

Examen

QCM du MOOC/COURS “Des Bases de données à Big Data”

(*BIG DATA MANAGEMENT*)

Global Q&A

Pr Serge Miranda

Mai 2020

Chaque question vaut 1 point sauf celles ayant une indication spécifique, -0.5 par mauvaise réponse dans choix multiple explicitement indiqué.

Question01

1. Les 3 V de base de définition du Big Data par M.Stonebraker correspondent à:

- ☒ Volume, Variété, Vélocité
- ☐ Volume, Valeur et Véracité
- ☐ Volume, Versatilité et Value ;

- **Réponse:**

- Volume, Variété, Vélocité

Explication: Les 3 V de base du Big Data sont traditionnellement Volume, Variété et Vélocité. Le "Volume" se réfère à la grande quantité de données, la "Variété" parle de la diversité des types de données et la "Vélocité" concerne la vitesse à laquelle les nouvelles données sont générées et recueillies.

Question02

1. Le quatrième paradigme des sciences de Jim Gray concerne:

- ☒ La science des données
- ☐ La science des services
- ☐ La science de la programmation

- **Réponse:**

- La science des données

Explication: Jim Gray a décrit le quatrième paradigme de la science comme étant la science des données. Après l'empirisme, la modélisation théorique et la simulation computationnelle, le quatrième paradigme se concentre sur l'obtention de connaissances à partir de l'énorme quantité de données.

Question03

1. Une base de données STRUCTUREES correspond à (2 choix):

- ☐ Un schéma variable prédéfini
- ☒ Un schéma fixe prédéfini
- ☒ Un Schéma que l'on peut définir en SQL2 ou SQL3
- ☐ Une absence de Schéma et de meta-data

- **Réponse:**

- Un schéma fixe prédéfini
- Un Schéma que l'on peut définir en SQL2 ou SQL3

Explication: Une base de données structurée a généralement un schéma fixe qui est défini à l'avance. Les bases de données structurées sont souvent gérées à l'aide de langages SQL tels que SQL2 ou SQL3

Question04

1. Une base de données SEMI-Structurées correspond (2 choix):

- ☐ a l'existence des meta data
- ☒ absence d'interface de type SQL et de schéma prédéfini
- ☒ des TRIPLE DATA STORES au format RDF

- **Réponse:**

- des TRIPLE DATA STORES au format RDF
- absence d'interface de type SQL et de schéma prédéfini

Explication: Les bases de données semi-structurées, bien qu'elles ne suivent pas un schéma strict comme les bases de données structurées, contiennent généralement des métadonnées qui décrivent certaines structures des données. Cependant, elles ne possèdent généralement pas d'interface de type SQL et de schéma strictement prédéfini.

Question05

1. Une base de données NON-Structurée correspond à (2 choix):

- ☒ une approche N.O.SQL
- ☒ une approche non relationnelle
- ☐ l'existence de metadata de type balise XML
- ☐ l'existence d'un schéma prédéfini

• **Réponse:**

- une approche N.O.SQL
- une approche non relationnelle

Explication: Les bases de données non structurées ne sont pas organisées de manière traditionnelle et n'ont généralement pas de schéma prédéfini. Elles adoptent souvent une approche non relationnelle, et c'est pourquoi elles sont souvent associées à des bases de données N.O.SQL.

Question06

1. Les Propriétés TIPS d'un SGBD relationnel font référence à:

- ☐ Transaction, Innovation, Produit Cartésien et Schéma
- ☒ Transaction, Interface, Persistance et Structuration
- ☐ Transaction, Innovation, Pagination et SQL

• **Réponse:**

- Transaction, Interface, Persistance et Structuration

Explication: Bien que "TIPS" ne soit pas un acronyme couramment associé aux SGBD relationnels, la réponse la plus logique ici serait Transaction, Interface, Persistance et Structuration, car ils sont tous des concepts pertinents pour les bases de données relationnelles.

Question07 *

1. Les propriétés **RICE** de l'approche OBJET (2 choix):





- ☒ s'appliquent à la définition d'une classe d'objets qui est un cas **particulier** de type de données

- ☐ signifient Réutilisabilité, Identification, Cohérence et Encapsulation
- ☒ signifient Réutilisabilité, Identification, Complexité et Encapsulation
- ☐ servent à la définition de relations normalisées au sens de Codd

• **Réponse:**

- s'appliquent à la définition d'une classe d'objets qui est un cas particulier de type de données
- signifient Réutilisabilité, Identification, Complexité et Encapsulation

Propriétés **RICE** d'un SGBD OBJET

◦		Réutilisabilité (Héritage ou polymorphisme) Graphe héritage
		IDENTIFICATION système
		Construction d'objets Complexes Orthogonalité TUPLE et SET Graphes d'agrégation
		Encapsulation (Messages, Méthodes, Classes d'objets)

Explication: Les propriétés RICE de l'approche OBJET s'appliquent à la définition d'une classe d'objets. Ces propriétés sont essentielles pour définir et gérer des objets dans une approche orientée objet. Ces propriétés permettent d'assurer l'intégrité, la cohérence et la sécurité des objets dans le système.

Question08 ?


1. Un OBJET (2 choix):

- ☒ Correspond à un couple (OID, VALEUR) avec OID = Object Identification (pouvant être un pointeur)
- ☐ instance d'une classe (qui est donc un ensemble d'objets)
- ☐ est toujours un tuple (n-uplet) dans une relation
- ☐ est toujours une colonne dans une table SQL

• **Réponse:**

- Correspond à un couple (OID, VALEUR) avec OID = Object Identification (pouvant être un pointeur)
- instance d'une classe (qui est donc un ensemble d'objets)

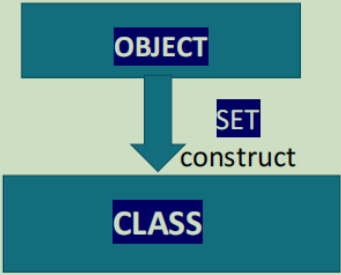
Explication: Un objet, dans le contexte de la programmation orientée objet, est une instance d'une classe. Il est identifiable par son identifiant d'objet ou OID. L'OID est un identifiant unique pour chaque objet, qui peut être représenté par un pointeur. En revanche, dans un contexte de base de données relationnelle, un objet n'est ni un tuple ni une colonne.




« Objets » et « Classe d'objets » ?

DEFINITIONS :

- **OBJET?**
 - **OBJET = (OID, VALUE)**
 - OID : Object Identifier
 - → HASHING, b-TREE, ...
 - *cf (KEY, VALUE) in N.O.SQL
- Une **CLASSE** d '**OBJETS**
 - est un (« valeurs potentielles »)
 - a/possède un (« valeurs réelles »)
- : **TYPE de DONNEES qui vérifie les propriétés RICE !**





Deux possibilités pour le modèle OR

1. RELATION = CLASSE d objets

Approche « simpliste » (UNISQL, NF2, POSTGRES,...)
 EX : CREATE OBJECT CLASS PILOT PUBLIC
 (PL# NUMERIC, PLNOM CHAR, ADR CHAR) ;

- **Avantage : concept commun unique**

CLASSE-RELATION

- **PB1 : objets = tuples (non encapsulés) !**
- **PB2 : opérateurs génériques et spécifiques**
 - Algèbre Complexe ? Opérateur pour parcourir les graphes (hiérarchie structurelle) expression de chemin ?
- **PB3: propriété de fermeture**
 - Réinterprétation des opérateurs relationnels ?

Question09

1. Le concept de **relation** dans le modèle relationnel de Codd (2 choix):
 - ☐ correspond une TABLE dans SQL2 dans laquelle les colonnes peuvent être multivaluées (des ensemble de valeurs)
 - ☒ correspond à un **prédicat** à n **variables** en logique du premier ordre

- ☐ correspond à l'union d'ensembles appelés domaines
- ☒ correspond au **sous-ensemble** du produit **cartésien** de n ensembles appelés domaines
- ☐ correspond à une matrice de valeurs à un instant donné (au sens algèbre linéaire)

• **Réponse:**

- correspond à un prédicat à n variables en logique du premier ordre
- correspond au sous-ensemble du produit cartésien de n ensembles appelés domaines

Explication: Dans le modèle relationnel de Codd, une relation est vue comme un sous-ensemble du produit cartésien de n domaines (ensembles de valeurs possibles pour un attribut). Dans une perspective logique, chaque tuple dans une relation correspond à un prédicat à n variables.

Question10 *

1. Le concept de **domaine** du modèle relationnel de Codd (2 choix):

- ☒ Joue le rôle d'un type de données sémantique (ensemble de valeurs)
- ☒ Est un pré-requis à la création des relations
- ☐ Est uniquement un simplificateur syntaxique comme en SQL
- ☐ Ne peut être utilisé qu'une fois dans la définition d'une même relation

• **Réponse:**

- Joue le rôle d'un type de données sémantique (ensemble de valeurs)
- Est un pré-requis à la création des relations

Explication: Un domaine dans le modèle relationnel de Codd est similaire à un type de données. Il spécifie un ensemble de valeurs possibles pour un attribut. Le domaine ne sert pas simplement de simplificateur syntaxique, et il n'est pas limité à une utilisation unique par relation.

Question11

1. Un domaine **primaire** dans le modèle relationnel de Codd:

- ☒ permet de traduire automatiquement la clé primaire dans une relation
- ☐ permet de traduire automatiquement la clé étrangère dans une relation

- **Réponse:**

- permet de traduire automatiquement la clé primaire dans une relation

Explication: Dans le modèle relationnel, un domaine primaire fait référence au domaine qui sert de clé primaire pour une relation. Il ne se réfère pas à la clé étrangère.

Question12 *

1. Le concept de **transaction** avec ses propriétés ACID (2 points ; 2 réponses):

- ☐ Ne comprend que des opérations de type SELECT/FROM/WHERE entre les verbes BEGIN et END
- ☐ Evite le problème de DEAD LOCK dans tous les cas
- ☐ Evite le problème de LIVELOCK dans certains cas
- ☒ Permet de résoudre des problèmes de cohérence de la BD en cas d'interférences entre mises à jour **concurrentes**
- ☒ Permet d'assurer la cohérence de la BD en cas de **pannes**
- ☐ Offre des points de retour arrière partiels (SAVE POINTS) pour normaliser la cohérence

- **Réponse:**

- Permet de résoudre des problèmes de cohérence de la BD en cas d'interférences entre mises à jour concurrentes
- Permet d'assurer la cohérence de la BD en cas de pannes

Explication: Les transactions sont définies par leurs propriétés ACID : Atomicité, Cohérence, Isolation et Durabilité. Ces propriétés garantissent que la base de données reste cohérente et fiable, même en cas de pannes ou d'erreurs. Les points de sauvegarde (SAVE POINTS) permettent un retour arrière partiel, offrant plus de flexibilité lors de la gestion des transactions.

➤ Les propriétés TRANSACTIONNELLES « **ACID** » visent à résoudre 2 problèmes importants dans la cohérence d'une base de données : lesquels ?

➤ **ACID** et 2 PB : **PANNE** et **CONCURRENCE**

➤ AC Atomicité :

(TOUT ou RIEN des opérations de MAJ concernant une transaction) et Cohérence de la BD quelle que soit la PANNE qui pourrait se produire

➤ ID Isolation et Durabilité :

Chaque transaction « bien formée » et écrite de manière isolée avec des mécanismes de verrouillage (Verrouillage en lecture/écriture/intention) à deux phases maintiendra durablement la cohérence de la BD quelle que soit la concurrence

Question13

1. Le Théorème de Codd démontre (2 choix):

- ☐ La faisabilité d'implantation de l'algèbre relationnelle (et donc de SQL)
- ☒ La **complétude** du calcul relationnel (langage Alpha)
- ☐ La fermeture du calcul relationnel
- ☐ L'orthogonalité du langage relationnel
- ☒ L'équivalence entre le calcul **relationnel** et l'**algèbre** relationnelle qui devient donc la référence de complétude des langages relationnels

• Réponse:

- La complétude du calcul relationnel (langage Alpha)
- L'équivalence entre le calcul relationnel et l'algèbre relationnelle qui devient donc la référence de complétude des langages relationnels

Explication: Le théorème de Codd montre que le calcul relationnel (comme exprimé par son langage Alpha) est complet en ce sens qu'il peut exprimer toute requête que l'algèbre relationnelle peut exprimer. En outre, le théorème établit l'équivalence entre le calcul relationnel et l'algèbre relationnelle, indiquant que les deux sont tout aussi puissants en termes d'expressivité.

Question14

1. La relation suivante n'est pas en 3NF : **AVION (AV, AVNOM, CAP, LOC)** avec **AV** clé primaire et le lien N:1 suivant **CAP -> LOC**. Quelle est la bonne décomposition en 3 NF de cette relation
- ☐ AVION1 (AV, AVNOM, LOC) et AVION 2 (CAP, LOC)
 - ☐ AVION1 (AV, CAP, LOC) et AVION2 (AV, AVNOM)
 - ☐ AVION1 (AV,CAP) et AVION2 (CAP, AVNOM, LOC)
 - ☒ AVION1 (AV, AVNOM, CAP) et AVION2 (CAP, LOC)

- **Réponse:** 4. AVION1 (AV, AVNOM, CAP) et AVION2 (CAP, LOC)

Explication: La forme normale 3 (3NF) vise à éliminer les dépendances transitives. Dans cette relation, **CAP** détermine **LOC**, ce qui signifie que **LOC** est transitivement dépendant de la clé primaire **AV** via **CAP**. Pour décomposer cette relation en 3NF, nous devons séparer **CAP** et **LOC** de la clé primaire, résultant en deux relations : AVION1 qui contient la clé primaire **AV** et AVION2 qui a **CAP** comme clé primaire avec l'attribut **LOC** associé.

Considérez les relations PIECE (P#, PNOM, COULEUR, POIDS), FOURNISSEUR (F#, FNOM, ADR, TEL) et COMMANDE (C#, P#, F#, DATE, QTE)

Compléter les requêtes suivantes dans l' algèbre de Codd et SQL2

Quels sont les noms et adresses des fournisseurs qui ont fourni des pièces de couleur rouge ?

Algèbre de Codd Soit la partie de la requête suivante :

```
1  PV1 = Join PIECE (P# = P#) COMMANDE
2  PV2 = ??
3  RES1 = SELECT PV2 ( COULEUR = Rouge)
4  RES = PROJECT RES1 (FNOM, ADR)
```

Question15

1. Compléter l'équation PV2 en remplaçant ?? par
- ☐ select PV1 (COULEUR = Rouge)
 - ☐ Join PV1 (PNOM= FNOM) Fournisseur
 - ☒ Join PV1 (F#= F#) Fournisseur

- **Réponse:** 3. Join PV1 (F#= F#) Fournisseur

Explication: Pour trouver les fournisseurs qui ont fourni des pièces de couleur rouge, nous devons relier les informations des fournisseurs à celles des commandes et des pièces. PV1 est le résultat de la jointure de PIECE et COMMANDE sur l'attribut P#. Pour obtenir les noms et adresses des fournisseurs concernés, nous devons ensuite joindre le résultat PV1 avec la table FOURNISSEUR sur l'attribut F#. Donc, la bonne opération pour PV2 est de joindre PV1 avec FOURNISSEUR sur F#.

```
1 SQL2
2 SELECT (FNOM, ADR)
3 From FOURNISSEUR
4 Where F# IN
5     (SELECT F# from COMMANDE, PIECE Where ??);
```

Question16

1. Pour compléter la requête remplacer ?? par

- ☐ P# in (SELECT P# from COMMANDE where Couleur = rouge)
- ☒ COMMANDE.P#= PIECE.P# and Couleur='rouge'
- ☐ COMMANDE.F# = Fournisseur.F# and Couleur = rouge

- **Réponse:** 2. COMMANDE.P#= PIECE.P# and Couleur='rouge'

Explication: Pour identifier les fournisseurs qui ont fourni des pièces de couleur rouge, nous avons besoin de relier les commandes aux pièces basées sur leur P#. L'attribut couleur doit ensuite être utilisé pour filtrer les pièces qui sont rouges. Par conséquent, la condition correcte pour la sous-requête est que **COMMANDE.P#** doit être égal à **PIECE.P#** et que la couleur de la pièce doit être rouge.

Traitez dans l'algèbre de Codd et de 2 manières différentes en SQL (ou SEQUEL) la requête suivante:

Quels sont les noms et adresses des fournisseurs qui ont fourni TOUTES les pièces de couleur rouge ?

```

1  FC = JOIN FOURNISSEUR (F# = F#) COMMANDE
2  DD = ??
3  P1 = SELECT PIECE (Couleur = rouge)
4  DS = Project P1 (P#)
5  RES = DD / DS

```

Question17

1. Pour compléter la requête il faut remplacer ?? par

- ☒ JOIN FC (P# = P#) PIECE
- ☐ PROJECT FC (FNOM, ADR, P#)

• **Réponse:**

1. JOIN FC (P# = P#) PIECE

Explication: Pour identifier les fournisseurs qui ont fourni toutes les pièces de couleur rouge, nous avons besoin de joindre les informations de fournisseurs, commandes et pièces. La table FC est une jointure des fournisseurs avec leurs commandes. Ensuite, pour compléter l'information, nous devons joindre FC avec PIECE sur l'attribut P#.

```

1  SQL2
2  SELECT (FNOM, ADR) From FOURNISSEUR
3  Where
4      NOT EXIST
5      (SELECT * from Piece where Couleur = 'rouge' and
6          NOT EXIST
7          (SELECT * from COMMANDE where ??));

```

Question18

1. Pour compléter la requête il faut remplacer ?? par

- ☐ COMMANDE.P# = FOURNISSEUR.F#
- ☐ COMMANDE.P# = PIECE.P#
- ☒ COMMANDE.P# = PIECE.P# and COMMANDE.F# = FOURNISSEUR.F#

- **Réponse:** 3. `COMMANDE.P# = PIECE.P#` and `COMMANDE.F# = FOURNISSEUR.F#`

Explication: La sous-requête vise à trouver les pièces de couleur rouge qui n'ont pas été fournies par un fournisseur particulier. Pour cela, nous devons associer une commande à une pièce basée sur leur P# et vérifier si cette commande a été passée par le fournisseur concerné.

Question19

1. Le **deuxième** manifeste de **STONEBRAKER** sur le modèle Objet Relationnel (2 choix) :
 - ☒ fait du domaine une **classe** d'objets (propriétés RICE à ce niveau) et **ne touche pas aux relations**
 - ☐ prend en compte l'héritage structurel seulement au niveau des relations et l'encapsulation au niveau des domaines
 - ☒ est compatible avec **SQL3**.
 - ☐ nécessite un opérateur pour traduire l'héritage structurel à partir de l'héritage entre domaines
 - ☐ Fait de la relation une classe d'objets (propriétés RICE à ce niveau) et ignore le concept de domaine

- **Réponse:**

1. fait du domaine une classe d'objets (propriétés RICE à ce niveau) et ne touche pas aux relations
2. est compatible avec SQL3.

Explication: Le modèle objet-relationnel de Stonebraker tente de combiner les avantages du modèle relationnel avec ceux du modèle objet. Le premier choix mentionne que les domaines sont traités comme des classes d'objets avec des propriétés RICE. Le modèle est également compatible avec SQL3, ce qui est indiqué dans le troisième choix.

Question20

1. Le **troisième** manifeste de Chris Date est une extension du modèle relationnel de Codd comprenant (2 choix):
 - ☐ L'apparition des pointeurs bi-directionnels entre domaines primaires

- ☒ **L'héritage** (opérationnel) entre domaines primaires et les propriétés RICE au niveau des domaines
- ☒ un opérateur **d'héritage** structurel (avec des clés primaires en argument d'entrée)
- ☐ L'encapsulation des relations avec des ROWID sur les nuplets
- ☐ est incompatible avec SQL3

• **Réponse:**

- L'héritage (opérationnel) entre domaines primaires et les propriétés RICE au niveau des domaines
- un opérateur d'héritage structurel (avec des clés primaires en argument d'entrée)

Explication: Le troisième manifeste de Chris Date vise à étendre le modèle relationnel de Codd pour y incorporer des fonctionnalités orientées objet tout en préservant les principes fondamentaux du modèle relationnel. L'héritage entre les domaines primaires et les propriétés RICE sont des ajouts clés de ce manifeste. De plus, un opérateur d'héritage structurel est introduit pour faciliter l'héritage entre les relations.

Question21

1. EXEMPLES du Manifeste de Chris Date Considérez les requêtes suivantes dans le modèle objet relationnel de DATE sur l'exemple de schéma vu en cours rappelé ci-dessous :

Where editor = « Dunod » and LOAN = « Truc » and B# → title = « Concepti% » and B# → keyword in { Software, CONCERN (Software)}

- ☐ CORRECTE
- ☒ INCORRECTE

• **Réponse:** 2. INCORRECTE

Explication: La requête semble combiner différents critères de manière ambiguë. Le modèle objet-relationnel de Date se préoccupe des pointeurs et des relations entre objets, mais le format de cette requête semble incorrect. De plus, sans connaître le schéma exact, il est difficile de déterminer si les attributs et les relations utilisés existent.

Soit le schéma « objet relationnel » vu en cours (les types REF sont en gras avec le préfixe «REF ») enrichi par la classe Formation :

```
1  Pilote (PL#,PLNOM, ADR, REFAV, PhotoPL) <REFAV: Avion préféré d'un
   pilote>
2  Avion (AV#,AVNOM, (CAP), LOC, REFAVBIS, REFPIIL)
3      <REFPIIL Pilote attitré d'un avion; REFAVBIS : Référence
   avion de rechange> ; <(CAP)/ : ensembles des capacités d'un
   avion> ; <AVNOM : nom Avion (B727, A300,..)> ;
4  Vol (Vol#, PL#, AV#, REFPIILLOTE, REFAVION, VD, VA, HD, HA)
5  Formation (PL#, AVNOM, REFP, Date) <REFP : Pointeur du 'Pilote'
   ayant reçu la formation>
```

Question22 ?

1. La requête suivante en SQL3

```
1  Select Refpilote → Refav→ Refpil → Plnom
2  From Vol
3  Where Refavion→ Refavbis→ Refpil→ ADR='Nice';
```

Est:

1. INCORRECTE
2. CORRECTE

- **Réponse:**

1. INCORRECTE

Explication: La requête tente de suivre plusieurs pointeurs REF successifs à partir d'un objet initial. Cependant, il semble que certains de ces pointeurs n'existent pas dans le schéma fourni. Par exemple, **Refavion→ Refavbis→ Refpil** semble incorrect car **Refavbis** pointe vers un autre avion, pas directement vers un pilote.

Question23

1. La requête suivante en SQL3

- ```
1 Select Avnom
2 From Vol, Avion
3 Where vol.pl# = Avion.refpil and VD='Nice';
```

Est:

- ☐ INCORRECTE
- ☒ CORRECTE

- Réponse: 2. CORRECTE

**Explication:** Cette requête semble être formulée correctement. Elle joint les tables **Vol** et **Avion** sur les attributs de pointeur **vol.pl#** et **Avion.refpil**, et filtre ensuite sur l'attribut **VD** de la table **Vol**.

## Question24 \*

1. La requête suivante en SQL3

- ```
1 Select Refpilote→ Plnom "
2 From Vol, Formation, Avion
3 Where Vol.refpilote= Formation.Refp and Formation.Avnom=
  'Airbus' ;
```

Est:

- ☒ INCORRECTE
- ☐ CORRECTE

- Réponse : INCORRECTE

1. La requête essaie de relier **Vol**, **Formation**, et **Avion** en utilisant la condition **Vol.refpilote= Formation.Refp**. Cependant, il n'y a pas de condition qui lie la table **Avion** à l'une des autres tables, ce qui pourrait entraîner un produit cartésien. La sélection de **Refpilote→ Plnom** est également incertaine car **Refpilote** est un champ de référence.

- 旧答案

- Réponse: 2. 正确

Explication: La requête rejoint les tables **Vol**, **Formation**, et **Avion** sur les attributs de pointeur appropriés et filtre ensuite les résultats où **Formation.Avnom** est égal à 'Airbus'. La sélection semble correctement formulée selon le schéma fourni.

Question25

1. La requête suivante en SQL3

- ```
1 Select f.date
2 From Formation f
3 Where f.REFP → REFAV → REFAVBIS → REFPIIL → ADR = 'Nice'
```

Est:

- ☒ Correcte
- ☐ Incorrecte

- **Réponse:**

- 1. Correcte

**Explication:** La requête utilise le pointeur **REFP** pour naviguer à travers plusieurs relations: de la table **Formation** à **Pilote**, puis à **Avion**, et enfin à un autre **Pilote**. Elle filtre ensuite sur l'adresse 'Nice'. Selon le schéma fourni, cette navigation semble plausible et correctement formulée.

La table **Formation** a un champ **REFP** qui est une référence au **Pilote**. Cependant, **Pilote** a un champ **REFAV** qui est une référence à **Avion**. **Avion** a un champ **REFAVBIS**, qui est une autre référence à un autre avion (avion de rechange), et cet avion de rechange a un champ **REFPIIL** qui est une référence à **Pilote**. Enfin, **Pilote** a un champ **ADR**.

La chaîne de références semble correcte, bien qu'elle soit complexe.

---

## • La requête suivante en SQL3

---

- ```
1  Create Domain DOCNO primary
2  Function LOAN <boolean value>
3  Create domain BNO UNDER DocNO primary
4  Create Domain HandoutNO UNDER DocNO primary
5  Function COPY
6
7  Create Domain Dtitle Character (12)
8  Create Domain Dauthor Character (12)
9  Create Domain Dpage INT
```



```

10 Create Domain Deditor Character (12)
11
12 Create Domain THESAURUS
13 Function SYNONYMY...
14 Function HIERARCHY...
15 Function CLOSENESS...
16 Function CONCERN,...
17
18 Create Relation Document
19 (DOC# : DOCNO, Primary Key
20 title : Dtitle
21 NBRPAGE : Dpage
22 key word SET-OF: THESAURUS)
23
24 Create Relation BOOK
25 (B# : BNO Primary key
26 author SET-OF: Dauthor
27 editor : Deditor)
28
29 Create Relation HANDOUT
30 (H# : HANDOUTNO Primary Key
31 author : Dauthor)
32

```

Les requêtes suivantes dans le langage de DATE sont-elles correctes

参考P293

Question26

1. Q1 :

```

1 Select B# → NBRPAGE
2 From Book
3 Where NBRPAGES > 500;

```

Est:

- ☐ INCORRECTE
- ☒ CORRECTE

• Réponse:

1. INCORRECTE

Explication: La requête tente de sélectionner `B# → NBRPAGE` de la relation `Book`, mais il semble que `NBRPAGE` ne soit pas un attribut de `Book` mais plutôt de `Document`. De plus, il y a une incohérence entre `NBRPAGE` et `NBRPAGES` :

Question27

1. Q2 :

```
1  Select B#→ NBRPAGE
2  From Document
3  Where editor = « Dunod » and LOAN = « True » and B#→ title = «
   Concept% »
```

Est:

☒ INCORRECTE

☐ CORRECTE

• Réponse:

1. INCORRECTE

Explication: La requête tente d'accéder à `B#→ NBRPAGE` depuis la relation `Document`, ce qui est correct, mais elle fait également référence à l'attribut `LOAN` qui n'est pas défini dans le schéma fourni. De plus, elle tente de comparer `B#→ title` alors que `B#` est une clé pour `BOOK`, pas pour `Document` :

`From Document` doit être `From Book`

Question28

1. Q3:

```
1  Select B#→ NBRPAGE
2  From Book
```

```
1  Select p.PLNOM
2  From Pilote p
3  Where p.REFAVBIS→ REFPILOTE→ ADR = 'Nice'
```

Est:

☒ INCORRECTE

☐ CORRECTE

- Réponse:

1. INCORRECTE Parce que le table **pilote** n'a pas le **REFAVIS** ; 因为pilote没有 REFAVIS

Explication: La requête semble combiner deux requêtes distinctes sans aucune syntaxe de jointure ou de sous-requête. Elle tente d'accéder à **B#→ NBRPAGE** de la relation **Book** et ensuite à sélectionner **p.PLNOM** de la relation **Pilote** , mais il n'y a aucune liaison entre ces deux opérations.

Question29

1. Le Théorème CAP du NO SQL indique que seulement 2 des 3 propriétés suivantes peuvent être satisfaites:

- ☐ Cohérence, Atomicité et Persistance (pagination)
- ☒ Cohérence, Atomicité et Partitionnement
- ☐ Cohérence, Atomicité et Parallélisme.

- Réponse:

- Cohérence, Atomicité et Partitionnement

➤ Théorème **CAP** (Eric Brewer, Prof Berkeley,

◦

- Consistency (Cohérence),
- *AVAILABILITY* (Disponibilité),
- Partitioning (partitionnement)

Explication: Le théorème CAP, proposé par Eric Brewer, stipule qu'il est impossible pour un système de données distribué de fournir simultanément plus de deux des trois garanties suivantes : Cohérence, Disponibilité et Tolérance au partitionnement. La bonne réponse est donc Cohérence, Atomicité et Partitionnement.

Question30

1. Trois systèmes (CLE , Valeurs) pour N.O.SQL (3 choix):

- ☐ Valeur = BLOB
- ☐ Valeur = SET
- ☒ Valeur = Document (JSON)
- ☐ Valeur = Matrice
- ☒ Valeur = Colonnes
- ☒ Valeur = Graphe

- **Réponse:**

- Valeur = Document (JSON)
- Valeur = Colonnes
- Valeur = Graphe

Explication: Dans le contexte des bases de données NoSQL, les modèles couramment utilisés incluent les bases de données orientées document (comme MongoDB) qui stockent des données sous forme de documents JSON, les bases de données orientées colonnes (comme Cassandra) et les bases de données orientées graphe (comme Neo4j).

Question31

1. The following course query "What are the names of the planes insuring a flight from Nice?" is written in CQL (Cassandra):

- ☒ SELECT Pname from Plane **join each** flight on plane.P# = flight.P# and DC = 'Nice';
- ☐ SELECT Pname from Plane where plane.P# = flight.P# and DC = 'Nice';
- ☐ SELECT Pname from Plane any f in flight satisfies f.DC = 'Nice'
- ☐ MATCH (p.plane[:insuresflight]→(f.flight) where f.DC = 'Nice' return p.PNAME

- **Réponse:**

- SELECT Pname from Plane join each flight on plane.P# = flight.P# and DC = 'Nice';

Explication: ~~Sans avoir le schéma exact de la base de données Cassandra, il est difficile de donner une réponse précise. Cependant, en se basant sur la syntaxe générale et les informations fournies, la requête qui semble la plus proche de la syntaxe correcte pour une requête qui joindrait deux tables (ou collections) en Cassandra est la troisième option. Les autres options ont des erreurs de syntaxe ou n'utilisent pas une syntaxe typique de Cassandra.~~

Quelques exemples SQL sur les SGBD cle-valeur NO SQL avec N1QL (Couchbase), CQL (Cassandra), ...



Exemple type : *Quels sont les numéros et noms des pilotes qui assurent un vol au départ de Nice ?*

N1QL :

```
SELECT PIL#, PNOM  
FROM pilote  
WHERE ALL F IN vol SATISFIES F.VD= 'Nice' et ADR = 'Nice' ;
```

CQL3

```
SELECT PIL#, PNOM  
FROM PILOTE  
JOIN EACH vol ON Pilote.pil#=Vol.PIL# et VD = 'Nice' et ADR = 'Nice' ;
```