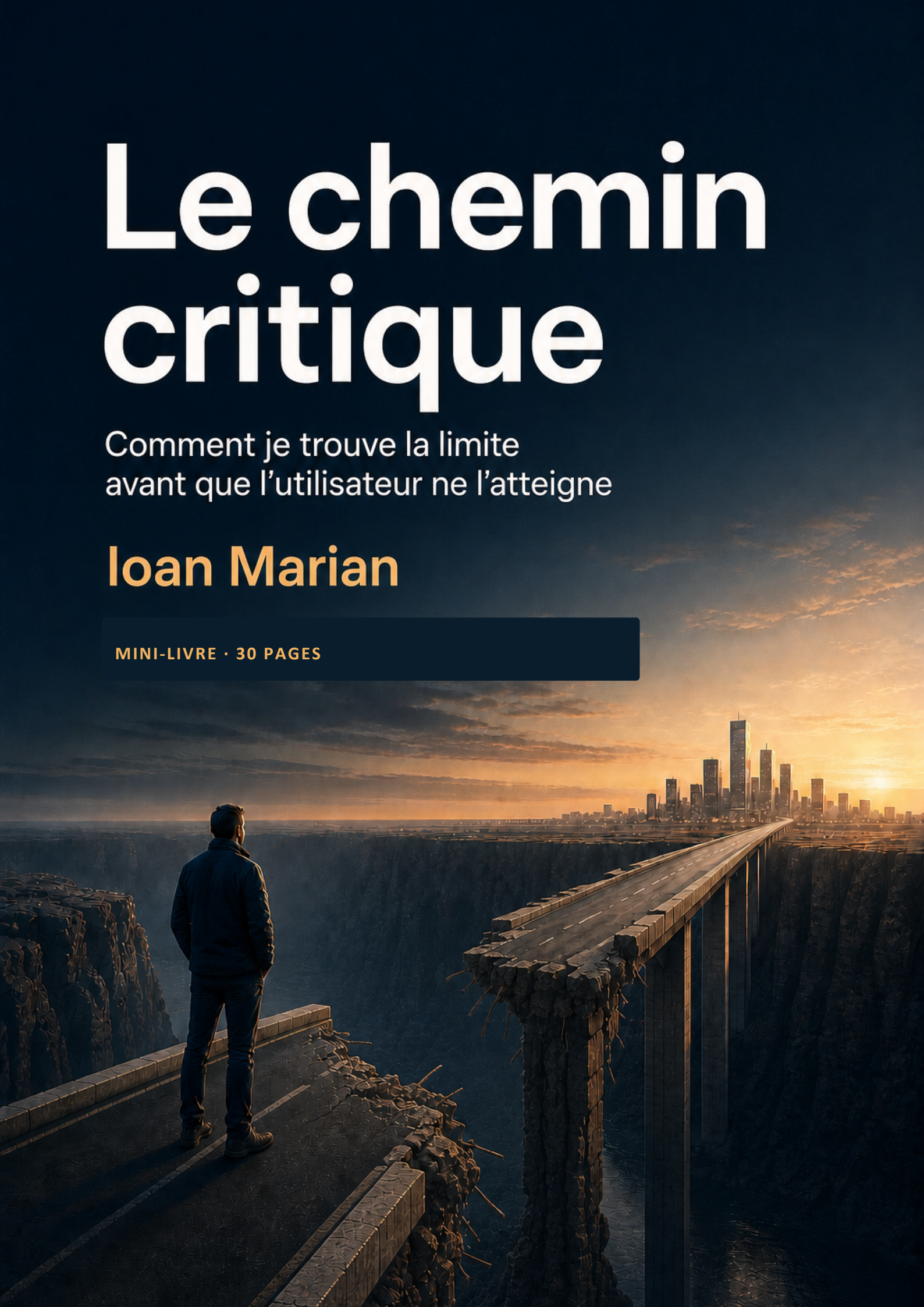
Le chemin critique

Comment je trouve la limite
avant que l'utilisateur ne l'atteigne

Ioan Marian

MINI-LIVRE · 30 PAGES



OUVERTURE

Je voulais voir jusqu’où elle pouvait aller

Une expérience menée chez moi, avant le premier client.

En 1998, je travaillais chez moi sur une application de comptabilité sous Access. Avant de la remettre à son premier client, je n’ai pas seulement vérifié qu’elle s’ouvrait, calculait et enregistrait. Je voulais voir ce qui se passerait lorsque les données augmenteraient.

J’ai augmenté progressivement le volume. Avec quelques milliers d’enregistrements, elle fonctionnait bien. Avec quelques centaines de milliers, elle perdait son rythme. Autour d’un million, elle ne répondait plus. C’est moi qui ai rencontré la limite, dans ma chambre, et non le client après la livraison.

J’avais commencé à mesurer dès 1994, sur les ordinateurs 286 et 386 de la faculté. L’expérience de 1998 m’a laissé une question : jusqu’où cela fonctionne-t-il et que se passe-t-il lorsque le volume augmente ? Le mini-livre peut être lu seul. Chaque leçon explique le problème, le test et le résultat.

POURQUOI

Je n’optimise pas pour obtenir un chiffre impressionnant. Je supprime le travail inutile afin que l’application reste rapide, facile à comprendre et facile à modifier lorsque les données et les besoins augmentent.

Un test me dit que l’application fonctionne. Un test de volume me dit jusqu’où elle peut aller.

LA CARTE DU MINI-LIVRE

De l’effet à la cause, puis de la solution à la continuité

Les 26 leçons suivent l’ordre dans lequel je travaille : d’abord, je rends le problème visible, puis je supprime le travail inutile et, enfin, je laisse une méthode que quelqu’un d’autre peut poursuivre.

01 Je comprends ce qui se passe

Les leçons du 01 au 09
Incidents, journaux, repères, volume et points de rupture.

02 Je supprime le travail inutile

Les leçons 10 à 21
Le chemin critique, ID-first, parallélisme, recherche et modèles.

03 Je construis la continuité

Les leçons 22 à 26
Préparation, progrès, évaluation, protection et transfert.

COMMENT LE LIRE Vous pouvez parcourir les 30 pages ou commencer par un problème proche de votre système. Chaque leçon contient suffisamment de contexte pour être comprise séparément.

LEÇON 01 | COMPRENDRE CE QUI SE PASSE

Une ligne de code a bloqué toute une plateforme

Il était midi. Je m'apprêtais à partir en pause lorsque le responsable est venu me voir, inquiet : « L'application intranet ne fonctionne plus. »

J'ai demandé calmement : « Laquelle ? » Je pensais qu'une page isolée s'était arrêtée temporairement. Je ne m'attendais pas à la réponse : « Plus aucune page ne fonctionne. »

La plateforme était utilisée chaque jour par de nombreuses personnes. Elle regroupait des documents et plusieurs applications internes. Elle fonctionnait depuis des années sans incident comparable.

Une course contre la montre a commencé. En une quinzaine de minutes, le composant qui provoquait le blocage a été isolé. Il avait été développé plusieurs mois auparavant. Le composant a été désactivé immédiatement et les utilisateurs ont pu reprendre leur activité.

Ce n'est qu'ensuite que j'ai ouvert le code source. Je connaissais le modèle que le composant devait respecter. Dès l'ouverture du code, la ligne qui ne suivait pas ce modèle a attiré mon attention en une vingtaine de secondes.

La lecture des données n'avait pas été conservée comme une étape séparée. Le code lisait une liste, puis chaque élément de la boucle déclenchait d'autres lectures.

Au début, le volume était faible. La page fonctionnait normalement et personne ne remarquait le problème. Puis le volume a grandi discrètement. Chaque lecture semblait courte, mais sa répétition pour chaque élément a complètement changé le temps total.

J'ai déplacé la lecture avant la boucle et réutilisé son résultat. Après les tests dans tous les environnements, le composant corrigé est revenu en production le jour même.

À ESSAYER Choisissez un incident déjà visible. Rétablissez d'abord le service ; reconstruisez ensuite la cause étape par étape.

IDÉE

La plateforme était déjà bloquée. Je n'avais plus à mesurer sa lenteur. Je devais trouver ce qui l'avait arrêtée.

LEÇON 02 | COMPRENDRE CE QUI SE PASSE

D’ici demain, il faut qu’elle soit quatre fois plus rapide

J’étais au milieu d’une discussion d’architecture lorsque mon responsable est entré. La demande concernait une page sur laquelle je n’avais jamais travaillé. Une autre équipe l’avait développée et j’en avais hérité sans contexte ni documentation.

À midi, la demande était claire : « Demain, la première page doit être quatre fois plus rapide. Et ce soir, je veux les étapes de mise en œuvre. »

La page ne produisait pas d’erreur. Elle fonctionnait, mais c’était la page d’accueil d’une plateforme SharePoint interne, très utilisée, et son chargement prenait plusieurs secondes.

Un seul fichier XSL - un modèle qui transformait les données en page visible par l’utilisateur - regroupait tous les cas et était appelé plusieurs fois pendant la construction de la même page.

Plusieurs possibilités existaient. Je pouvais réécrire entièrement l’affichage. Je pouvais conserver temporairement le résultat dans un cache : la page aurait semblé plus rapide, mais la cause serait restée. Ou je pouvais diviser le grand fichier et exécuter uniquement la partie nécessaire.

Je n’ai pas choisi selon ma préférence. En une dizaine de minutes, j’ai ajouté des temps dans le journal et mesuré séparément les composants. En une dizaine de minutes supplémentaires, j’ai déplacé le choix du cas en C# et divisé le fichier initial en plusieurs fichiers XSL plus petits. Chacun était appelé une seule fois, uniquement lorsque c’était nécessaire.

Je n’ai changé ni les règles ni les informations affichées. J’ai seulement changé la manière de répartir le travail. Après les tests dans tous les environnements, la solution a été mise en production le jour même.

Le résultat a dépassé l’objectif : la page est devenue presque six fois plus rapide.

La pression a disparu, mais la leçon est restée : le gain vient parfois simplement d’une bonne répartition du travail.

À ESSAYER Lorsque le temps est court, mesurez les composants et cherchez la plus petite modification qui supprime le plus de travail répété.

IDÉE **La pression ne m’oblige pas à tout changer. Elle m’oblige à choisir avec précision.**

LEÇON 03 | COMPRENDRE CE QUI SE PASSE

Nous n’avons plus les documents

On m’a demandé de transférer une grande archive de documents depuis une plateforme interne vers un autre espace de stockage. Elle contenait des procédures, des documents internes et des archives représentant des années de travail.

J’ai téléchargé les documents, je les ai organisés dans des dossiers et déposés à la destination indiquée. Tout semblait correct.

Quelque temps plus tard, j’ai appris que les documents n’étaient plus disponibles, ni à la source ni à la destination. La question était simple et difficile : existait-il encore une copie quelque part ?

Par chance, je n’avais pas encore supprimé les dossiers temporaires de l’ancien serveur. Cette copie conservée sans intention a permis de récupérer toute l’archive.

Je ne présente pas cet incident comme une bonne procédure. J’ai eu de la chance. Si j’avais supprimé les dossiers temporaires juste après le transfert, les documents n’auraient pas pu être récupérés.

Certains documents peuvent être recréés en quelques heures. D’autres ne pourront jamais l’être.

Je fais toujours une sauvegarde.

À ESSAYER

Pour un transfert important, vérifiez séparément la destination, la sauvegarde et la restauration avant de fermer la source.

IDÉE

Une sauvegarde qui n’a jamais été restaurée reste une promesse.

LEÇON 04 | COMPRENDRE CE QUI SE PASSE

Cherchez une limite que vous pouvez reproduire

Ne commencez pas par le système entier. Choisissez une seule opération et rendez son point de rupture visible.

Choisissez une lecture que vous pouvez répéter. Gardez les mêmes colonnes et multipliez le nombre de lignes par dix à chaque étape, jusqu'à ce que le temps dépasse le seuil accepté ou commence à croître plus vite que le volume. Revenez ensuite au même nombre de lignes et changez uniquement le nombre de colonnes lues. Ne modifiez pas les deux axes dans le même test.

Si vous apprenez maintenant	Si vous coordonnez	Si vous projettez
Une seule modification	Ne demandez pas seulement un chiffre	Deux axes

À ESSAYER Choisissez une lecture et augmentez seulement le nombre de lignes jusqu'à ce que la courbe change de forme.

LEÇON 05 | COMPRENDRE CE QUI SE PASSE

Le nom du fichier indique ce qui a démarré et à quel moment

J’utilise le modèle ClientList_AAAAMMJJ_HHMMSS_FFF.txt. Par exemple, ClientList_20260806_115522_012.txt indique que l’opération ClientList a démarré le 6 août 2026 à 11:55:22,012.

Chaque exécution reçoit son propre fichier. Ainsi, je ne mélange pas deux demandes ayant des volumes, des temps ou des résultats différents. Le nom m’indique rapidement l’opération et l’heure de départ. Le contenu montre les étapes dans l’ordre où elles se sont produites.

J’enregistre le début de chaque phase, sa durée, le temps cumulé depuis le démarrage, le nombre de lignes, de fichiers ou d’appels, ainsi que les valeurs métier nécessaires pour vérifier un calcul. À la fin, j’enregistre le résultat et l’erreur éventuelle. Je n’écris pas de données personnelles ni de contenu sensible inutile à l’analyse.

01	**ClientList**	Le nom de l’opération métier lancée.
02	**AAAAMMJJ**	L’année, le mois et le jour du démarrage.
03	**HHMMSS**	L’heure, la minute et la seconde du démarrage.
04	**FFF**	La fraction de seconde qui distingue deux exécutions démarrées presque au même moment.

À ESSAYER

Créez un nouveau fichier pour chaque exécution importante et donnez-lui un nom assez clair pour le retrouver sans ouvrir le code.

IDÉE

Un bon journal d’exécution ne dit pas seulement que l’opération est terminée. Il raconte toute l’histoire de l’exécution.

LEÇON 06 | COMPRENDRE CE QUI SE PASSE

J’observe plusieurs jours ou je lis l’historique déjà disponible

Une valeur isolée devient d’abord une question, pas une optimisation urgente.

Si le risque le permet, j’observe pendant quelques jours et je vérifie si l’écart se répète. Si je dispose déjà de deux semaines d’historique, je peux voir immédiatement si la tendance était déjà présente.

Pour chaque page, je conserve séparément l’historique des exécutions et je calcule p50, p95 et p99. Je calcule les mêmes valeurs pour l’ensemble de l’application, mais la valeur globale peut masquer la dégradation d’une seule page. p95 est le temps dans lequel 95 % des exécutions se sont terminées ; les 5 % restants ont duré plus longtemps.

Si je dispose de 1 000 exécutions comparables et que p95 vaut 0,3 seconde, environ 950 se sont terminées en 0,3 seconde au maximum et environ 50 ont été plus lentes. L’analyse est à la fois manuelle et automatique : les tests quotidiens et les rapports standardisés me montrent l’état de chaque page et son évolution dans le temps.

Aujourd’hui Une observation	**Quelques jours** Répétition	**Deux semaines** Historique
Périodiquement Intervention		

À ESSAYER Calculez le p95 pour toutes les exécutions et séparément pour chaque page. Vérifiez ensuite si l’écart se répète.

IDÉE

Je lis la valeur dans son contexte, je confirme la tendance et j’interviens avant qu’elle ne devienne un incident.

LEÇON 07 | COMPRENDRE CE QUI SE PASSE

L’algorithme, les données d’entrée, le tri et le résultat

Je ne mesurais pas encore SQL Server ni les millisecondes ; j’apprenais à rendre le problème vérifiable.

En COBOL, je partais de données d’entrée que je connaissais, j’exécutais le calcul et le tri, puis je vérifiais le rapport produit. Dans dBase IV et FoxPro, je gardais le même ordre : une entrée connue, des étapes explicables et un résultat vérifié.

Mon premier repère n’était pas un temps très court. C’était un cas que je pouvais répéter et dont je pouvais vérifier le résultat après chaque changement.

Entrée	Étapes	Sortie
Données contrôlées	Calcul et tri	Rapport vérifié

À ESSAYER Construisez un tableau simple avec l’opération, le volume, le temps et le résultat vérifié.

IDÉE

Mon premier repère n’était pas une milliseconde. C’était un résultat que je pouvais vérifier.

LEÇON 08 | COMPRENDRE CE QUI SE PASSE

Un million de lignes n’a pas un coût unique

Il s’agit d’un laboratoire SQL distinct ; il ne fait pas partie de la série C# automatisée du repository. Je veux observer séparément l’effet du nombre de lignes et celui du nombre de colonnes.

1. Je crée un modèle synthétique comportant un million de lignes et cinquante colonnes : un ID, dix-neuf valeurs numériques, dix dates, dix titres et dix descriptions.

2. Je sélectionne tout d’abord l’ID. Je répète le même test avec 1, 10, 100, 1 000, 10 000, 100 000 et 1 000 000 de lignes.

3. Je repars d’une seule ligne pour chaque groupe de 5, 10, 20 et 50 colonnes. Je ne prends pas simplement les premières colonnes physiques. Chaque groupe mélange des valeurs numériques, des dates, des titres et des descriptions.

4. Le groupe de 5 contient l’ID, deux valeurs numériques, une date et un titre. Le groupe de 10 ajoute d’autres valeurs numériques, d’autres dates, un titre et deux descriptions. Le groupe de 20 contient quatre descriptions. Celui de 50 contient les dix descriptions.

5. Les descriptions ont des longueurs différentes. Pour cette raison, dix colonnes peuvent déplacer beaucoup plus de données que cinq, et l’augmentation ne suit pas seulement le nombre de colonnes.

6. Le tableau montre les résultats pour le volume d’un million de lignes. Les volumes sont des estimations reproductibles de la charge synthétique. Les temps SQL dépendent du serveur, des index, du cache et du réseau et doivent être mesurés dans l’environnement dans lequel la base de données fonctionne.

SELECTION	VALEURS RETOURNÉES	VOLUME ESTIMÉ (Mo)
Uniquement ID	1 000 000	8,00
5 colonnes	5 000 000	77,00
10 colonnes	10 000 000	478,00
20 colonnes	20 000 000	2 820,00
50 colonnes	50 000 000	7 170,00

À ESSAYER

Créez une table de test avec des textes courts et longs. Comparez les mêmes lignes en sélectionnant 1, 5, 10, 20 et 50 colonnes.

IDÉE

Un million de lignes n’a pas un coût unique. La quantité de données portée par chaque ligne compte également.

LEÇON 09 | COMPRENDRE CE QUI SE PASSE

Le même travail utile, d’une instance à un milliard de nouvelles instances

Il s’agit d’un calcul indicatif, et non d’un benchmark inclus dans le manifeste public. Je conserve exactement un milliard de mises à jour et je montre comment les allocations changent lorsque l’instruction new est placée à l’intérieur de la répétition.

1. Je prends une petite classe dont chaque objet occupe 24 octets, soit la taille observée dans la démonstration publique avec dix millions d’objets.

2. J’exécute un milliard de mises à jour utiles dans chaque variante. Avec un seul objet, il reçoit toutes les mises à jour. Avec 1 000 objets, chacun en reçoit un million. Avec un milliard d’objets, chacun reçoit une seule mise à jour.

3. La colonne de mémoire est calculée : le nombre d’objets multiplié par 24 octets. Pour un milliard d’objets, il y a environ 24 Go d’allocations cumulatives.

4. Je ne publie pas ici de temps d’exécution. Ils doivent être mesurés sur l’ordinateur du lecteur, car le JIT, le processeur et le ramasse-miettes peuvent fortement modifier le résultat.

INSTANCES CRÉÉES	MISES À JOUR / INSTANCE	MÉMOIRE CALCULÉE (Mo)
1	1 000 000 000	0,000024
1 000	1 000 000	0,024000
10 000	100 000	0,240000
1 000 000	1 000	24,000000
1 000 000 000	1	24 000,000000

À ESSAYER

Gardez un milliard de mises à jour et changez seulement le nombre d’objets créés. Comparez le temps et la mémoire allouée.

IDÉE

Le nombre d’itérations de la boucle ne m’indique pas le nombre d’instances créées. C’est la position de l’instruction new qui le décide.

LEÇON 10 | RÉDUIRE LE TRAVAIL INUTILE

La première version lisait tout le jeu de données pour une seule page

L'exemple historique utilise des valeurs approximatives : 60 colonnes, 100 000 lignes et quelques dizaines de lignes affichées.

La première version renvoyait environ 60 colonnes pour quelque 100 000 lignes. Cela représentait près de 6 000 000 de valeurs pour une seule page. J'ai changé l'ordre des opérations. Je sélectionnais d'abord les ID, je conservais ceux de la page courante, puis seulement je lisais tous les champs de ces lignes.

Avec deux lectures, je renvoyais environ 100 000 identifiants et 3 000 valeurs pour la page en cours. Il y avait 103 000 valeurs au total contre 6 000 000. Avec trois lectures, à la page 6, j'avais un COUNT, environ 300 identifiants et 3 000 valeurs. Le total était 3 301. Une page très profonde nécessitait plus d'identifiants, et l'avantage devait être mesuré à nouveau.

Ces rapports décrivent les données renvoyées à l'application ; ils ne prétendent pas que le travail interne de SQL Server diminue automatiquement dans la même proportion. Le filtre, l'index et le tri déterminent ce que la base doit lire et ordonner. L'ID était le dernier critère de tri ; ainsi, l'égalité des autres champs ne pouvait pas modifier l'ordre des lignes d'une requête à l'autre.

6 000 000 Forme initiale	**103 000** Deux lectures	**3 301** Trois lectures, page 6

À ESSAYER Dessinez le parcours d'une page dans quatre cases : lecture, traitement, dépendances et affichage.

LEÇON 11 | RÉDUIRE LE TRAVAIL INUTILE

La même page, mais 7,65 Go, 4,38 Mo ou 0,38 Mo

Il s'agit d'un autre laboratoire SQL distinct, avec des lignes synthétiques plus larges que dans le test précédent. Je veux afficher les mêmes cinquante lignes d'une grande table. Je compare trois parcours et je vérifie que la page finale est identique.

1. Je crée un million de lignes. Chaque ligne contient cinquante colonnes : un ID, des nombres, des dates, des titres et des descriptions pouvant atteindre 800 caractères.

2. Dans le premier parcours, un SELECT - l'instruction SQL qui lit les données - renvoie les cinquante colonnes de toutes les lignes. L'application reçoit l'ensemble complet et ne conserve que les cinquante lignes de la page courante.

3. Dans le deuxième parcours, je sélectionne tous les ID triés. Je prends les ID de la page courante, puis j'exécute un autre SELECT pour lire toutes les colonnes uniquement pour ces cinquante ID.

4. Dans le troisième parcours, j'exécute COUNT pour obtenir le nombre total de lignes, je sélectionne seulement les ID nécessaires jusqu'à la page demandée et je lis toutes les colonnes uniquement pour les cinquante lignes affichées.

5. Je calcule la même valeur de contrôle à partir de tous les champs de la page. Les trois parcours renvoient la même page ; seul le travail nécessaire pour y parvenir change.

6. Ce laboratoire reproduit le parcours historique en trois lectures : COUNT, les ID jusqu'à la fin de la page et les lignes complètes de la page. Il n'utilise pas l'index étroit et Skip/Take du laboratoire suivant. C'est pourquoi la page 6 a des temps différents : les conditions diffèrent, pas le résultat.

VARIANTE	VALEURS RETOURNÉES	VOLUME (Mo)	TEMPS (secondes)
L'ensemble : 50 colonnes	50 000 000	7 650,00	21,00
TOUS les ID + page	1 002 500	4,38	0,63
COUNT + 300 ID + page 6	2 801	0,38	0,10
COUNT + 62 850 ID + page 1 257	65 351	0,63	0,15

À ESSAYER

Répétez l'expérience avec des lignes larges. Comparez l'ensemble complet, tous les ID et seulement les ID nécessaires jusqu'à la page demandée.

IDÉE

Je ne m'intéresse pas seulement au nombre de fois où une variante est plus rapide. Je veux savoir si le temps final reste faible lorsque le volume augmente.

LEÇON 12 | RÉDUIRE LE TRAVAIL INUTILE

Le nombre de lectures était une décision mesurée.

Je choisissais la variante qui lisait et transportait le moins de données.

Dans les livres et les exemples SQL que j'étudiais alors, la pagination était généralement présentée avec une seule requête. Je ne partais pas d'un modèle connu à deux ou trois lectures. J'y suis arrivé en testant séparément les ID ordonnés, les lignes de la page courante et COUNT - l'opération qui compte toutes les lignes correspondantes.

À trois lectures, j'exécute d'abord COUNT, qui compte le total. Ensuite, j'utilise TOP pour sélectionner les identifiants ordonnés nécessaires jusqu'à la fin de la page demandée. Dans cette liste, je garde les identifiants de la page en cours et, à la troisième lecture, je charge toutes les colonnes uniquement pour ces lignes.

Si un filtre WHERE difficile réduisait déjà le résultat à quelques milliers de lignes, deux lectures pouvaient être meilleures que trois. Si l'utilisateur avait besoin du total, je le mesurais séparément. Je ne transformais pas l'architecture en règle immuable. Je testais une, deux et trois lectures sur les mêmes données et je choisissais le coût total le plus faible.

01	**Je lis une fois**	Filtre, ordre, page et champs dans un plan efficace.
02	**Je lis deux fois**	Les identifiants, puis les objets ; l'ensemble intermédiaire est petit.
03	**Je lis trois fois**	COUNT, ID et objets sont des coûts justifiés.
04	**Je mesure**	Je choisis d'après les mesures, pas d'après le nombre de commandes.

À ESSAYER Mesurez séparément les variantes avec une, deux et trois lectures. Gardez celle qui déplace le moins de données.

IDÉE

Le nombre de requêtes était secondaire. L'objectif était de lire moins de données.

LEÇON 13 | RÉDUIRE LE TRAVAIL INUTILE

Neuf secondes sont devenues 0,17 seconde, mais le temps absolu raconte mieux l’histoire

Il s’agit d’un laboratoire SQL distinct, et non de la suite numérique du tableau précédent. Je reconstruis dix millions de lignes larges, avec une autre distribution synthétique et un autre plan SQL, puis je demande la même page par deux parcours.

1. Je crée une table synthétique de dix millions de lignes, cinquante colonnes et dix descriptions de 50 à 200 caractères.

2. Une page contient cinquante lignes. Je trie selon le champ choisi, puis selon l’ID, afin que la même demande produise toujours le même ordre.

3. Je crée un index étroit qui contient uniquement le champ de tri et l’ID.

4. Dans la variante habituellement utilisée, je sélectionne les cinquante colonnes et j’applique directement Skip/Take sur celles-ci.

5. Dans la variante ID-first, j’applique Skip/Take uniquement sur l’ID. Je sélectionne ensuite toutes les colonnes seulement pour les cinquante ID de la page.

6. Je vérifie les mêmes cinquante lignes, les mêmes 2 500 valeurs et la même valeur de contrôle.

7. Je mesure le temps complet et les lectures logiques. Une lecture logique est une page de données consultée par SQL Server, pas une ligne renvoyée à l’application.

8. Je répète chaque variante dans trois processus indépendants et je conserve le temps représentatif.

PAGE	LIGNES IGNORÉES	TOUTES LES COLONNES (secondes)	ID-FIRST COMPLET (s)	RAPPORT
1 257	62 800	0,1660	0,0040	≈ 40×
10 000	499 950	1,3800	0,0240	≈ 57×
50 000	2 499 950	6,5500	0,1200	≈ 55×
70 000	3 499 950	9,2000	0,1670	≈ 55×

À ESSAYER

Comparez Skip/Take sur toutes les colonnes avec Skip/Take uniquement sur l’ID, suivi de la lecture des lignes de la page courante.

IDÉE

Après un gain important, je vérifie le temps final et la marge restante.

LEÇON 14 | RÉDUIRE LE TRAVAIL INUTILE

Les mêmes 101 000 lignes : une seconde ou un tiers de seconde

Il s'agit d'un troisième laboratoire SQL distinct, avec vingt colonnes et une autre charge synthétique. Je veux vérifier si je peux réduire le temps de lecture sans modifier les données. Modulo 4 signifie que je répartis les lignes selon le reste de la division de l'ID par 4. Ce reste ne peut être que 0, 1, 2 ou 3 : j'obtiens donc quatre groupes sans chevauchement.

1. Je crée une petite table de 1 000 lignes et une grande table de 100 000 lignes. Chaque ligne contient vingt colonnes : des nombres, des dates, des titres et des descriptions. Au total, je lis 2 020 000 valeurs, soit environ 307,55 Mo.

2. Dans la première variante, je lis la petite table, puis la grande. Ce sont deux requêtes complètes exécutées l'une après l'autre.

3. Pour les deux autres variantes, la petite table reste la première tâche. Je divise la grande table en quatre groupes selon le reste de la division de l'ID par 4 : 0, 1, 2 et 3. Chaque ligne appartient à un seul groupe.

4. J'exécute d'abord les cinq tâches l'une après l'autre. Ce test montre le coût du découpage sans l'avantage du parallélisme.

5. J'exécute ensuite les mêmes cinq tâches en même temps, sur des connexions séparées.

6. Je réunis les lignes, je les trie par source et par ID, puis je calcule la même valeur à partir de tous les champs. Toutes les variantes renvoient les mêmes 101 000 lignes.

VARIANTE	TÂCHES	LIGNES	VOLUME (Mo)	TEMPS (secondes)
Deux lectures complètes, en série	2	101 000	307,55	1,02
Les mêmes cinq parties, en série	5	101 000	307,55	1,10
Les mêmes cinq parties, en parallèle	5	101 000	307,55	0,35

À ESSAYER

Divisez la grande table en quatre groupes avec modulo 4. Exécutez-les d'abord en série, puis en parallèle, et vérifiez le même résultat.

IDÉE

Modulo 4 me donne quatre groupes clairs. La mesure me dit ce que j'ai gagné. Dans ce test, la même lecture s'est terminée en environ un tiers du temps.

LEÇON 15 | RÉDUIRE LE TRAVAIL INUTILE

Les mêmes 300 000 résultats : 16,035200 secondes ou 0,015229 seconde

Je ne change que la voie d'accès à l'objet enfant approprié. Le calcul appliqué à chaque résultat et la valeur finale restent identiques.

1. Je crée quatre niveaux. Le premier contient 300 objets. Chaque objet a dix enfants ; les niveaux suivants contiennent donc 3 000, 30 000 et 300 000 objets.

2. Les 300 000 objets du dernier niveau sont les résultats utiles. J'effectue le même calcul pour chacun.

3. Dans la variante LINQ/Where, pour chaque parent, je parcours à nouveau toute la liste du niveau suivant et je vérifie quels objets lui appartiennent.

4. Dans la variante avec array, j'accède directement aux positions des enfants concernés. Je ne parcours plus toute la liste pour chaque parent.

5. LINQ/Where effectue environ 9,09 milliards de vérifications. L'array ne visite que les 333 000 objets nécessaires.

6. Les deux variantes produisent 300 000 résultats et la même somme finale. Je change uniquement le chemin d'accès à l'objet enfant.

VARIANTE	ÉLÉMENTS VÉRIFIÉS	TEMPS MOYEN (secondes)	RAPPORT	RÉSULTATS
LINQ/Where répété	9 090 900 000	16,035200	1×	300 000
Array avec accès direct	333 000	0,015229	≈ 1 053×	300 000

À ESSAYER

Construisez les mêmes relations avec LINQ/Where et avec un array de positions. Comptez les vérifications, pas seulement les secondes.

IDÉE

La recherche peut coûter plus cher que l'opération effectuée après avoir trouvé le résultat.

LEÇON 16 | RÉDUIRE LE TRAVAIL INUTILE

25 000 mots dans 100 000 documents : environ 41 heures ou 26 minutes

Je veux trouver 25 000 termes dans un grand document. Pour un seul terme, IndexOf est la solution simple et naturelle : il parcourt le texte et indique où le mot apparaît. Répétée pour 25 000 termes, cette solution parcourt à nouveau le document 25 000 fois.

1. Je crée un document synthétique d'environ 200 pages contenant 1 105 000 caractères.

2. Je crée 25 000 termes. Les deux méthodes doivent trouver exactement les mêmes 100 000 correspondances.

3. IndexOf est la solution directe : je cherche le premier terme dans tout le texte, puis le deuxième, et je continue jusqu'au 25 000e terme. Le document est parcouru de nouveau pour chaque terme.

4. Aho-Corasick est la méthode optimisée pour rechercher de nombreux termes en même temps. Je construis une seule structure pour tous les termes, puis je parcours le document une seule fois. Cette structure est construite en environ 0,006374 seconde et peut être réutilisée.

5. Le tableau projette le temps mesuré pour un document sur 100 000 documents similaires. Il ne prétend pas que j'ai exécuté le test pendant 41 heures.

MÉTHODE	UN DOCUMENT (secondes)	100 000 DOCUMENTS (secondes)	TEMPS EN UNITÉS FAMILIÈRES	RAPPORT
IndexOf répété	1,479386	147 938,60	Environ 41 h 6 min	1×
Aho-Corasick	0,015784	1 578,43	Environ 26 minutes et 18 secondes	≈ 94×

À ESSAYER

Construisez la carte Aho-Corasick une seule fois, puis comparez les mêmes correspondances avec des recherches IndexOf répétées.

IDÉE

Quand je cherche 25 000 choses, j'essaie de ne pas parcourir le même document 25 000 fois.

LEÇON 17 | RÉDUIRE LE TRAVAIL INUTILE

Quand plusieurs threads sur le même ordinateur ne résolvent pas le problème

Un thread est un chemin d'exécution suivi par l'ordinateur. Le point de départ est un traitement vidéo où un seul thread exécute les tâches l'une après l'autre sur un seul ordinateur. L'estimation est de douze jours.

1. Dans la variante initiale, un seul ordinateur traite les travaux l'un après l'autre sur un seul thread. Le temps estimé pour tout le lot est de douze jours.

2. J'essaie plusieurs threads sur le même ordinateur. Le gain n'est pas suffisant, car les tâches partagent le même processeur, la même mémoire et les mêmes limites locales.

3. Les tâches vidéo sont indépendantes. Un collègue propose Azure Batch, un service capable de démarrer plusieurs ordinateurs et d'envoyer une tâche à chacun.

4. Je divise le lot, je conserve un identifiant pour chaque tâche et je réunis les résultats à la fin. Une tâche échouée peut être relancée sans répéter tout le lot.

5. Le tableau montre la répartition mathématique idéale des douze jours. Dans une exécution réelle, j'ajoute le démarrage des ordinateurs, le transfert, l'attente, les reprises, les limites du service et le coût financier.

TÂCHES SIMULTANÉES	ORDINATEURS	LIMITE IDÉALE	CE QUE CELA SIGNIFIE
1	1	12 jours	point de départ
50	jusqu'à 50	5 heures et 46 minutes	partage idéal
100	jusqu'à 100	2 heures et 53 minutes	partage idéal
200	jusqu'à 200	1 h 26 min	partage idéal

À ESSAYER

Avant de distribuer le travail, vérifiez que les tâches sont indépendantes et que leurs résultats peuvent être réunis et relancés en toute sécurité.

IDÉE

Parfois, je n'ai pas à forcer le même ordinateur. Je dois diviser le problème en unités qui peuvent être exécutées sur des ordinateurs différents.

LEÇON 18 | RÉDUIRE LE TRAVAIL INUTILE

Un million de lignes et deux solutions trop coûteuses

Un script PowerShell devait créer un fichier d'environ un million de lignes. Le fichier était nécessaire ; le problème venait de la manière dont je le construisais.

La première variante écrivait immédiatement chaque ligne. Le résultat était correct, mais elle produisait environ un million d'opérations sur le fichier. Les ouvertures, écritures et synchronisations répétées devenaient le coût dominant.

La deuxième variante conservait toutes les lignes dans un seul texte. Dans PowerShell, le texte est immuable : à chaque ajout, un nouveau texte est créé, tout l'historique est recopié, puis la ligne suivante est ajoutée. Pour quatre unités, le travail vaut 1 + 2 + 3 + 4 = 10, alors que le résultat final ne contient que quatre unités.

J'avais besoin du même fichier final, mais sans un million d'écritures et sans reconstruire à plusieurs reprises un seul énorme texte.

Ligne par ligne	Un seul texte	Objectif
≈ 1 000 000 d'écritures	L'historique est copié	Le même fichier final

À ESSAYER

Construisez le même texte avec les quatre méthodes. Vérifiez le même hash, puis comparez le temps et l'allocation cumulée.

IDÉE

Je voulais le même fichier sans un million d'écritures et sans reconstruire à plusieurs reprises une seule chaîne immense.

LEÇON 19 | RÉDUIRE LE TRAVAIL INUTILE

Dix enregistrements disaient oui. Un million disait non

En février 2004, Web Forms et Web Controls constituaient la voie habituelle en ASP.NET. Les composants visuels et les événements permettaient de construire rapidement la première page.

J'ai construit un test en C#. Avec dix enregistrements, la page fonctionnait bien. Avec un million, elle se bloquait. Le grand volume m'a montré que je ne voulais pas continuer dans la même direction.

Je n'ai pas essayé la pagination ID-first dans la page Web Controls. En cinq minutes environ, j'ai décidé de construire un autre moteur autour de l'objet métier, du contexte et de l'action.

Le schéma était cli;list;detail;save;5. cli désignait l'objet Client. list était l'écran que je voulais atteindre. detail était l'écran d'où venait la demande. save était l'action. 5 était l'ID du client existant. Pour une création, j'utilisais l'ID -1.

La ligne n'était pas le moteur terminé. Elle en était la carte. Les mêmes étapes pouvaient être utilisées pour Client, Facture ou Utilisateur. Dans mon propre modèle, j'ai appliqué la pagination ID-first et le test avec un million de lignes est passé sous une seconde.

cli L'objet Client	**liste / détail** Destination et origine	**save / 5** L'action et l'ID

À ESSAYER Prenez une page existante et notez sur une seule feuille son objet, son contexte, son action et son identifiant.

IDÉE

Je n'ai pas construit le moteur en cinq minutes. J'ai choisi la direction dans laquelle j'allais le construire pendant les neuf mois suivants.

LEÇON 20 | RÉDUIRE LE TRAVAIL INUTILE

Une modification devait être faite une seule fois, pas dans chaque page

Les contrôles Web étaient des composants d'ASP.NET qui aidaient à construire rapidement une interface Web. La première page était facile à écrire ; la question était ce qui se passait quand le modèle atteignait des dizaines de pages.

Avec Web Controls, je pouvais rapidement construire une première page, mais chaque nouvelle page commençait avec beaucoup de travail. Si une règle commune changeait, je devais rechercher tous les endroits où elle avait été répétée. Mon moteur gardait les étapes communes en un seul endroit et laissait l'application la configuration et le comportement spécifiques.

Les configurations de l'application se trouvaient dans des fichiers XML - des fichiers texte structurés qui décrivaient les différences. Je ne surchargeais pas web.config, le fichier général de configuration de l'application, avec les détails de chaque objet. Je pouvais ainsi distinguer clairement ce qui appartenait au moteur, ce qui appartenait à l'application et ce qui devait être modifié lorsqu'un nouveau type de page apparaissait.

01	**3 pages**	La duplication est masquée ; le coût semble acceptable.
02	**40 pages**	La duplication se propage à de nombreux endroits.
03	**Le moteur**	Les étapes communes restent en un seul endroit.
04	**XML**	L'application déclare les différences sans réécrire le moteur.

À ESSAYER Choisissez trois classes semblables et notez ce qui peut être commun sans cacher leurs différences réelles.

IDÉE

Le problème était le nombre d'endroits où la même décision devait être répétée, pas le nombre de pages.

LEÇON 21 | RÉDUIRE LE TRAVAIL INUTILE

L’IA peut construire et multiplier très rapidement un bon modèle

Une fois que le premier exemple est clair, des classes similaires peuvent être créées beaucoup plus rapidement.

Je peux donner à l’IA une classe complète, les conventions, les tests et les différences du nouvel objet. Elle peut rapidement proposer Client, Facture ou Utilisateur sur la même structure. Elle peut compléter les validations, les journaux et les tests répétitifs, puis comparer les implémentations.

Le modèle me donne de la rapidité parce que je ne reprends pas les mêmes décisions à partir de zéro. Il me donne de la sécurité parce que le journal et les tests sont déjà dans l’exemple. Il me donne un résultat plus facile à prévoir parce que l’objet suivant suit des étapes connues. L’IA augmente ces avantages, mais elle peut aussi rapidement multiplier une erreur.

À ESSAYER	Avant de générer la deuxième classe, écrivez les règles que la première classe a déjà démontrées.

IDÉE	**L’IA multiplie le modèle. La décision sur ce qui mérite d’être multiplié reste la mienne.**

LEÇON 22 | CONSTRUIRE LA CONTINUITÉ

Trente minutes pour quelques personnes. Beaucoup plus pour tout le chantier

Les trente minutes n’ont pas disparu. Elles sont revenues sous la forme d’un travail plus difficile et plus coûteux.

Au moment de la décision, la situation paraît petite et personnelle : terminons-nous la pause ou nous levons-nous maintenant ? Mais la conséquence ne se mesure pas seulement au temps de chacun. Elle apparaît dans le travail de toute l’équipe, le béton perdu, le transport, le retard du chantier et la reprise du travail.

Cette idée est restée dans mon mode de travail. Je regarde d’abord la petite décision. Puis je regroupe tous ses effets sur l’équipe et le système. Ce qui semble bon marché en un seul endroit peut devenir très cher après avoir compté toutes les nouvelles étapes.

À ESSAYER

Choisissez une action courte qui bloque d’autres activités et notez le coût de son report d’une journée.

IDÉE

Les trente minutes économisées ont déplacé le travail vers une forme beaucoup plus difficile et plus coûteuse.

LEÇON 23 | CONSTRUIRE LA CONTINUITÉ

De 0,25 seconde à presque 40 secondes

Un petit coût, multiplié chaque semaine, ne reste pas petit.

Je pars de 0,25 seconde et j'ajoute cinq pour cent chaque semaine. Après 104 semaines, le résultat mathématique est d'environ 40 secondes. Dans une application réelle, la courbe ne sera pas aussi régulière. Certaines semaines, rien ne change ; d'autres connaissent un saut. L'exemple montre pourtant combien il est facile d'ignorer une perte qui paraît supportable isolément.

Le temps peut augmenter pour de nombreuses raisons. Il peut y avoir davantage de données, une nouvelle boucle, un appel externe ou un index disparu de la base de données. Si je ne regarde que le temps final, toutes ces causes se mélangent dans la même phrase : la page est lente.

À ESSAYER

Choisissez une page et conservez chaque semaine son p50, son p95, son volume et la version de l'application.

IDÉE

Cinq pour cent ne semblent pas inquiétants un vendredi. Répétés pendant deux ans, ils changent la nature du problème.

LEÇON 24 | CONSTRUIRE LA CONTINUITÉ

Le fait que cela fonctionne aujourd’hui n’est que le point de départ

Une démonstration confirme la fonction actuelle. L’historique montre comment l’application change.

Je pars de ce qui existe : utilisateurs, lignes, pages, règles, intégrations, temps et erreurs. Je recherche ensuite l’historique. À quelle vitesse les données ont-elles augmenté ? Combien d’exceptions sont apparues ? Combien d’endroits faut-il modifier pour une nouvelle règle ? Quelle part du temps appartient au système et quelle part aux dépendances externes ?

Je n’extrapole pas mécaniquement un mois. Je compare les périodes, les volumes et les types d’exécution. Je normalise le temps en fonction du volume traité avant de dire que le système est devenu plus lent. Si le nombre de lignes reste stable et que le temps augmente, je cherche une version plus lente, une nouvelle voie ou une dépendance qui a changé.

Aujourd’hui L’état actuel	**Historique** Changements au fil du temps	**Demain** Le scénario

À ESSAYER Prenez une fonction qui marche aujourd’hui et écrivez trois scénarios de croissance que vous pouvez tester.

IDÉE **Je compare le temps au volume et je construis les scénarios à partir de l’historique.**

LEÇON 25 | CONSTRUIRE LA CONTINUITÉ

La protection pouvait être oubliée

Le code de test pouvait modifier des données réelles, et son arrêt dépendait d'une étape manuelle.

J'ai utilisé cette pratique parce qu'elle donnait accès aux cas réels absents des données de test. J'exécutais peu d'exemples, choisis avec soin, et je gagnais du temps. Les exécutions réussies ont créé l'impression que la méthode était sous contrôle.

Un jour, fatigué, l'étape de protection a été oubliée. Le code a effectué des opérations là où il ne fallait pas, et les données réelles ont dû être récupérées. Le problème n'était pas que j'ai choisi consciemment de supprimer les données. Le problème était que le système ne refusait pas l'opération lorsque le contexte sortait de mon attention.

01	**Je gagne immédiatement**	Je teste rapidement des cas réels et variés.
02	**Je répète sans incident**	La pratique commence à paraître normale.
03	**J'oublie la protection**	L'attention peut céder un jour ordinaire.
04	**J'en paie le prix**	L'opération atteint les données réelles.

À ESSAYER Transformez une règle qui dépend de l'attention en un contrôle automatique capable de refuser l'exécution.

IDÉE

Si la sécurité dépend du fait que je m'en souvienne à chaque fois, je n'ai pas de protection. Je n'ai qu'un espoir.

LEÇON 26 | CONSTRUIRE LA CONTINUITÉ

Un système que quelqu'un d'autre peut poursuivre conserve la méthode, pas la personne

Je ne veux pas que le prochain problème commence par la question : « Où est l'auteur ? »

Le journal conserve l'exécution. Les tests conservent le résultat. Les repères conservent ce que nous savons de la limite. Le modèle conserve la séparation entre le commun et le spécifique. Les documents conservent les décisions. Les exemples montrent comment ajouter l'objet suivant. Aucun d'entre eux ne remplace les humains, mais leur permet de travailler sans dépendre de la mémoire d'une seule personne.

C'est l'aboutissement naturel du chemin critique : pas seulement une page plus rapide, mais une méthode de travail qui peut être observée, vérifiée et transmise. Si le système peut continuer sans moi, ma contribution ne disparaît pas. Elle devient une partie de la manière dont le système s'explique et évolue.

À ESSAYER

Demandez à un collègue d'expliquer le parcours d'une fonction sans votre aide et notez ce qui manque dans le modèle.

IDÉE

Le meilleur signe que j'ai laissé quelque chose de précieux est que le travail continue sans faire de moi un point de défaillance unique.

AVANT LE PROCHAIN CHANGEMENT

Dix questions qui gardent le chemin clair

Je n’ai pas besoin de toutes les réponses avant de commencer, j’ai besoin d’une question assez claire pour le prochain test.

01 **Quel résultat fonctionnel doit rester identique ?**	02 **Combien de temps cela prend-il maintenant, et quel est le volume ?**
03 **Combien de fois l’opération se répète-t-elle ?**	04 **Où le travail est-il multiplié ?**
05 **Quel est le premier point de rupture ?**	06 **Que montre mon journal en production ?**
07 **Quel travail puis-je éliminer, et pas seulement accélérer ?**	08 **Comment puis-je vérifier que le résultat reste correct ?**
09 **Que se passe-t-il si le volume est multiplié par dix ?**	10 **Quelqu’un d’autre peut-il continuer sans dépendre de ma mémoire ?**

LE CHEMIN CRITIQUE

Je commence par les choses simples, je les mesure, j’observe ce qui se passe lorsque les données et les répétitions augmentent, et je ne change que ce que je peux vérifier. Puis je laisse la méthode assez claire pour que quelqu’un d’autre puisse la poursuivre.

CODE ET RÉSULTATS PUBLICS

https://github.com/IoanMarian2001/DemoCriticalPath
Version exacte utilisée par le livre : book-v1.0.0