LA MAÎTRISE DES SYSTÈMES D'INFORMATION

ADELI

La LETTRE n° 32

Juillet 1998

Chemin de la Liberté

Compte à rebours

« O temps, suspends ton vol ! et vous, heures propices,
Suspendez votre cours ! »
Lamartine (Le Lac)

Tout a été dit ou presque sur l'an 2000 : discours alarmistes, optimistes, humoristiques, voire eschatologiques.

« Pearl harbor électronique », « El Niño digital », apocalypse virtuelle, à la croisée des chemins techniques et religieux, l'an 2000 suscite tous les fantasmes ; chacun y voit midi à sa porte, si l'on peut dire, l'occasion de faire la fête ou de se retrancher dans un abri avec des stocks de nourriture et de batteries électriques. Un projet de loi punissant de prison les responsables informatiques n'ayant pas mis leur système en conformité a été un temps envisagé aux États-Unis, puis abandonné.

Dans un article récent (25/6/98) USA Today faisait part des inquiétudes de la CIA : la modification systématique de tous les logiciels de la terre serait l'occasion rêvée d'introduire des bogues supplémentaires qui pourraient être mis sur le dos du fameux bogue de l'an 2000, opportunité sans précédent d'obtenir des informations confidentielles ou d'implanter des virus. D'autant, renchérit l'article, que beaucoup de programmeurs sont étrangers…

L'an 2000 sera-t-il digne des meilleurs romans de science-fiction ? Devons-nous nous préparer à une avalanche de cyber-attaques, de procès extraordinaires et de faillites des plus grandes compagnies d'assurance ? Après tout, 2000 ne sera peut-être qu'une année comme une autre.

Faut-il se réjouir de l'effet magique d'un nombre rond ou avoir peur du zéro ?

Qui n'a jamais regardé avec émerveillement un compteur kilométrique passer de 99.999 à 100.000, comme si son automobile avait soudain pris un coup de neuf ?

L'idée de repartir à zéro, remettre les compteurs à zéro en oubliant le chiffre des centaines et à plus forte raison celui des milliers est plutôt agréable.

Voir l'année se terminer par 2 zéros, comme en 1800 ou 1900, cela est déjà extraordinaire, mais cela est statistiquement à portée de presque tout être humain et deviendra assez fréquent compte tenu de l'allongement de l'espérance de vie dans nos pays occidentaux.

Vivre une année à trois zéros est beaucoup plus rare, puisque cela ne sera que la seconde fois dans notre calendrier grégorien.

Le zéro est arbitraire

La peur de l'an mil est un phénomène historique contesté par beaucoup d'historiens qui reprochent à Michelet d'avoir créé ce mythe romantique quasiment de toutes pièces : la plupart des hommes de l'époque ignoraient sans doute la valeur exacte de l'année calendaire dans laquelle ils vivaient.

La diffusion des almanachs ne remonte qu'au XV^{ème} siècle, grâce au développement de l'imprimerie.

Le projet de prendre comme point de départ de l'ère chrétienne la naissance du Christ est l'œuvre d'un moine, Denys le petit, en 532. Cette idée fut reprise et adoptée par Charlemagne en l'an 800 (cela devait sans doute être plus agréable pour lui d'être sacré empereur lors d'une année ronde), puis progressivement adoptée en occident.

On peut se demander comment Denys le petit était arrivé, 532 ans après, à déterminer la date exacte de la naissance du Christ. Quelques éléments historiques tels que la mort d'Hérode et le fameux édit de recensement imposé par les romains, permettent d'élaborer des hypothèses.

Aujourd'hui les recherches continuent et tous les spécialistes sont unanimes au moins sur un point : sur l'échelle du temps chrétien, on ne sait pas exactement où est le zéro !

La place du zéro sur l'échelle du temps est donc purement arbitraire et ne devrait pas avoir de conséquence fâcheuse. Ce serait donc plutôt la distance à ce zéro arbitraire qui poserait problème, comme une ficelle trop courte qui nous rattacherait à un point fixe.

Que vient faire une horloge dans un ordinateur ?

Les premiers ordinateurs étaient conçus pour faire des calculs, il n'y a aucune raison de mettre une horloge dans un boulier ou dans une règle à calcul.

L'horloge des ordinateurs ne servait pas du tout à donner l'heure, elle ne servait qu'à synchroniser quelques opérations qui sinon se seraient déroulées dans un ordre inopportun.

Le temps de l'ordinateur était un temps arbitraire, en dehors du temps, un zéro comme celui de la naissance du christ auquel venaient s'ajouter des unités toutes comptées de la même façon par une horloge interne.

Si l'on voulait dater l'information, il fallait fournir la date à l'ordinateur, sous forme de carte « accept », afin qu'il sache bien que nous étions le 3 janvier 1972 et pas le 4.

Et puis un jour la « date système » est apparue (peut-être dans les premiers 360, son origine est difficile à dater, justement !). On ne lui a pas tout de suite fait confiance et l'on a continué à introduire la date au pupitre ou sur une carte perforée. D'ailleurs, cette date il fallait souvent la forcer pour faire croire à la machine que nous étions un autre jour, exécuter aujourd'hui les traitements d'hier, parce que nous avions pris un peu de retard…

Le premier PC annoncé par IBM en août 1981 ne possédait pas d'horloge interne propre (Real Time Clock). Un logiciel (situé en partie dans le BIOS résidant dans la mémoire ROM et en partie dans le DOS résidant en mémoire RAM) comptait des interruptions régulières qui étaient générées par une sorte de minuterie.

DOS convertissait le compteur de « tic-tac » obtenu en expressions conventionnelles de l'heure du jour lorsqu'un logiciel le lui demandait, et si, lors d'une telle requête DOS déterminait que minuit était passé, il incrémentait alors la date système.

Chaque fois que le système était relancé, la date et l'heure étaient initialisées au 1er janvier 1980 à 0 heure, 0 minutes et l'utilisateur était censé indiquer à nouveau au système les bonnes valeurs (ce qu'il ne faisait d'ailleurs pas toujours, car cela ne lui servait pas à grand chose, les applicatifs tournant alors sur son PC étant très cloisonnés).

Une première horloge matérielle fut introduite dans le PC/AT en août 1984 (une horloge temps réel CMOS Motorola MC146818). Cette horloge était alimentée par une batterie qui maintenait la date et l'heure de façon permanente lorsque le système était éteint.

La version 3 du DOS, qui sortit en même temps permettait de lire automatiquement dans l'horloge CMOS la date et l'heure au moment du lancement du système (boot) et mettait alors en concordance l'heure et le compteur de tic-tac.

En dehors de cette initialisation au moment du lancement, l'horloge matérielle n'était jamais consultée. Si l'utilisateur modifiait l'heure ou le jour par une commande DOS, cette modification n'était pas prise en compte par l'horloge.

Les versions ultérieures de DOS permirent de modifier l'horloge temps réel directement par une commande DOS, mais l'horloge n'était consultée (et cela est encore le cas aujourd'hui) qu'au moment du boot.

La plupart des PC et cartes mères fabriquées depuis fin 95 (entre autre par Compaq, Dell, Hewlett-Packard ou IBM) et tous les Apple sont équipés d'une horloge interne qui gère le passage à l'an 2000 sans aucun problème.

Les PC plus anciens peuvent avoir besoin d'un changement de BIOS (upgrade du BIOS possible sur certaines cartes mères fabriquées à partir de 1991) ou être remis à l'heure manuellement le premier janvier 2000. Rien de plus compliqué que de remettre les pendules à l'heure (à condition d'y penser), et pas de quoi en faire une pendule (expression étrange, s'il en est !), puisque d'ailleurs nos PC marquent rarement l'heure exacte.

Pourquoi les horloges retardent-elles ?

Nous avons tous remarqué que l'heure indiquée par nos PC est généralement fausse, souvent en retard de plusieurs minutes. L'intervalle de temps utilisé par DOS pour compter le « tic-tac » lui est fourni par une source de fréquence de 1,193 Mhz, non calibrée et généralement non réglable (1,193 est le quart de la fréquence de la première horloge PC qui était de 4,77 Mhz, et c'est aussi le tiers de la fréquence de balayage du standard de télévision NSTC qui est de 3,579 Mhz.

La minuterie est programmée pour diviser cette fréquence par 65 536 afin de produire un taux de 18,206 tic-tac par seconde, ce qui détermine la résolution de l'horloge DOS standard qui est environ de 54,925 millisecondes.

DOS autorise des expressions de temps en centième de seconde, mais n'est pas capable de maintenir cette précision interne ni de représenter l'heure de façon exacte avec la précision exacte du « tic-tac ».

L'exactitude de l'heure dépend de l'ensemble des logiciels qui tournent sur le PC et qui peuvent ne pas laisser suffisamment de temps au système pour que tous les intervalles soient comptés. Ceci n'est effectivement pas toujours possible et certaines interruptions sont perdues corps et bien.

L'accumulation d'interruptions perdues explique que sur un PC trop chargé l'horloge semble marcher plus lentement.

Mais à quoi sert donc l'heure ?

Question absurde pour notre monde envahi d'horloges en tout genre. L'heure sert à être à l'heure, un point c'est tout.

À l'heure au travail, à la réunion, au rendez-vous avec les copains. À savoir que l'on est en retard ou en avance. À titre individuel, le besoin de précision n'est pas très grand. Une horloge qui donne l'heure, cela est bien suffisant pour un usage courant, même sur un cadran de 12 heures : on sait bien si l'on est le matin ou l'après-midi.

Heure ou date sont arbitraires, le véritable risque en matière de temps c'est celui de la durée, surtout lorsqu'on est un être vivant, avec une durée ou espérance de vie, une date de validité inscrite quelque part, dans un code génétique ou dans un fichier.

Les lettres doivent être datées, le tampon de la poste peut faire foi, le moindre paquet d'information envoyé sur TRANSPAC est daté afin de pouvoir être reclassé à l'arrivée juste après le précédent et juste avant le suivant. La durée de cuisson du riz ou de tout autre aliment ne saurait être doublée sans dommage gustatif grave.

Un accouchement provoqué avec un mois d'avance pourrait provoquer quelques déficiences. Les logiciels s'arrêtent lorsque leur licence a expiré et les produits périmés sont enlevés du rayon. Dans tous ces cas, ce n'est pas le produit lui-même qui est observé.

Qu'il ait l'air jeune ou vieux, peu importe. Ce qui compte c'est l'information inscrite, sur l'enveloppe, sur l'étiquette, sur la carte d'identité. Le traitement automatisé se fie uniquement à l'information pour trier, archiver, détruire, à la fois d'autres informations, mais aussi des objets réels du monde réel, ou encore des objets pseudo réels de ce monde virtuel qui est celui des objets financiers : la date de l'écriture comptable déterminera des montants d'intérêts ou d'agios, dettes ou avoirs suivant le cas. Un mauvais calcul de la durée de stationnement devrait enrichir fabuleusement la ville de Paris.

Le temps du système et le temps des traitements se sont finalement mélangés, au risque d'une certaine confusion.

Le temps s'est progressivement infiltré dans la machine et ne manque pas une occasion de se rappeler à notre bon souvenir. Le prix des télécommunications varie en fonction de l'heure, les agendas se mettent à sonner pour nous rappeler nos rendez-vous, les pénalités s'accumulent pour le moindre traitement en retard.

Le temps c'est de l'argent, bien sûr ! Et certains distributeurs de logiciels (ne nommons personne, il vaut mieux être discrets) vont profiter de votre angoisse pour essayer de vous vendre la dernière version de leur traitement de texte ou de leur tableur, garantie « compatible an 2000 ».

De quoi faut-il donc avoir peur ?

De la peur de ceux qui ont peur ?
Des escrocs en tout genre qui trouveront le moyen d'en profiter ?
Des effets foudroyants du 1er bogue informatique planétaire : factures fausses, trains qui déraillent et pannes en tout genre ?

Il y aura bien sûr des problèmes, des calculs faux, des gens lésés et d'autres heureux grâce à des factures perdues ou des intérêts mal composés, peut-être des salaires versés avec un peu de retard, des remboursements de sécurité sociale erronés et quelques fichiers supprimés par erreur.

Ne soyons pas trop pessimistes

Le pire devrait être évité grâce aux armées de programmeurs se livrant depuis plusieurs mois à des contrôles systématiques, débusquant au passage d'autres bogues enfouis dans les lignes de code. La profession informatique devrait sortir de cette expérience avec une nouvelle maturité, la prise de conscience de l'importance des tests et un peu plus de rigueur dans l'utilisation des outils de génie logiciel.

Pour ma part, j'ai bien peur que les horloges ne s'arrêtent pas, même une seconde, malgré les supplications des poètes. ▲

Martine Otter

Après l'an 2000

Un vieux programmeur COBOL, paisiblement installé dans le chômage depuis quelques années, attendait sa préretraite.

Mais à l'approche de l'an 2000, on est venu le prier d'aller réparer les bogues des programmes contemporains de son activité passée. Le pauvre type se voit alors contraint de reprendre ses activités à un rythme d'enfer.

Un jour, il tombe sur une annonce :
"On cherche des volontaires pour aller 3 ans dans l'espace faire le tour de Mars, en état de léthargie artificielle."
Le type se dit "Parfait, je serai de retour après l'an 2000 et enfin tranquille."

Sa candidature est retenue. Et hop, le voilà dans le vaisseau spatial, on lui injecte de la glycérine, on le congèle et la fusée part.

Quand il reprend connaissance, après un profond sommeil dont il n'a pu apprécier la durée, il est de retour sur une terre étrange où les gens sont habillés bizarrement.

La personne qui l'accueille lui déclare "Nous sommes désolés, votre vaisseau spatial a subi l'une des erreurs que nous redoutions au passage à l'an 2000 et vous avez tourné un peu plus longtemps que prévu."

"Mais alors, en quelle année sommes-nous ?" dit le programmeur, affolé.
"En 9998" répond l'autre. " À propos, je vois sur votre fiche que vous connaissez le COBOL !" ▲

Recueilli et adapté par Alain Coulon

Le bêtisier de l'été

La-men-table ! ! !

Selon le principe de Peter, les employés très incompétents ne posent pas de problème, car ils sont rapidement détectés et mis "hors d'état de nuire". Voici quelques exemples prouvant que ce n'est malheureusement pas toujours vrai.

Dans certaines grosses entreprises, personne n'a le temps de rien, et surtout pas de vérifier que les programmeurs travaillent correctement, en suivant les procédures, etc.

Conséquence : le travail est uniquement évalué au vu des résultats : ils sont corrects, donc le programme est bon. Que l'individu ait écrit 3000 lignes alors qu'il aurait pu le faire en 500, qu'il y ait passé 3 semaines au lieu de 3 jours… ceci est secondaire.

Ce comportement est à rapprocher de certains hauts reponsables qui ne jugent le bon fonctionnement d'une entreprise qu'au travers d'un seul élément : le montant de la marge bénéficiaire.

Tous les exemples cités dans cet article sont rigoureusement authentiques… et proviennent tous de la même entreprise ! Simplement, comme il s'agit de programmes écrits en cobol dans un environnement IBM gros système (SGBD DB/2, transactionnel CICS…), j'ai transcrit le code en "pseudo-code" français, pour permettre à tout un chacun, même non informaticien, de comprendre.

Il suffit simplement de savoir que "`A ← B + C`" signifie "mettre en A la somme de B et C", et que "`A ← B`" signifie "copier le contenu de B dans A".

Pourquoi faire simple quand on peut faire compliqué ?

C'est le défaut des programmeurs qui se jettent sur le travail d'écriture du code sans analyser, et qui ont tendance à associer productivité et nombre de lignes de code écrites.

Je précise bien "de code", car en général, plus le programme est mal écrit, moins y il a de commentaires pour aider à la compréhension.

Premier cas

Trouvé plusieurs fois dans un seul et même programme, le code ci-dessous, en cobol, occupait six lignes (if … / then / … / else / … / end-if) !

Ce qui a été écrit	Ce qu'il suffisait de faire

`si A = 0`
`  alors B ← 0`
`  sinon B ← A`
`fin`
	`B ← A`

Deuxième cas

Ceci est un grand classique :

Ce qui a été écrit	Ce qu'il suffisait de faire

si condition
alors
  traitement 1
  traitement 2
sinon
  traitement 1
  traitement 3
fin
	traitement 1
	si condition
	  alors traitement 2
	  sinon traitement 3
	fin

Si traitement 1 ne fait qu'une seule ligne, ce n'est pas encore trop grave ; mais j'ai rencontré de tels cas avec plus de dix instructions bêtement répétées dans les deux termes de l'alternative.

Troisième cas

Une simple variante du précédent :

Ce qui a été écrit	Ce qu'il suffisait de faire

si condition1
  alors fin
si condition2
  alors fin
si ...
si conditionN
  alors fin
	si condition1
	ou condition2
	ou ...
	ou condition3
	  alors fin

Quatrième cas

Comment avoir un gros programme en évitant de varier le code :

Ce qui a été écrit	Ce qu'il suffisait de faire

fonction x(n)
début
  traitement (n)
fin fonction
	fonction x
	début
	  pour I variant de 1 à 10
	  traitement (I)
	  fin
	fin fonction

(40 fois dans le programme) : pour I variant de 1 à 10 exécuter x(I) fin	(40 fois dans le programme) : exécuter x

Il faut savoir que la boucle ci-dessus, écrite en cobol et bien aérée, occupait à chaque fois 6 lignes, et que la fonction appelée ne l'était jamais autrement que dans une telle boucle !

Cinquième cas

Le programmeur disposait d'une fonction envoyant l'information transmise vers l'imprimante. Ayant des espacements variables à faire, il a défini des fonctions lui permettant de créer divers espacements en une seule instruction. Après avoir défini une fonction `espace1` générant une ligne blanche, il a écrit deux autres fonctions générant respectivement 2 et 3 lignes blanches :

```
fonction espace2
début
   exécuter espace1
   exécuter espace1
fin fonction
```

```
fonction espace3
début
   exécuter espace1
   exécuter espace1
   exécuter espace1
fin fonction
```

Jusqu'ici, tout va bien.

Par contre, quand il a voulu écrire une fonction permettant de créer un nombre N de lignes blanches, il s'est quelque peu emmêlé les pinceaux. Jugez-en :

Ce qui a été écrit	Ce qu'il suffisait de faire

fonction espace(N)
début
  Triple ← partie entière (N/3)
  Reste ← reste (N/3)
  pour I variant de 1 à Triple
  exécuter espace3
  fin
  si Reste = 2
  alors exécuter espace2
  sinon
  | si Reste = 1
  | alors exécuter espace1
  | fin
  fin
fin fonction
	fonction espace(N)
	début
	  pour I variant de 1 à N
	  exécuter espace1
	  fin
	fin fonction

Sans commentaires…

Sixième cas

Ici, notre programmeur atteint de "codorrhée"[1] s'est surpassé, au point que je ne puis que vous expliquer le problème sans pseudo-code, ou bien alors réserver 8 pages de LA LETTRE rien que pour çà !

Le problème : il s'agissait de lire un fichier, de cumuler certaines valeurs en fonction de divers critères, et d'écrire les résultats cumulés dans un autre fichier. Le fichier en entrée contenait onze informations numériques (quantitée livrée en moins d'un jour, quantité livrée le lendemain, quantité livrée en 2 jours, …, quantité totale à livrer, quantité totale déjà livrée, etc.), toutes du même format, en deux groupes distincts (soit au total 22 valeurs). Cinq niveaux de cumul étaient à faire (par produit, par gamme, par vendeur, par secteur, par région).

La bonne réponse : il suffisait de définir, tant en entrée qu'en sortie, un tableau à deux dimensions (2 par 11), en interne un tableau à 3 dimensions (5 par 2 par 11) et d'utiliser des indices pour parcourir l'ensemble des données. Ceci se fait aisément en 200 lignes de cobol.

La programmation effectuée : il y a 11 informations ayant chacune un sens propre, donc on déclare 11 variables (que la sémantique des données soit sans importance lui échappe complètement). Il y en a 2 groupes, ce qui fait 22 variables pour le fichier en entrée, et autant pour le fichier en sortie. Les 5 niveaux de cumul nécessitent autant de structures de 22 variables, soit 110 zones de travail. Bien sûr, pour le traitement, il faut faire les assignations de cumul et de copie pour chaque variable. Bref, le programme fourni a ainsi atteint 800 lignes !

Quelle différence y a-t-il entre une table relationnelle et un fichier ?

Nombreux sont ceux qui ne le savent pas et créent des requêtes plus complexes que nécessaire et qui coûtent généralement beaucoup plus cher en exploitation.

---

[1] *Une personne atteinte de "logorrhée" (en grec, logos = la parole) ne peut s'empêcher de parler continuellement. Vous pouvez donc aisément en déduire le sens du mot "codorrhée".*

Premier cas

Nous avons deux tables A et B. A est identifié par une clé C, B par une clé D, et contient un champ X ayant une contrainte d'intégrité sur le champ C de A, c'est-à-dire qu'on ne peut créer ou mettre à jour une ligne de B qu'en mettant obligatoirement dans X une valeur correspondant à une ligne existante de A.C. En termes merisiens, on a une cardinalité (1,1) de B vers A.

Donc, quand on lit B, on est sûr qu'il existe dans A une ligne telle que A.C = B.X. Pourtant, on trouve parfois des requêtes de ce genre :

Ce qui a été écrit	Ce qu'il suffisait de faire
`lister D de B` `  dont B.X = A.C`	`lister D de B`

En d'autres termes, notre programmeur demande seulement les lignes de B qui correspondent à une ligne existante dans A. Alors que toutes les lignes de B sont dans ce cas… Le moteur du SGBD effectuera le contrôle (il n'utilise les contraintes d'intégrité qu'en mise à jour) et consommera donc plus de ressources.

Nota : ce contrôle aurait été pertinent si le champ D avait été facultatif ou si aucune contrainte d'intégrité n'avait été définie.

Deuxième cas

Un programme reçoit les paramètres P1 et P2, et doit lire une table T (contenant notamment un champ K), avec un filtre dépendant des paramètres :

- si P1 = 'A', lire les lignes telles que K = P2 ;
- si P1 = 'B', lire les lignes telles que K ≥ P2 ;
- si P1 = 'C', lire les lignes telles que K > P2.

Qu' a fait notre programmeur codorrhétique ? voyez vous-mêmes :

Ce qui a été écrit	Ce qu'il suffisait de faire

`définir liste1 curseur pour`
`   lister K, X, Y, Z de T`
`     dont K = P2 ;`
 	
`définir liste2 curseur pour`
`   lister K, X, Y, Z de T`
`     dont K >= P2 ;`
 	
`définir liste3 curseur pour`
`   lister K, X, Y, Z de T`
`     dont K > P2 ;`
 	
`si P1 = 'A'`
`alors ouvrir liste1`
`   faire jusqu'à fin de liste1`
`   |  lire liste1`
`   |  exécuter traitement`
`   fermer liste1`
`fin`
 	
`si P1 = 'B'`
`alors`
`   ouvrir liste2`
`   faire jusqu'à fin de liste2`
(suite colonne 2)
	`définir liste curseur pour`
	`   lister K, X, Y, Z de T`
	`     dont (K=P2 et P1≠'C')`
	`       ou (K>P2 et P1≠'A') ;`
	
	`ouvrir liste`
	`faire jusqu'à fin de liste`
	`|  lire liste`
	`|  exécuter traitement`
	`fermer liste ;`
	(suite de la colonne 1)
	`   |  lire liste2`
	`   |  exécuter traitement`
	`   fermer liste2`
	`fin`
	
	`si P1 = 'C'`
	`alors`
	`   ouvrir liste3`
	`   faire jusqu'à fin de liste3`
	`   |  lire liste3`
	`   |  exécuter traitement`
	`   fermer liste3`
	`fin`

À noter que dans ce cas, les performances ne sont pas dégradées, mais bonjour la complication du programme…

Troisième cas

Il faut récupérer des informations dans deux tables, sachant que pour chaque ligne de la table T1 correspondent de 0 à N lignes de la table T2. Chaque ligne de T1 est identifiée par le champ C et contient le champ E, chaque ligne de T2 est identifiée par les champs (C, D) et contient le champ F. Dans le pseudo-code ci-dessous, il suffit de savoir que pour chaque information lue, on sait si elle a une valeur ou pas.

Il existe en SQL une fonction appelée jointure externe qui permet de lier deux tables, et de récupérer les données de l'une même s'il n'y a pas de correspondance dans l'autre. En d'autres termes, si la table T1 contient les lignes [C1, C2, C3] et la table T2 les lignes [(C1,D1), (C1,D2), (C3,D2), (C4, D4)], une requête avec une jointure simple ne fournira que les combinaisons existantes entre les deux tables, soit :

```
C1 D1 E1 F1
C1 D2 E1 F2
C3 D2 E3 F3
```

…alors qu'une requête avec une jointure externe sur la table T1 fournira :

```
C1 D1 E1 F1
C1 D2 E1 F2
C2 -  E2 -
C3 D2 E3 F3
```

Malheureusement, bien que le programme en question ait été écrit bien après l'installation d'une version du SGBD permettant ce genre de jointure, le programmeur a complètement ignoré cette option, et a travaillé comme ceci :

Ce qui a été écrit	**Ce qu'il suffisait de faire**

définir liste1 curseur pour
  lister C, E de T1 ;
 	
définir liste2 curseur pour
  lister C, D, F de T2
  dont C = T1.C ;
 	
ouvrir liste1
faire jusqu'à fin de liste1
| lire liste1
| ouvrir liste2
| si liste2 vide
| alors
| exécuter traitement A
| sinon
| faire jusqu'à fin de liste2
| | lire liste2
| | exécuter traitement B
| fin
| fermer liste2
fermer liste1 ;
	définir liste curseur pour
	  lister T1.C, T2.D, T1.E, T2.F
	  de T1,
	  joint à T2 si T2.C = T1.C ;
	
	ouvrir liste
	faire jusqu'à fin de liste
	| lire liste
	| si D a une valeur
	| alors exécuter traitement B
	| sinon exécuter traitement A
	| fin
	fermer liste ;

Bref, plutôt que de laisser le moteur du SGBD faire tout le travail en une seule fois, l'individu a géré un appareillage de fichiers, obligeant le moteur du SGBD à créer un nouveau curseur sur la deuxième table pour chaque ligne lue de la première table. Je ne vous dis pas les performances…

Quatrième cas

En simplifiant, il s'agissait d'afficher à l'écran une liste d'informations, l'affichage de chacune étant conditionnée par six paramètres (arrivées/mouvements, pour un responsable donné ou pas, pour une livraison donnée ou pas, pour un état d'en-cours ou pas, pour un fournisseur donné ou pas, pour un code pièce donné ou pas).

La complication[2] des tables du SGBD faisait que chaque combinaison des deux premiers paramètres nécessitait des requêtes différentes. Par contre, pour les quatre autres paramètres, il n'y avait pas de difficulté particulière. Voyons comment aurait pu être réalisée la recherche pour une combinaison donnée des deux premiers paramètres… et comment elle l'a été en réalité.

Nous appellerons P1, P2, P3, P4 les 4 paramètres, et V1, V2, V3, V4 les variables contenant les valeurs recherchées (la valeur "blanc" signifiant "pas de recherche").
Une condition "V=blanc ou P=V" est remplie (c-à-d considérée comme vraie) :
- si V est à blanc, la valeur de P important donc peu (pas de filtrage)
- ou si P a la même valeur que V (filtrage).

Ce qui a été écrit

```
définir liste1 curseur pour
lister P1, P2, P3, P4, A, B de T ;

définir liste2 curseur pour
lister P1, P2, P3, P4, A, B de T
  dont P1 = V1 ;

définir liste3 curseur pour
lister P1, P2, P3, P4, A, B de T
  dont P2 = V2 ;

définir liste4 curseur pour
lister P1, P2, P3, P4, A, B de T
  dont P1 = V1
    et P2 = V2 ;
```

...(je vous épargne le détail)

```
définir liste16 curseur pour
lister P1, P2, P3, P4, A, B de T
  dont P1 = V1
    et P2 = V2
    et P3 = V3
    et P4 = V4 ;

si V1=blanc
alors
|  si V2=blanc
|  alors
|  |  si V3=blanc
|  |  alors
|  |  |  si V4=blanc
|  |  |     alors ouvrir liste1
|  |  |     sinon ouvrir liste2
|  |  sinon
|  |  |  si V4=blanc
|  |  |     alors ouvrir liste3
|  |  |     sinon ouvrir liste4
|  sinon
|  |  si V3=blanc
|  |  alors
```

...(je vous épargne le détail)

```
faire jusqu'à fin de liste
|  si V1=blanc
|  alors
|  |  si V2=blanc
```

(suite colonne 2)

Ce qu'il suffisait de faire

```
définir liste curseur pour
   lister P1, P2, P3, P4, A, B de T
     dont (V1=blanc ou P1=V1)
       et (V2=blanc ou P2=V2)
       et (V3=blanc ou P3=V3)
       et (V4=blanc ou P4=V4) ;

ouvrir liste
faire jusqu'à fin de liste
|  lire liste
|  exécuter traitement
fermer liste ;
```

(suite de la colonne 1)

```
|  |  alors
|  |  |  si V3=blanc
|  |  |  alors
|  |  |  |  si V4=blanc
|  |  |  |     alors lire liste1
|  |  |  |     sinon lire liste2
|  |  |  sinon
|  |  |  |  si V4=blanc
|  |  |  |     alors lire liste3
|  |  |  |     sinon lire liste4
|  |  sinon
```

...(je vous épargne le détail)

```
|  exécuter traitement
fin

si V1=blanc
alors
|  si V2=blanc
|  alors
|  |  si V3=blanc
|  |  alors
|  |  |  si V4=blanc
|  |  |     alors fermer liste1
|  |  |     sinon fermer liste2
|  |  sinon
|  |  |  si V4=blanc
|  |  |     alors fermer liste3
|  |  |     sinon fermer liste4
|  sinon
```

...(je vous épargne le détail)

```
fin
```

[2] *et non pas "complexité".*

Et ceci pour chacun des quatre cas de figure. Notre adepte du "big is beautiful" a donc crée 64 requêtes au lieu de quatre, avec toutes les contraintes qui s'ensuivent, et a ainsi justifié plusieurs semaines de travail pour un monstrueux programme de plus de 4000 lignes.

La solution simple se faisait en 3 jours pour un programme de 700 lignes…

Les index à l'index

Dans un SGBD relationnel, il est primordial de définir un ou plusieurs index correspondant aux principaux types d'accès. Ces index doivent être optimisés, sinon l'utilisation des tables peut coûter très cher en temps machine.

Un exemple pour bien comprendre. Une table a un index sur plusieurs champs, et dans cet ordre : A, B, C. Si on fait une recherche avec une sélection sur A, C, l'index sera utilisé pour le premier champ. Si on fait une recherche avec une sélection sur B, C, l'index ne sera pas utilisé, et la recherche sera entièrement séquentielle.

Note pour mieux comprendre :
un index est dit "unique" si pour une valeur (ou un ensemble de valeurs pour une clé à plusieurs champs) donnée d'un index il ne peut exister dans la table qu'une seule ligne. A contrario, un index "multiple" permet d'avoir plusieurs lignes avec le même identifiant.

Bien sûr, plus il y a d'index, et plus il y a de champs dans un index, plus les mises à jour sont chères. Il faut donc judicieusement doser. Voici donc quelques exemples d'intempérance, qu'un administrateur de Base de Données compétent ne devrait jamais laisser passer... mais qui sont hélas passées !

Premier cas

Une table a été définie avec un index unique sur les champs A, B, C. Un deuxième index, également unique, a été défini sur les champs D, B, A, C, E.

Remarquez que les trois champs du premier index figurent dans le second. Cependant, puisque le premier champ est D, on peut quand même mettre les trois autres, puisqu'il s'agit d'une séquence différente, et que pour une valeur donnée de l'ensemble D, B, A il est possible d'avoir plusieurs valeurs de C.

Par contre, l'ensemble D, B, A, C est forcément unique, donnant donc une seule ligne possible, et donc une seule valeur de E. Avoir mis ce cinquième champ dans l'index est "non pertinent", puisqu'aucun affinage supplémentaire de séquence n'est possible. Par contre, toute mise à jour de E nécessitera une mise à jour de l'index, et donc un coût supplémentaire ; et ce sans aucune possibilité d'économie par ailleurs.

Deuxième cas

Comme ci-dessus, une table a été définie avec un index unique sur les champs A, B, C. Un deuxième index a été défini sur les champs D, B, A, C, mais multiple.

Il est évident que ce second index contenant les champs du premier est unique de fait. L'avoir défini multiple est non seulement stupide, mais peut occasionner également un coût supplémentaire lors des requêtes, puisque si on fait une recherche impliquant l'usage du second index, le moteur du SGBD prévoira la possibilité de récupérer plusieurs lignes pour un même identifiant.

Troisième cas

Une table a été définie, un de ses champs (A) pouvant avoir un nombre indéterminé de valeurs (A1,…An), mais dont deux d'entre elles (mettons A3 et A7) ont une signification importante, et servent souvent de critère de sélection. Le "concepteur" de la table a donc ajouté un champ supplémentaire B pouvant avoir trois valeurs : B=1 si A vaut A3, B=2 si A vaut A7, B=3 dans tous les autres cas.

Ceci pourrait être utile si on voulait lister cette table en commençant par A3 et A7, même si leur valeur les place en séquence n'importe où dans la table.

Dans la réalité, il s'avère que tous les programmes utilisant cette table, soit s'intéressent exclusivement aux lignes contenant A3 ou aux lignes contenant A7, ou bien traitent tout sans donner d'importance particulière à A3 et A7. Bref, le seul programme utilisant le champ B… est celui qui met à jour la table ! De plus, le champ B ne figure dans aucun des index définis pour cette table, ce qui le rend totalement inutile.

En guise de conclusion

On pourrait continuer longtemps ainsi, parler des tables "dupliquées" parce qu'une partie de leur contenu a un usage spécifique, et qu'il a été "jugé" plus simple de dupliquer le contenu plutôt que d'ajouter un champ différenciateur, avec comme résultante autant de programmes de mise à jour que de versions de table, avec 90% du code identique (mais pas commun, ce serait trop astucieux !)… etc.

Mais il faut bien s'arrêter quelque part, et je le ferai sur un dernier exemple.

Deux programmeurs avaient été chargés de modifier des programmes transactionnels existants, afin de permettre aux utilisateurs de définir des critères de sélection (comme Domaine, Famille de produit, Fournisseur, Période…) et d'éditer des listes correspondant à ces critères. Selon les fonctions, il y avait chaque fois entre 8 et 12 critères (dont plusieurs se retrouvaient dans toutes les fonctions), chacun d'entre eux pouvant être, au choix :

- laissé à blanc (= non fourni) ;
- '*', demandant de filtrer sur une liste de valeurs prédéfinies ;
- rempli avec une valeur exacte ;
- pour quelques critères, rempli avec un préfixe (ex: 'A' renverra toutes les informations dont la valeur commence par 'A').

Bref, rien de bien compliqué. On avait donc en gros à chaque fois 10 paramètres, chacun pouvant avoir 3 ou 4 types de valeurs.

Comment a procédé le premier ?
Simplement en créant un premier algorithme généralisé permettant de remplir la table de paramètres destinée à SQL, quatre algorithmes correspondant à chaque type de valeur, puis en les appliquant (par appel de fonction) à chaque paramètre. Dans un langage évolué tel que PL/1, Pascal, C ou même Qbasic, ceci aurait aisément tenu en une centaine de lignes.
Le cobol n'étant malheureusement pas aussi souple, quelque 200 lignes ont été nécessaires. Comme malheureusement l'implémentation locale n'autorise pas les requêtes dynamiques (c'est-à-dire construites dans le programme à l'exécution) en production – qui auraient pu être générées avec 20 lignes supplémentaires –, il a utilisé des requêtes statiques (c'est-à-dire écrites dans le source du programme et compilées avec lui), beaucoup plus complexes puisque non optimisables. Ceci a entraîné un supplément de 200 autres lignes.

En résumé, il a ajouté en moyenne 400 lignes à chaque programme, mis 3 jours pour mettre au point le premier, puis seulement ½ journée pour chacun des suivants. Soit 8 programmes modifiés et testés en à peine plus d'une semaine.

Et comment a travaillé l'autre ?
Eh bien, lui a vu non pas 4 fonctions à appliquer à 10 variables, mais 10 variables ayant chacune 3 ou 4 traitements possibles. Bref, il a dupliqué à mort, sans algorithme commun.
Le résultat ?

1.0 à 1.200 lignes de plus pour chaque programme, une semaine de travail pour chacun, soit un mois pour traiter seulement 4 programmes… et bien sûr des bogues résiduels.

Théorème

Une personne très incompétente ne peut être neutralisée que si ses supérieurs sont suffisamment compétents pour s'en apercevoir et agir en conséquence. ▲

Jean-Luc Blary

Le téléphone portable…

…un authentique BESOIN

La problématique du vendeur tient en une triple constatation :

- 1° le client est celui qui a mon argent dans sa poche ;
- 2° moi, je suis obligé de vendre pour survivre ;
- 3° mais mon client n'est pas obligé d'acheter.

Dans un marché "de vendeurs", l'offre des fournisseurs est supérieure à la demande naturelle des acheteurs potentiels ; dans une telle situation, le commerçant doit dynamiser l'attrait de son produit, de préférence, en développant un argumentaire qui flatte les motivations de ses futurs clients.

Tout le monde connaît le FOMEC, mnémonique du camouflage militaire qui se décline en : Forme, Ombre, Mouvement, Éclat, Couleur.

Par analogie, pour former les futurs vendeurs, les écoles de commerce déclinent le mot **BESOIN** considéré comme un acronyme : **Bien-être**, **Égoïsme**, **Sécurité**, **Orgueil**, **Intérêt**, **Nouveauté**

Application au téléphone mobile

Lorsqu'on demande pourquoi le constructeur national se diversifie dans les T.P. (des Travaux Publics au Téléphone Portable en passant par la Télévision Publicitaire), la réponse tombe froide et logique :

"C'est un marché de n milliards de dollars".

J'ai oublié la valeur de n, l'étendue du marché (national, européen, mondial) et sa durée (globale ou annuelle). Mais, j'ai retenu qu'il y avait beaucoup d'argent à gagner en soulageant le portefeuille des consommateurs.

Reprenons notre acronyme et appliquons-le au téléphone portable.

B comme bien-être

Le téléphone portable est un bien facile à acquérir, d'usage simple, sans aménagement particulier (aucun branchement, aucune acquisition de mobilier). Il est peu encombrant et ne nécessite pas d'investissement.
Il suffit de suivre les flèches qui conduisent au magasin, de payer une somme modeste et de repartir avec le produit en état de marche dont on maîtrise les deux principales fonctions : appeler quelqu'un pour lui dire quelque chose et répondre à un appel.
Et on pourra toujours envisager de s'en débarrasser facilement si l'objet cesse de plaire.

E comme égoïsme

Le portable est un objet rigoureusement individualisé, identifié au chiffre de son propriétaire. Il fait corps avec son possesseur et ne supporte aucun prêt de longue durée.

S comme Sécurité

Le portable répond à la préoccupation sécuritaire qui permet de joindre ses proches à tout moment ou d'être joint par eux.
"Ne vous inquiétez pas, tout va bien et je rapporte le pain !"
Le maintien physique de ce cordon ombilical permet d'être rassuré en temps immédiat sur l'état de ses amis.

O comme Orgueil

Le téléphone portable est un objet valorisant que l'on est fier d'arborer, de préférence sous un étui fixé à la ceinture, comme un colt de western. Il souligne l'importance de son possesseur lorsque celui-ci est appelé en public.

I comme Intérêt

L'éventail des formules d'abonnement, l'attrait des remises, des cadeaux, la comparaison des offres donne à tout acheteur l'impression d'avoir fait le meilleur achat aux meilleures conditions.
Le forfait mensuel qui dissocie la consommation du paiement crée un climat de liberté et suggère une impression de gratuité.
Quelle satisfaction de pouvoir afficher auprès de ses amis un prix de revient inférieur au leur pour une utilisation plus souple !

N comme Nouveauté

Qui aurait imaginé cette explosion technologique, il y a seulement quelques années lorsqu'on misait sur le téléphone de voiture ou sur le bi-bop tombés aujourd'hui en parfaite désuétude.

En résumé, le portable est l'un des premiers produits qui répond à l'ensemble des facteurs de motivation des acheteurs.

Ainsi, le téléphone portable est un véritable Besoin, au sens des écoles commerciales. La vertigineuse croissance des ventes de ces petits appendices sonores le confirme.

Mais peut-être est-ce le besoin vu du côté du marchand.

N'existerait-il pas chez l'individu des besoins différents qu'il suffirait de dynamiser ?

Il nous reste à décliner, de la même façon, avec la même efficacité, des acronymes tels que plaisir, paix et bonheur. ▲

Alain Coulon

Déchiffrage nécessaire

Sachez lire les documentations commerciales

Nous vous soumettons une grille de déchiffrage des informations commerciales.
À gauche, l'expression utilisée par le promoteur ; à droite, une interprétation critique.

Les slogans

Nouveau	Couleur différente du modèle précédent
Entièrement nouveau	Pièces non interchangeables avec le modèle précédent
Nouvelle génération	L'ancien a échoué, celui-ci marchera peut-être mieux
Inégalé	Presque aussi bon que la concurrence
Robuste	Trop lourd pour le soulever
Léger	Plus léger que robuste
Multimédia	Émet un bip après chaque erreur
Simple d'utilisation	Fonctions réduites pour limiter les coûts de fabrication
Langage indépendant de la plate-forme	Ne marche sur aucune plate-forme
Faible maintenance	Impossible à réparer
Compatible Windows 95	Ne marche pas avec Windows 3.11
Compatibles avec tous les standards	Compatible avec certains de nos standards
Ergonomie avancée	Incompréhensible
Futuriste	Ne marchera qu'avec les futurs ordinateurs
Haute fiabilité	Marche assez longtemps pour pouvoir sortir de l'usine

Les argumentaires

Des années de développement	On a enfin réussi à faire marcher une version
Nous avons expérimenté de nombreuses approches	Nous continuons à tourner en rond
Nous préparons un rapport détaillé sur une nouvelle solution du problème	Nous venons de prendre trois jeunes stagiaires
Une percée technologique majeure	Ça marche comme-ci comme-ça, mais ça en jette !
Le client sera satisfait	Nous avons déjà tellement dépassé les délais que le client sera heureux d'avoir enfin quelque chose
Ce concept sera entièrement révisé	La seule personne qui le connaissait vient de quitter le projet
Les résultats des tests sont extrêmement satisfaisants	Nous sommes très surpris que ce système fonctionne

Recueilli par Alain Coulon

Europanto

De la débabélisation à l'Europanto

Dans la lettre n° 29 de janvier 1998, nous avons publié un article qui esquissait la future migration des langues européennes vers le tout anglais.

Un article, destiné au n° 30 d'avril 1998, préconisait une méthode de révision et de simplification de l'orthographe de la langue anglaise. Cette transformation consistait à réconcilier l'orthographe et la prononciation. L'exemple (prélevé sur Internet) aboutissait à un résultat du type "Ze drim vil becom tru" jugé trop germanique par notre comité de lecture.

Le présent article, au lieu de se perdre dans une parodie comme le précédent (évoqué, puis révoqué) se bornera à donner une information objective, publiée dans "Le Monde" du vendredi 24 avril 1998 et reprise par France-Inter.

Diego Marani est interprète auprès des Communautés européennes. Immergé dans l'anbrux (anglais bruxellois) jargon dans lequel s'effectuent les conversations entre européens, il propose, mi-facétieux, mi-sérieux, une novlangue européenne qu'il appelle Europanto.

Pour ce faire, il s'est appuyé sur l'exemple de la monnaie unique. L'euro, comme son prédécesseur l'écu, est construit à partir d'un panier de monnaies européennes : x % de mark allemand, y % de florin hollandais, z % de franc français, etc.

De même, il construit l'Europanto à partir d'un panier des principales langues européennes : anglais, allemand, français, italien, espagnol et néerlandais. Tant pis pour les grecs, les portugais et autres scandinaves.

L'hebdomadaire belge francophone "Le soir illustré" publie dans chacun de ses numéros, une chronique émise par l'"Instituto Europanto de Bricopolitik" dont Diego Marani se proclame Directeur.

Ça donne le texte suivant que j'ai lu plusieurs fois avant d'en attraper le sens.

"De problems des Kossovo esse essentialmente zwei ;
primero, er esse tropo manige people there die speake de wrong lingua.
secundero, er est tropo manige people die esse aan seine platz nicht.

After todo est handelt van zwei banale problemas where Belgica can seine preziosa experienza put aan el service des pax.
De Europanto Instituto de Bricopolitik habe de question deepamente gestudied und eine logica soluzie gefounded".

Ce charabia permet de faire passer des idées hardies, sous une forme humoristique, alors que l'expression de ces mêmes élucubrations, dans une langue officielle, provoquerait des réactions virulentes.

Pour parler ou écrire Europanto, chacun peut partir de sa langue maternelle en remplaçant les tournures qui risquent d'être mal comprises par des à peu près dans une autre langue qu'il maîtrise plus ou moins.

Ach so, Io wish sie eine felice Europa. Nostri systems de informacion will be capiti by alle nuestros european amici.

Modéliser, par des valeurs consensuelles, certains concepts facilitent la communication.

Transformer des informations en données susceptibles d'être traitées par de systèmes informatiques, c'est bien pour échanger sur les problèmes techniques et économiques.

Mais, s'interdire de communiquer des idées personnelles, précises et nuancées, en raison des obstacles linguistiques, que certains s'obstinent à vouloir ignorer, est-ce mieux ? ▲

Alain Coulon

Logique et management

- Dites à quelqu'un qu'il y a trois milliards d'étoiles dans la galaxie et il vous croira. Dites-lui que la peinture n'est pas sèche et il aura besoin de toucher pour en être sûr.
- L'hiver est la saison pendant laquelle les gens essaient de conserver leur maison aussi chaude que pendant l'été, lorsqu'ils se plaignent de la chaleur.
- La longueur de la minute dépend du côté de la porte des toilettes où l'on se trouve.
- Si vous faites croire aux gens qu'ils pensent, ils vous aimeront, mais si vous les faites penser, ils vous haïront.
- Les principales causes des problèmes sont les solutions.
- Mesurez avec un micromètre, marquez avec une craie, coupez avec une hache.
- L'ordre est le plaisir de la raison ; mais le désordre est le délice de l'imagination.
- L'expérience, c'est ce don merveilleux de reconnaître les erreurs quand on les refait.
- Les hommes raisonnables ne font jamais rien.
- Le Quotient Intellectuel d'un groupe est égal au Q.I. du membre le plus bête du groupe, divisé par le nombre de personnes dans le groupe.
- La somme de l'intelligence répartie sur la planète est constante ; la population augmente.
- L'erreur est humaine ; l'attribuer à quelqu'un d'autre est encore plus humain.
- Si un câble informatique a une extrémité, alors il en a une autre.
- Aucun plan de bataille ne survit au contact avec l'ennemi.
- Il vaut mieux essayer de changer le cahier des charges pour l'adapter au programme que de s'échiner à tenter l'inverse.
- Si on vous donne un travail difficile, proposez-le à quelqu'un de plus paresseux que vous, il trouvera plus simple.
- Un système tendra à grossir dans le sens de la complexité plutôt que de la simplification jusqu'à ce que l'instabilité résultante devienne intolérable.
- Toute technologie est largement dominée par ceux qui peuvent gérer ce qu'ils ne comprennent pas.

- Il est tellement plus facile de proposer des solutions quand on ne connaît rien au problème.
- L'information se détériore au fur et à mesure qu'elle circule dans les bureaucraties.
- Dans chaque organisation, il y a toujours une personne qui comprend ce qui se passe ; cette personne doit être licenciée.
- Il est plus facile d'accorder l'absolution qu'une permission.
- L'un des avantages de fixer des objectifs vagues à un projet, c'est que vous n'aurez pas de difficultés à estimer les dépenses correspondantes.
- Un projet mal planifié prend trois fois plus de temps que prévu, alors qu'un projet soigneusement planifié ne prend que deux fois plus de temps.
- Les équipes de projet détestent les comptes rendus hebdomadaires d'avancement des travaux parce que ceux-ci mettent trop vivement en lumière l'absence de leur progrès. ▲

Citations recueillies par Alain Coulon

Petits malheurs vécus

Ces citations d'une tonalité pessimiste tournent autour de la malchance. Sans doute, êtes-vous comme beaucoup d'entre nous (d'une parfaite mauvaise foi, entretenue par une solide paranoïa) plus sensible aux événements négatifs qu'aux événements heureux.

Lois générales

- S'il y a 50 % de chance de faire le mauvais choix, dans 80 % des cas vous ferez le mauvais choix.
- Tout ce qui est bien dans la vie est illégal, immoral, ou fait prendre du poids. Et si cela ne rentre pas dans l'une de ces catégories, c'est cancérigène à terme.
- Le problème quand on résiste à la tentation, c'est qu'elle ne se reproduit plus.
- Pour découvrir quelque chose il faut être en train de chercher autre chose.

Quelques exemples

- Tous les autobus qui passent dans l'autre sens disparaissent dans l'horizon et ne reviennent jamais.
- Le modèle que vous voulez n'est jamais celui qui est en vente. Si le vêtement vous plaît, ils ne l'ont pas dans votre taille. Seules les chaussures affreuses sont à votre pointure.
- Où que vous alliez à bicyclette, c'est toujours en montant et contre le vent.
- Tout fil coupé à la bonne longueur est trop court.
- Que vous soyez piéton ou automobiliste, le feu tricolore vous interdit toujours le passage.
- Si la localité que vous cherchez se trouve par chance sur la carte, elle sera soit à l'extrême bord, soit à cheval sur le pli.
- Conservez quelque chose suffisamment longtemps et vous finirez par le jeter. Jetez-le et vous en aurez besoin dès le lendemain.
- Le numéro de la revue qui vous manque est celui qui contient l'article que vous vouliez lire. Tous vos amis l'auront jeté, perdu ou prêté.
- Les documents vitaux démontrent leur aptitude à se déplacer spontanément depuis l'endroit où vous les avez laissés vers un endroit où vous ne pouvez pas les trouver.
- Un document confidentiel s'oublie sur la vitre du photocopieur.

- Un objet tombe toujours dans des endroits les plus inaccessibles ou sur des objets plus fragiles. Un petit objet qui tombe roule se cacher sous un objet plus grand.

- La saleté que vous cherchez à enlever est toujours de l'autre côté de la vitre.

- Quand on fait un faux numéro, ça ne sonne jamais occupé.

Loi de la proportionnalité

- L'intensité de la plainte d'un client mécontent est inversement proportionnelle au montant de son achat.

- Dans chaque assistance, ce sont les gens dont les fauteuils sont les plus éloignés de l'entrée qui arrivent toujours le plus tard.

- La probabilité qu'on vous voie est proportionnelle à la stupidité de votre action ; jouer au démineur augmente la probabilité d'arrivée de votre chef de service.

- L'utilité apparente d'un article devient inversement proportionnelle à son utilité réelle dès que vous l'avez payé.

- La distance jusqu'à la porte d'embarquement d'où part votre vol est inversement proportionnelle au temps qui vous reste avant le décollage.

- La densité locale de moustiques est inversement proportionnelle à la quantité restante de produit anti-moustique.

- Les opportunités se présentent toujours au moment le plus inopportun.

- Un raccourci est la plus longue distance d'un point à un autre.

Cas des sauvegardes

- La probabilité qu'une imprimante, un ordinateur ou un réseau plante augmente quand on se rapproche de la date limite de remise du rapport.

- Un ordinateur ne plante que le jour où son utilisateur néglige de faire une sauvegarde toutes les demi-heures.

- Toute sauvegarde automatique se fera au moment où vous ne vouliez pas qu'elle se fasse pour écraser un ancien fichier que vous aviez oublié de renommer.

- À quoi bon sauvegarder son travail puisque le plantage survient toujours juste avant la sauvegarde. Retarder les sauvegardes retarde le plantage.

- Lors de toute compression d'un ficher de plus de 1,4 Mo, il n'y a que 10 Ko occupés sur la dernière disquette.

- Si le disque dur vous laisse tomber, vous chercherez la disquette sur laquelle vous avez sauvegardé les fichiers importants. Vous ne les trouverez que sur la dernière disquette introduite dans le lecteur et ils seront illisibles.

Voyageurs avec bagages

- Quand votre avion est en retard, la correspondance est à l'heure. Quand votre avion est à l'heure, la correspondance est annulée.
- La première valise dont on aura besoin est celle qui est enfouie au fond de la soute.
- Votre bagage arrive toujours en dernier sur le tapis roulant.
- Votre place réservée est toujours à l'autre bout du train.
- Votre billet se trouve invariablement dans la dernière poche que vous fouillez.
- Quel que soit le nombre d'appareils pour composter c'est toujours celui que vous avez choisi qui ne marche pas. Au moment de composter, le billet est toujours dans le mauvais sens.
- Plus le bagage est volumineux, plus le train est bondé. Plus le bagage est lourd, plus le porte-bagages est haut et plus la place au sol est rare.
- La première chose dont vous avez besoin en ouvrant une valise se trouve toujours au fond.

L'appropriation de ces citations par nos lecteurs va créer une communauté de malchanceux. Nous serons tous persuadés que le sort est un génie malin qui nous en veut, au point de faire obstacle à nos actions.

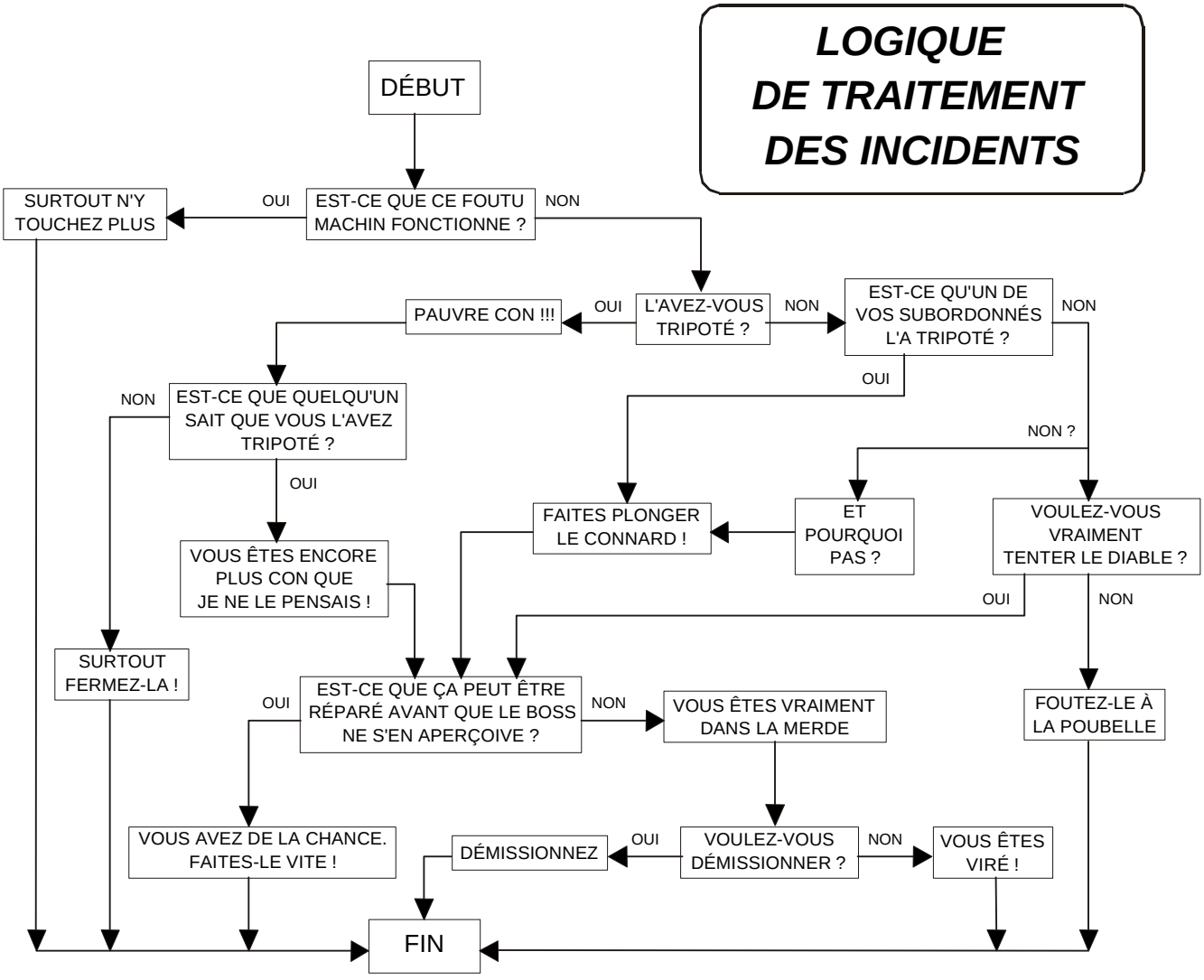
Il y a deux parades

Ne rien entreprendre, de peur que le mauvais génie ne vienne contrarier toutes nos initiatives, ce qui engendre à court terme l'ennui, l'incompétence et l'amertume.

L'autre parade consiste à jouer contre le mauvais génie en lui tendant des pièges. Dans un prochain numéro, nous vous dévoilerons cette méthode destinée à conjurer le mauvais sort.

Alain Coulon

L'incident...



Recueilli par Jean-Luc Blary