TD – BASE DE DONNÉES

1. AUTOUR DE LA DYNAMIQUE GRAVITATIONNELLE

L'objet est l'étude de solutions algorithmiques en vue de simuler une dynamique gravitationnelle afin, par exemple, de prédire une éclipse ou le passage d'une comète.

À partir de mesures régulièrement effectuées par différents observatoires, une base de données des caractéristiques et des états des corps célestes de notre Système solaire est maintenue à jour. L'objectif de cette partie est d'extraire de cette base de données les informations nécessaires à la mise en oeuvre des fonctions développées dans les parties précédentes, puis de les utiliser pour prévoir les positions futures des différentes planètes. Les données à extraire sont les masses des corps étudiés et leurs états (position et vitesse) à l'instant t_{min} du début de la simulation.

Une version simplifiée, réduite à deux tables, de la base de données du Système solaire est donnée figure 3. Les masses sont exprimées en kilogrammes, les distances en unités astronomiques ($1\ au = 1{,}5 \times 1011m$) et les vitesses en kilomètres par seconde. Le référentiel utilisé pour exprimer les composantes des positions et des vitesses est galiléen, orthonormé et son centre est situé à proximité du Soleil.

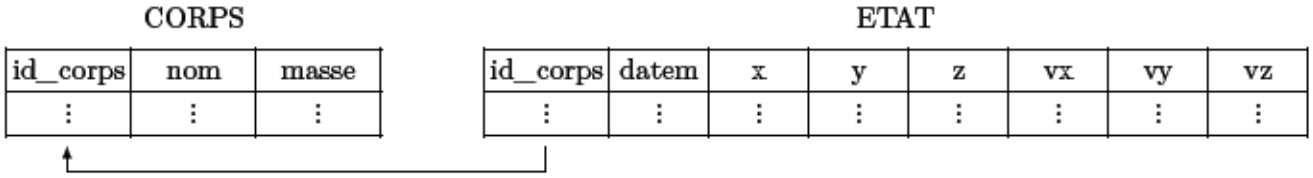


Figure 3 Schéma de la base de données

La table CORPS répertorie les corps étudiés, elle contient les colonnes

- *id_corps (clé primaire) entier identifiant chaque corps ;*
- *nom, chaine de caractères, désigne le nom usuel du corps ;*
- *masse de type flottant, contient la masse du corps.*

La table ETAT rassemble l'historique des états successifs (positions et vitesses) des corps étudiés. Elle est constituée de huit colonnes :

- *id_corps de type entier, identifie le corps concerné ;*
- *datem est la date de la mesure, sous forme d'un entier donnant le nombre de secondes écoulées depuis un instant d'origine ;*
- *trois colonnes de type flottant pour les composantes de la position x, y, z ;*
- *trois colonnes de type flottant pour les composantes de la vitesse vx, vy, vz.*

Q1.Écrire une requête SQL qui renvoie la liste des masses de tous les corps étudiés.

Les états des différents corps ne sont pas forcément tous déterminés exactement au même instant.

Nous allons assimiler l'état initial (à la date t_{min}) de chaque corps à son dernier état connu antérieur à t_{min}.

Dans toute la suite, on supposera que la valeur de t_{min}, sous le format utilisé dans la table ETAT, est accessible à toute requête SQL via l'expression $t_{min}()$.

Comptage des corps

On souhaite d'abord vérifier que tous les corps étudiés disposent d'un état connu antérieur à tmin().

Le nombre de corps présents dans la base est obtenu grâce à la requête SELECT count(*) FROM corps.

Q2.Écrire une requête SQL qui renvoie le nombre de corps qui ont au moins un état connu antérieur à tmin().

Récupération du dernier état des corps

Q3. Écrire une requête SQL qui renvoie, pour chaque corps, son identifiant et la date de son dernier état antérieur à tmin().

Simplification du problème

Le résultat de la requête précédente est stocké dans une nouvelle table date_mesure à deux colonnes :

- *id_corps de type entier, contient l'identifiant du corps considéré ;*
- *date_der de type entier, correspond à la date du dernier état connu du corps considéré, antérieur à tmin().*

Pour simplifier la simulation, on décide de négliger l'influence des corps ayant une masse strictement inférieure à une valeur fixée masse_min() et de ne s'intéresser qu'aux corps situés dans un cube, centré sur l'origine du référentiel de référence et d'arête arete() donnée. Les faces de ce cube sont parallèles aux plans formés par les axes du référentiel de référence.

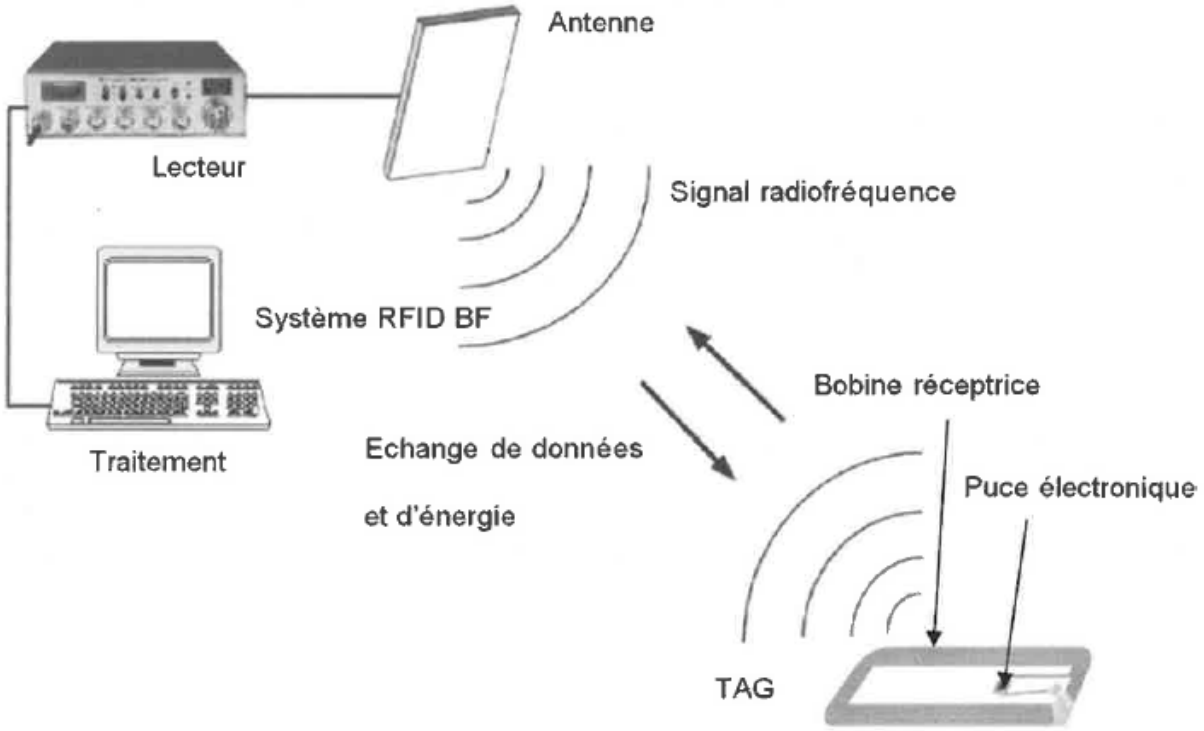
Q4. Écrire une requête SQL qui renvoie la masse et l'état initial (sous la forme masse, x, y, z, vx, vy, vz) de chaque corps retenu pour participer à la simulation. Classez les corps dans l'ordre croissant par rapport à leur distance à l'origine du référentiel.

2. ETUDE DU PASSE NAVIGO

Ce problème traite de systèmes d'identification par radio fréquence (RFID) : l'étude d'un système de type navigo .

Un système RFID passif est composé de deux entités qui communiquent entre elles (Fig.1) :

- *Un TAG passif (dénommé TAG par la suite) ou étiquette intelligente (aussi appelé transpondeur), associé à l'élément à identifier. Il est capable de répondre à une demande venant d'un lecteur. Le TAG n'a pas d'alimentation de type batterie ou pile mais est autoalimenté par l'onde électromagnétique reçue.*
- *Une station de base ou lecteur RFID qui a pour mission d'identifier le TAG. Le lecteur envoie une onde électromagnétique en direction de l'élément à identifier, cette onde alimente le TAG qui peut alors communiquer avec le lecteur grâce à sa puce électronique interne. En retour, le lecteur reçoit l'information renvoyée par le TAG.*



A chaque fois qu'un point de contrôle est franchi, le système collecte des données relatives au passage : lieu, date, heure...

Ces données sont ensuite enregistrées dans la base de données du transporteur, qui comporte trois tables :

La table PASSAGES dédiée aux passages des voyageurs, constituée des champs :

- ***id*** *qui est l'identifiant du passage (clé primaire),*
- ***date*** *qui contient la date du passage au format aaaa-mm-jj,*
- ***heure*** *qui contient l'heure du passage au format hh:mm:ss,*
- ***id_point*** *qui est l'identifiant (entier) du point de passage,*
- ***id_titre*** *qui est l'identifiant (entier) du titre de transport utilisé ;*

la table POINTS dédiée aux points de passage, constituée des champs (entiers) :

- ***id*** *qui est l'identifiant du point de passage (clé primaire),*
- ***zone*** *qui est le numéro de la zone où se trouve le point de passage,*
- ***ligne*** *qui est le numéro de la ligne sur laquelle se trouve le point de passage ;*

la table TITRES dédiée aux titres de transport, constituée des champs (entiers)

- ***id*** *qui est l'identifiant du titre de transport (clé primaire),*
- ***zone_min*** *qui est la plus petite zone couverte par le titre,*
- ***zone_max*** *qui est la plus grande zone couverte par le titre.*

Ces informations sont utilisées par le transporteur pour effectuer des études statistiques sur la fréquentation de ses lignes, en vue d'améliorer le service.

Par exemple, certaines lignes desservant majoritairement des zones d'activités ou des établissements d'enseignement connaissent une forte baisse de leur fréquentation en été, ce qui permet d'alléger le service ; pour choisir la période concernée, il faut connaître précisément l'évolution de la fréquentation au cours de l'été, ainsi que sa répartition au cours de la journée.

Q5. Donner la requête SQL permettant de récupérer les dates et les heures de tous les passages ayant eu lieu sur la ligne numérotée 1 entre le 1er juillet et le 31 août 2014 (inclus).

Un autre exemple de problématique est celui de l'efficacité des dézonages : à certaines périodes de l'année, les voyageurs sont autorisés à emprunter l'ensemble du réseau de transport quelles que soient les zones de validité de leur abonnement. Pour évaluer l'efficacité d'une telle mesure, il faut connaître le nombre de trajets en ayant bénéficié.

Q6. Donner la requête SQL permettant de compter le nombre de passages dézonés, c'est-à-dire ayant eu lieu hors de l'intervalle de validité du titre utilisé, effectués le 31 décembre 2014.

3. RUGBY MANAGER

On possède une base de données ***rugby.db3*** avec 2 tables: JOUEURS et CLUBS.

On souhaite modéliser les transferts des joueurs de club en club comme les transferts réels. En effet, l'application donne un certain budget à son utilisateur et en fonction de ses victoires, il pourra acheter des joueurs de plus en plus chers.

Nous allons donc étudier comment les tables de joueurs et leurs caractéristiques sont associées aux clubs et au salaire des joueurs, Ces attributs représentent leur valeur financière.

Joueurs							
Nom	Poste	age	poids	taille	VMA	Identifiant_Club	Salaire/an
Morellin Thibault	Pilier Gauche	18	105	187	13,6	102	16000
Nkengiang Jules	Pilier Droit	27	115	176	12,9	10	35000
Yeleze Olivier	Troisième ligne Aile	17	90	182	14,5	45	12000
Ramos Thomas	Arrière	20	80	178	18,5	31	40000

CLUBS		
Id_Club	Nom	Budget
1	Stade Français	50 M€
2	Ulster	165 M€
31	Stade Toulousain	150 M€
45	Racing Club de Toulon	540 M€

Q7. Donner la requête qui permet de trouver les joueurs de plus de 23 ans qui ont une Vitesse Maximale Aérobie supérieure à 13km/h.

Q8. Donner la requête qui permet de connaître les clubs dont les joueurs ont un salaire supérieur à 30000€/ans.

Q9. Donner la requête qui permet de vérifier que le Stade Toulousain a bien au moins 3 talonneurs.

4. PREVENTION DES RISQUES AERIENS

Ce problème s'intéresse à différents aspects relatifs à la sécurité aérienne et plus précisément au risque de collision entre deux appareils. Dans la première partie nous abordons l'enregistrement des plans de vol des différentes compagnies aériennes. La deuxième partie explique la manière d'attribuer à chaque vol un couloir aérien répondant au mieux aux souhaits des compagnies aériennes. Enfin, la troisième partie s'intéresse à la procédure de détection d'un risque de collision entre deux avions se croisant à des altitudes proches.

Une liste d'opérations et de fonctions qui peuvent être utilisées dans les fonctions Python figure en fin d'énoncé. Les candidats peuvent coder toute fonction complémentaire qui leur semble utile. Ils devront dans ce cas préciser le rôle de cette fonction, la signification de ses paramètres et la nature de la valeur renvoyée.

Afin d'éviter les collisions entre avions, les altitudes de vol en croisière sont normalisées. Dans la majorité des pays, les avions volent à une altitude multiple de 1000 pieds (un pied vaut 30,48 cm) au-dessus de la surface isobare à 1013,25 hPa. L'espace aérien est ainsi découpé en tranches horizontales appelées niveaux de vol et désignées par les lettres « FL » (*flight level*) suivies de l'altitude en centaines de pieds : « FL310 » désigne une altitude de croisière de 31000 pieds au-dessus de la surface isobare de référence.

EUROCONTROL est l'organisation européenne chargée de la navigation aérienne, elle gère plusieurs dizaines de milliers de vol par jour. Toute compagnie qui souhaite faire traverser le ciel européen à un de ses avions doit soumettre à cet organisme un plan de vol comprenant un certain nombre d'informations : trajet, heure de départ, niveau de vol souhaité, etc. Muni de ces informations, EUROCONTROL peut prévoir les secteurs aériens qui vont être surchargés et prendre des mesures en conséquence pour les désengorger : retard au décollage, modification de la route à suivre, etc.

Nous modélisons (de manière très simplifiée) les plans de vol gérés par EUROCONTROL sous la forme d'une base de données comportant deux tables :

— la table `vol` qui répertorie les plans de vol déposés par les compagnies aériennes ; elle contient les colonnes
- `id_vol` : numéro du vol (chaine de caractères) ;
- `depart` : code de l'aéroport de départ (chaine de caractères) ;
- `arrivee` : code de l'aéroport d'arrivée (chaine de caractères) ;
- `jour` : jour du vol (de type `date`, affiché au format `aaaa-mm-jj`) ;
- `heure` : heure de décollage souhaitée (de type `time`, affiché au format `hh:mi`) ;
- `niveau` : niveau de vol souhaité (entier).

id_vol	depart	arrivee	jour	heure	niveau
AF1204	CDG	FCO	2016-05-02	07:35	300
AF1205	FCO	CDG	2016-05-02	10:25	300
AF1504	CDG	FCO	2016-05-02	10:05	310
AF1505	FCO	CDG	2016-05-02	13:00	310

Figure 1 Extrait de la table `vol` : vols de la compagnie Air France entre les aéroports Charles-de-Gaule (Paris) et Léonard-de-Vinci à Fiumicino (Rome)

— la table `aeroport` qui répertorie les aéroports européens ; elle contient les colonnes
- `id_aero` : code de l'aéroport (chaine de caractères) ;
- `ville` : principale ville desservie (chaine de caractères) ;
- `pays` : pays dans lequel se situe l'aéroport (chaine de caractères).

id_aero	ville	pays
CDG	Paris	France
ORY	Paris	France
MRS	Marseille	France
FCO	Rome	Italie

Figure 2 Extrait de la table `aeroport`

Les types SQL `date` et `time` permettent de mémoriser respectivement un jour du calendrier grégorien et une heure du jour. Deux valeurs de type `date` ou de type `time` peuvent être comparées avec les opérateurs habituels (`=`, `<`, `<=`, etc.). La comparaison s'effectue suivant l'ordre chronologique. Ces valeurs peuvent également être comparées à une chaine de caractères correspondant à leur représentation externe (`'aaaa-mm-jj'` ou `'hh:mi'`).

I.A – Écrire une requête SQL qui fournit le nombre de vols qui doivent décoller dans la journée du 2 mai 2016 avant midi.

I.B – Écrire une requête SQL qui fournit la liste des numéros de vols au départ d'un aéroport desservant Paris le 2 mai 2016.

I.C – Que fait la requête suivante ?

```
SELECT id_vol
FROM vol
     JOIN aeroport AS d ON d.id_aero = depart
     JOIN aeroport AS a ON a.id_aero = arrivee
WHERE
     d.pays = 'France' AND
     a.pays = 'France' AND
     jour = '2016-05-02'
```

I.D – Certains vols peuvent engendrer des conflits potentiels : c'est par exemple le cas lorsque deux avions suivent un même trajet, en sens inverse, le même jour et à un même niveau. Écrire une requête SQL qui fournit la liste des couples $(\mathrm{Id}_1, \mathrm{Id}_2)$ des identifiants des vols dans cette situation.

5. GESTION D'UNE ENTREPRISE

Une entreprise possède 5 sites de production. Le Directeur Général veut améliorer la sécurité. Il souhaite attribuer à chaque employé une carte personnelle et infalsifiable lui permettant l'accès à son site de production et peut-être à d'autres sites de l'entreprise. Chaque employé est affecté uniquement à un site que l'on appellera son site d'origine.
Chaque site de production ne pourra compter plus de 100 employés. La carte attribuée à l'employé comportera sa photo d'identité ainsi qu'une puce. Le code représentant le nom de la personne sera intégré dans la photo mais pas dans la puce.
Un tel code a été placé dans la photo de droite de la **figure 1**. Comme on peut le remarquer, il y a très peu de différences entre les deux photos, l'une sans code, l'autre avec un code intégré.

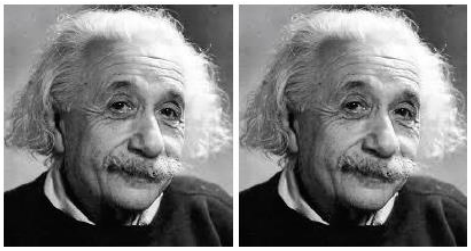


Figure 1 – Photos d'Einstein

Nous allons nous intéresser à la gestion des données qui sont stockées dans la base de données nommée *Personnel* et sera constituée de deux tables intitulées *Employes* et *ListeCategories*. Les contenus de ces tables se trouvent en **Annexe**, page 12.

La table *Employes* est constituée de 8 champs :

- *id* : de type INTEGER ;
- *nom* : de type TEXT ;
- *prenom* : de type TEXT ;
- *email* : de type TEXT – clé primaire (l'adresse email est définie sans son extension @cpp.com dans la table) ;
- *age* : de type INTEGER ;
- *code* : de type TEXT (voir descriptif plus loin) ;
- *site* : de type INTEGER – liste des sites où l'employé est autorisé à entrer en plus de son site d'origine (voir descriptif plus loin) ;
- *code_categorie* : de type INTEGER – indice où la catégorie est référencée dans la table nommée *ListeCategories* (clé étrangère).

La table *ListeCategories* est constituée de 2 champs :

- *id* : de type INTEGER – clé primaire ;
- *categorie* : de type TEXT – liste des métiers de l'entreprise.

Des lecteurs de codes sont installés aux portes des différents sites afin de limiter les accès aux seules personnes habilitées à y entrer.
Rappelons que le code est défini sous la forme d'une chaîne de 8 caractères, le premier caractère indiquant le numéro du site auquel appartient une personne. Par défaut, chaque personne est attribuée à un et un seul site, son site d'origine. Par contre, il est possible à toute personne de cette entreprise de se déplacer dans d'autres sites si l'autorisation en a été donnée (champ *site* de la table *Employes*). Le reste des caractères correspond à la clé de sécurité associée à chaque employé.
La gestion du champ *site* est particulière. On définit un entier qui permet de connaître les sites où l'employé est autorisé à entrer. On attribue les valeurs suivantes :

- 1 : pour le site numéroté 1 ;
- 2 : pour le site numéroté 2 ;
- 4 : pour le site numéroté 3 ;
- 8 : pour le site numéroté 4 ;
- 16 : pour le site numéroté 5.

Ainsi, si un employé appartenant au site 1 est autorisé à entrer dans les sites 2 et 3, la valeur du champ *site* aura comme valeur $2+4$, soit *6*. Pour un autre employé appartenant au site 4 autorisé à entrer dans les sites 1, 2 et 5, la valeur du champ *site* sera $1+2+16$, soit *19*.
Attention : comme on peut le remarquer sur les deux exemples précédents, le numéro du site d'origine (le site où l'employé travaille par défaut) n'est jamais pris en compte dans le calcul donnant la valeur du champ *site*.

L'équipe de direction souhaite avoir certaines informations au sujet des employés de l'entreprise.

Q1. Écrire en SQL la requête 1 suivante donnée en algèbre relationnel :

$$\pi_{nom,prenom}(\sigma_{age>50}(Employes))$$

Q2. Écrire en SQL la requête 2 donnant comme résultat l'adresse email (sans le nom de domaine) des employés pouvant accéder uniquement aux sites 3 et 4 en plus de leur site d'origine.

Q3. Écrire en SQL la requête 3 donnant comme résultat le nom et la catégorie des personnes de l'entreprise ayant au moins 20 ans. Les noms seront classés par ordre alphabétique.

Q4. Écrire en SQL la requête 4 donnant comme résultat la liste des âges des employés avec comme information associée le nombre d'employés ayant le même âge. On demande que cette liste soit décrémentale par rapport au nombre de personnes ayant le même âge.

Par exemple, dans la table Employes, il y a 2 personnes qui ont 22 ans.

CORRIGE

Q1 :
SELECT masse FROM corps

Q2 :
SELECT count (DISTINCT id_corp) FROM etat WHERE datem < tmin ( )

Q3 :
SELECT id_corps , MAX(datem) FROM etat WHERE datem < tmin ( ) GROUP BY id_corps

Q4 :
```
SELECT a.masse , b.x , b.y , b.z , b.vx , b.vy , b.vz FROM date_mesure AS c
JOIN etat AS b ON c . id_corps=b . id_corps AND c . date_der=b . datem
JOIN corps AS a ON c . id_corps=a . id_corps
WHERE ABS(b.x ) < arete( ) /2 AND ABS(b.y ) < arete( ) /2
AND ABS(b.z ) < arete( ) /2 AND a.masse > masse_min( )
ORDER BY SQRT(POWER(b.x ,2)+POWER(b.y ,2)+POWER(b.z , 2 ) ) ASC
```

II. RIFD

Q5

SELECT p.date,p.heure FROM passages as p JOIN points as pt ON p.id_point = pt.id WHERE pt.ligne=1 and (p.date >= "2014-07-01" AND p.date <= "2014-08-31")

Q6

SELECT COUNT p.id FROM passage, points, titres WHERE passages.id_point = points.id AND passages.id-titre = titres.id AND passages.date='2014-12-31' AND (points.zone < titres.zone_max OR points.zone > titres.zone_max)

III. RUGBY

#Q7
requete1="SELECT Nom FROM JOUEURS WHERE age > 23 AND VMA > 13"

#Q8 Club des salaires >3000
requete2="SELECT C.Nom FROM CLUBS as C JOIN JOUEURS as J ON
J.Identifiant_Club=C.Id_Club WHERE J.Salaire > 30000"

#Q8 Talonneur
requete3="SELECT Count(*) FROM JOUEURS as J JOIN CLUBS as C ON
J.Identifiant_Club=C.Id_Club WHERE C.Nom='Stade Toulousain' AND J.Poste='Talonneur')"

```
#en python
import os
os.chdir('C:/… repertoire de travail')
import sqlite3 # Import des commandes permettant de manipuler la base de données
basesql = u"rugby.s3db" # Base de données initiale
cnx = sqlite3.connect(basesql )
```

```
curseur = cnx.cursor ()
requete4="SELECT * FROM EQUIPE"
curseur.execute(requete4)
monequipe = curseur.fetchall()
print (monequipe)
print(type(monequipe))
```

IV. RISQUES AERIENS

A. La comparaison pour les jours et les heures fonctionne si les types sont **chaînes de caractères** mais pas nécessairement avec le format **date** et **time** spécifié.

```
SELECT COUNT(id_vol) FROM vol WHERE jour = "2016-05-02" AND heure < "12:00"
```

B. Requête donnant la liste des numéros de vols au départ d'un aéroport desservant Paris le 02mai 2016 :

```
SELECT id_vol
FROM vol JOIN aeroport
ON arrivee = id_aero
WHERE ville = "Paris" AND jour = "2016-05-02"
```

C. Que fait cette requête ?

```
SELECT id_vol
FROM vol
JOIN aeroport AS d ON d . id_aero = depart
JOIN aeroport AS a ON a . id_aero = arrivee
WHERE d . pays = "France" AND a . pays = "France" AND jour = "2016-05-02"
```

Elle renvoie les numéros des vols intérieurs à la France pour le jour du 2mai 2016.

D. Requête permettant d'obtenir les couples de numéros de vols susceptibles d'engendrer des conflits :

```
SELECT vol1 . id_vol AS Id1 , vol2 . id_vol AS Id2
FROM vol AS vol1 , vol AS vol2
WHERE
Id1<Id2 AND vol1 . niveau = vol2 . niveau
AND vol1 . jour = vol2 . jour
AND vol1 . depart = vol2 . arrivee
AND vol1 . arrivee = vol2 . depart
```

La suppression des doublons de la liste est obtenue par le test Id1<Id2 ;

5. GESTION ENTREPRISE

```
SELECT nom, prenom, age FROM Employes WHERE age >50;

SELECT email FROM Employes WHERE site=12

SELECT * FROM ‘Employes‘ JOIN ‘ListeCategories‘ ON ‘Employes‘.‘code_categorie‘ = ‘ListeCategories‘.‘id‘
WHERE ‘Employes‘.age >20 ORDER BY ‘Employes‘.‘nom‘ ASC ;

SELECT age , COUNT(*) as nb FROM Employes GROUP BY age ORDER BY nb DESC ;
```