, max) permet d'indiquer la fourchette de distance sur laquelle le chauffeur pratique un certain tarif kilométrique ;*
*- tarifKM est le prix sur la fourchette distance, soit donc un simple décimal typiquement à deux chiffres après la virgule : DECIMAL(3,2) => 1 chiffre avant et 2 chiffres après la virgule.*
*- normalement, l'on devrait empêcher le recoupement de fourchettes de distance, à savoir empêcher le chevauchement de plages (min, max).*

## 2. Fonctions stockées

| | |
|---|---|
| 2.1. | Implémenter la fonction stockée tarif_commission(distance INT) AS DECIMAL qui retourne la commission pratiquée en fonction de la longueur d'un trajet. |
| 2.2. | Implémenter la fonction stockée tarif_chauffeur(numChauffeur INT, distance INT) AS DECIMAL qui retourne le tarif pratiqué par un fournisseur en fonction de la longueur d'un trajet. |
| 2.3. | Implémenter la fonction stockée tarif_chauffeur(numChauffeur INT, distance INT) AS DECIMAL en tenant compte de l'évolution 1.3.<br>*N.B. : on pourra se servir d'un bloc WHILE … DO … END WHILE ;* |
| 2.4. | Quelle condition permet de vérifier si une course est dans le périmètre d'un chauffeur ? Exprimer cette condition de manière formelle. |
| 2.5. | Rédiger la fonction stockée est_dans_perimetre(…) AS BOOLEAN qui permet de tester si une course est dans le périmètre d'un chauffeur.<br>*N.B. : on pourra utiliser la fonction racine carrée SQRT(Nombre DECIMAL).* |

*Exemple d'utilisation de WHILE pour parcourir un jeu d'enregistrements :*

```
DECLARE curseur CURSOR FOR ( SELECT champ1, champ2 FROM table1 … ) ; -- lignes à parcourir
DECLARE i AS INT ; -- compteur pour parcourir les lignes
SELECT count(*) INTO @Nb FROM table1 … ; -- comptage du nombre de lignes à parcourir
i = 0
WHILE i < Nb DO
   FETCH curseur INTO var1, var2 ; -- récupère le contenu de la ligne courante et passé à la suivante
   ….
END WHILE ;
```

### 2.1. Commission pratiquée en fonction de la longueur du trajet

```
FUNCTION tarif_commission(distance INT) AS DECIMAL
BEGIN
   RETURN ( SELECT tarif FROM Commission WHERE distance >= min AND distance < max ) ;
END ;
```

### 2.2. Tarif pratiqué par un chauffeur en fonction de la longueur du trajet

```
FUNCTION tarif_chauffeur(numChauffeur INT, distance INT) AS DECIMAL
BEGIN
   SELECT tarifFixe  , tarifVariable INTO @tf, @tv
   FROM Chauffeur
   WHERE id = numChauffeur ;
   RETURN @tf + @tv * distance ;
END ;
```

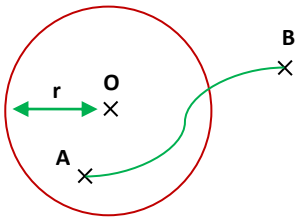
### 2.3. Tarif pratiqué par un chauffeur en tenant compte de l'évolution 1.3

```
FUNCTION tarif_chauffeur(numChauffeur INT, distance INT) AS DECIMAL
BEGIN
    -- récupération de la partie fixe du tarif pratiqué par le chauffeur
    SELECT tarifFixe INTO @tf
    FROM Chauffeur
    WHERE id = numChauffeur ;
     -- récupération des tarifs variables d'un chauffeur (on ordonne bien les tarifs !)
    DECLARE tvs AS CURSOR FOR (
        SELECT min, max, tarifKM
        FROM Tarif
        WHERE chauffeur = numChauffeur
        AND min < distance -- les autres tarifs sont inutiles
        ORDER BY min ASC
    )
    -- nombre de lignes à parcourir
    SELECT count(*) INTO @nb
    FROM Tarif
    WHERE chauffeur = numChauffeur
    AND min < distance
    -- calcul de la partie variable du tarif pratiqué par le chauffeur
    DECLARE i INT
    DECLARE tv DECIMAL
    i = 0
    tv = 0
    -- condition d'arrêt :
    open tvs ;
    WHILE i < @nb DO
        FETCH tvs INTO @min, @max, @tarifKM
        IF distance > @max THEN
            tv = tv + ( @max - @min ) * @tarifKM
        ELSE
            tv = tv + ( distance - @min ) * @tarifKM
        END IF ;
    END WHILE ;
    close tvs ;
    RETURN @tf + tv ;
END ;
```

### 2.4. Tarif pratiqué par un chauffeur en tenant compte de l'évolution 1.3

Une course est dans le périmètre du chauffeur si le lieu de départ et le lieu d'arrivée de la course sont dans le périmètre du chauffeur. Le périmètre du chauffeur est supposé être un cercle de centre son lieu principal. Il faut donc que le lieu de départ et lieu d'arrivée appartiennent à ce cercle.
Autrement dit, il faut que la distance entre le lieu de départ (resp. d'arrivée) de la course et le lieu principal du chauffeur soit inférieure ou égale au rayon du cercle (Chauffeur.périmètre).

Traduction mathématique de cette condition :




Soient trois points $A(x_a; y_a)$, $B(x_b; y_b)$ et $O(x_o; y_o)$. Un passager souhaite effectuer le trajet $AB$. Un chauffeur est localisé en $O$ et intervient dans un périmètre de $r$ kilomètres représenté par un cercle $C$.

$A$ et $B$ appartiennent à $C$ si et seulement si :

- $OA = \sqrt{(x_o - x_a)^2 + (y_o - y_a)^2} \leq r$
- $OB = \sqrt{(x_o - x_b)^2 + (y_o - y_b)^2} \leq r$

Traduction en SQL de cette condition :

Au regard du précédent schéma relationnel, on a :

- $A(Course.départLat, Course.départLng)$
- $B(Course.arrivéeLat, Course.arrivéeLng)$
- $O(Chauffeur.lat, Chauffeur.lng)$

Finalement, $A$ et $B$ appartiennent à $C$ si et seulement si :

- SQRT( ( Course.départLat - Chauffeur.lat ) ^ 2 + ( Course.départLng - Chauffeur.lng ) ^ 2 ) <= Chauffeur.périmètre
- SQRT( ( Course.arrivéeLat - Chauffeur.lat ) ^ 2 + ( Course.arrivéeLng - Chauffeur.lng ) ^ 2 ) <= Chauffeur.périmètre

Ouf ! Ça… c'est fait !

## 2.5. Tester si une course est dans le périmètre d'un chauffeur

```
-- une course est identifiée par le couple (passager, départ), passé en paramètres (p et d)
-- on doit tester si celle-ci est dans le périmètre du chauffeur passé en paramètre (c)
FUNCTION est_dans_perimetre(p INT, d TIMESTAMP, c INT) AS BOOLEAN
BEGIN
    -- récupération des points de départ et d'arrivée de la course
    SELECT départLat, départLng, arrivéeLat, arrivéeLng INTO @aLat, @aLng, @bLat, @bLng
    FROM course
    WHERE passager = p
    AND départDate = d ;
    -- on vérifie que le chauffeur est dans le périmètre
    SELECT count(*) INTO @ok
    FROM chauffeur
    WHERE SQRT( (@aLat - lat)^2 + (@aLng - lng)^2 ) <= périmètre
    AND SQRT( (@bLat - lat)^2 + (@bLng - lng)^2 ) <= périmètre ;
    -- résultat
    IF @ok > 0 THEN
        RETURN true ;
    ELSE
        RETURN false ;
    END IF ;
END ;
```

## 3. Procédures stockées

| | |
|---|---|
| 3.1. | Implémenter la procédure stockée valider_inscription(numChauffeur INT) permettant de valider l'inscription d'un chauffeur. |
| 3.2. | Implémenter la procédure stockée chauffeurs_proximite(…) permettant de lister les chauffeurs disponibles en fonction d'une course, c'est-à-dire tous les chauffeurs dans le périmètre de la course. |
| 3.3. | Implémenter la procédure stockée tarif_moyen(distance INT) qui affiche le tarif moyen (et l'écart type) pratiqué sur une distance donnée.<br>*N.B. : on pourra se servir de l'agrégat STD(…) pour le calcul de l'écart-type, auquel cas on précisera comment l'on pourrait faire sans. STD pour Standard Deviation.* |

### 3.1. Valider l'inscription d'un chauffeur

```
PROCEDURE valider_inscription(numChauffeur INT)
BEGIN
   -- on ne valide une inscription que s'il y a une inscription correspondant au n° du futur chauffeur
   -- passé en paramètre.
   SELECT count(*) INTO @nb
   FROM Inscription
   WHERE id = numChauffeur ;
   -- pas d'inscription, pas de validation d'inscription
   IF @nb = 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'N° d\'inscription inexistant !' ;
   END IF ;
   -- on ne valide une inscription que si l'inscription n'a pas encore été validée
   SELECT count(*) INTO @nb
   FROM Inscription
   WHERE id = numChauffeur ;
   -- si le chauffeur est déjà inscrit, rien à valider
   IF @nb > 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Inscription déjà validée !' ;
   END IF ;
   -- on procède à l'inscription en deux temps
   -- 1er temps : mise à jour du discriminant de la personne correspondante
   UPDATE Personne
   SET type = type & 'I'
   WHERE id = numChauffeur ;
   -- 2ème temps : enregistrement du chauffeur (valeurs par défaut)
   INSERT INTO Chauffeur(id, lat, lng, périmètre, tarifFixe, tarifKM)
   VALUES (numChauffeur, -1, -1, 0, 0, 0) ;
END ;
```

### 3.2. Affichage des chauffeurs de proximité

```
-- on passe en paramètre la course dont on souhaite connaître les chauffeurs pouvant la prendre en
-- charge.
PROCEDURE chauffeurs_proximite(p INT, d TIMESTAMP)
BEGIN
    SELECT CH.*
    FROM Chauffeur CH
    INNER JOIN Course CO
    WHERE CO.passager = p
    AND CO.départDate = d
    AND est_dans_perimetre(CO.passager, CO.départDate, CH.id) ;
END ;
```

### 3.3. Tarif moyen pratiqué sur une distance donnée

```
PROCEDURE tarif_moyen(distance INT)
BEGIN
    SELECT
        AVG(tarifFixe + distance * tarifKM) As 'Moyenne',
        STD(tarifFixe + distance * tarifKM) As 'Ecart-type'
    FROM Chauffeur CH ;
END ;
```