

Database system concepts

by Abraham Silberschatz, Henry F. Korth, S. Sudarshan · Language: FR

Summary by booksummaryai.com — <https://booksummaryai.com/country/fr/database-system-concepts-resume>

Summary

En 60 secondes

C'est le manuel universitaire de référence sur le fonctionnement réel des bases de données, construit autour d'une unique base de données universitaire regroupant enseignants, étudiants, cours et départements. Il commence par expliquer pourquoi les bases de données surpassent les fichiers, enseigne le modèle relationnel et SQL, des requêtes de base aux triggers et à la récursion, puis montre comment concevoir de bons schémas grâce aux diagrammes E-A et à la normalisation. La seconde moitié plonge sous le capot : stockage, index, optimisation de requêtes, transactions avec garanties ACID, contrôle de concurrence, récupération après panne et systèmes distribués. Au terme du livre, vous saurez lire un plan d'exécution de requête et comprendre pourquoi un virement bancaire ne s'effectue jamais à moitié.

Pourquoi le lire

Presque toutes les applications que vous utilisez reposent sur une base de données, et c'est ce livre qui a formé les personnes qui les ont construites. Il importe si vous êtes étudiant en informatique confronté à des examens, développeur qui écrit du SQL sans jamais avoir appris la théorie, ou ingénieur qui se prépare à des entretiens de conception de systèmes. Vous y gagnez un vocabulaire et des modèles mentaux — clés, ACID, sérialisabilité — qui rendent le comportement en production prévisible plutôt que mystérieux.

Les idées clés

Les fichiers pourrissent, pas les bases de données : plaidoyer pour un SGBD

Une base de données est un ensemble de données interreliées, et le SGBD est l'ensemble des programmes qui y accèdent. Le livre oppose ce modèle aux systèmes de traitement de fichiers, qui souffrent de redondance, d'incohérence, de difficultés d'accès aux données, d'isolement des données et de problèmes d'intégrité. Un SGBD fournit en revanche l'indépendance des données, un accès efficace, l'intégrité et la sécurité des données, l'accès concurrent et la récupération après panne. L'abstraction des données aux niveaux physique, logique et vue maintient le code applicatif stable même quand le stockage change. Les utilisateurs se répartissent entre utilisateurs naïfs, programmeurs d'applications et utilisateurs avertis, sans oublier les administrateurs de bases de données (DBA) qui dirigent l'ensemble.

Une base de données est un ensemble de données interreliées, et le SGBD est l'ensemble des programmes qui y accèdent. L'argument d'ouverture du livre est que les systèmes de traitement

de fichiers échouent de manière prévisible — redondance, incohérence, requêtes difficiles à écrire, isolement, problèmes d'intégrité — tandis qu'un SGBD vous offre l'indépendance des données, un accès efficace, l'intégrité, la sécurité, la concurrence et la récupération après panne. L'abstraction des données aux niveaux physique, logique et vue maintient le code applicatif stable même quand le stockage change. Les utilisateurs se répartissent entre utilisateurs naïfs, programmeurs d'applications et utilisateurs avertis, plus le DBA qui dirige l'ensemble.

À essayer: Trouvez dans votre vie un tableur où deux copies des mêmes données se contredisent, et transférez-le dans une table unique dotée d'une clé primaire.

Des tables à perte de vue : le modèle relationnel

Tout le livre pivote autour d'une petite table : la relation instructor, avec les attributs ID, name, dept_name et salary, où ID est la clé primaire. Les lignes sont des tuples, les colonnes des attributs, l'ordre des lignes n'a aucune signification, et chaque valeur est atomique — aucune liste cachée dans une cellule. Puis la contrainte qui rend le tout fiable : l'intégrité référentielle, illustrée par instructor.dept_name qui référence department.dept_name, de sorte qu'on ne peut affecter un enseignant à un département inexistant. Sur ce socle de tables simples, le livre construit l'algèbre relationnelle — sélection, projection, union, différence ensembliste, produit cartésien, renommage — la jointure étant définie à partir de ces primitives. Toute requête que vous écrierez jamais est une composition de ces six mouvements.

Une relation n'est qu'une table : les lignes sont des tuples, les colonnes des attributs, l'ordre des tuples est sans importance, et les valeurs d'attributs sont atomiques. La relation instructor — ID, name, dept_name, salary, avec ID comme clé primaire — est l'exemple fil rouge de tout le livre. Les clés forment une famille : superclé, clé candidate, clé primaire, clé étrangère, et elles garantissent l'intégrité, l'intégrité référentielle signifiant que instructor.dept_name doit correspondre à un department.dept_name réel. Les six opérations de base de l'algèbre relationnelle — sélection, projection, union, différence ensembliste, produit cartésien, renommage — définissent tout le reste, y compris la jointure.

À essayer: Ouvrez une table qui vous appartient et vérifiez : chaque colonne contient-elle une seule valeur atomique, et existe-t-il une colonne qui identifie de manière unique chaque ligne ?

SELECT-FROM-WHERE : SQL des premiers mots aux sous-requêtes

La structure de base d'une requête SQL est SELECT-FROM-WHERE : la clause SELECT choisit les colonnes, FROM nomme les relations, WHERE filtre les tuples avec des prédicats — par exemple, trouver les noms d'enseignants par département. La définition des données en SQL passe par CREATE TABLE pour spécifier les attributs, les types de données (tels que char, varchar, int, numeric) et les contraintes d'intégrité comme PRIMARY KEY, FOREIGN KEY et NOT NULL. Puis le langage s'approfondit : fonctions d'agrégation (avg, min, max, sum, count) avec group by et having, valeurs nulles, opérations ensemblistes (union, intersect, except), sous-requêtes imbriquées (in, some, all, exists), expressions de jointure (naturelle, interne, externe), vues,

clause with, grant et revoke, déclencheurs, et requêtes récursives avec with recursive.

La structure de base d'une requête SQL est SELECT-FROM-WHERE : la clause SELECT choisit les colonnes, FROM nomme les relations, WHERE filtre les tuples avec des prédicats. À partir de là, le livre superpose les agrégats (avg, min, max, sum, count) avec group by et having, les valeurs nulles, les opérations ensemblistes (union, intersect, except) et les sous-requêtes imbriquées utilisant in, some, all, exists. Le SQL intermédiaire ajoute les variantes de jointure — naturelle, interne, externe gauche/complète — les vues via CREATE VIEW, la clause with pour les sous-requêtes nommées temporaires, les transactions, les contraintes d'intégrité jusqu'à CREATE ASSERTION, et l'autorisation grant. Le SQL avancé introduit les fonctions et procédures stockées, les déclencheurs qui se déclenchent avant ou après des événements de ligne ou d'instruction, et les requêtes récursives avec with recursive.

À essayer: Écrivez cette semaine une requête utilisant group by avec having — calculez une moyenne par groupe, puis filtrez les groupes.

Le papier avant le code : le modèle E-R

Le modèle E-R mobilise entités, attributs, relations, ensembles d'entités et ensembles de relations pour concevoir les bases de données avant de passer aux tables, avec la base universitaire — enseignant, étudiant, cours, département — comme fil rouge. Ce sont les contraintes qui comptent : « takes » est une relation plusieurs-à-plusieurs entre étudiants et cours. Les ensembles d'entités faibles, comme « section » dépendant de « course », sont identifiés par un discriminant (sec_id) et possèdent une clé primaire (course_id, sec_id, semester, year). Les contraintes du modèle E-R englobent les cardinalités de correspondance (un-à-un, un-à-plusieurs, plusieurs-à-un, plusieurs-à-plusieurs), les contraintes de participation et les clés. Spécialisation, généralisation et agrégation traitent les hiérarchies et les relations entre relations.

Le modèle E-R permet de concevoir avec des entités, des attributs et des relations avant de passer aux tables, en prenant la base universitaire — enseignant, étudiant, cours, département — comme fil rouge. Ce sont les contraintes qui comptent : cardinalités de correspondance (un-à-un, un-à-plusieurs, plusieurs-à-plusieurs), contraintes de participation et clés, avec takes comme relation plusieurs-à-plusieurs entre étudiants et cours. Les ensembles d'entités faibles comme section dépendent de course et portent un discriminant (sec_id) ainsi qu'une clé primaire composite (course_id, sec_id, semester, year). Spécialisation, généralisation et agrégation traitent les hiérarchies et les relations entre relations.

À essayer: Esquissez les données de votre prochain projet sous forme d'entités et de relations, en marquant chaque relation un-à-un, un-à-plusieurs ou plusieurs-à-plusieurs avant d'écrire la moindre commande DDL.

Découpez la table jusqu'à ce que ça fasse mal : la normalisation

Les dépendances fonctionnelles (par exemple ID name, dept_name, salary dans la relation instructor) servent à décomposer les relations et à éviter la redondance grâce à la normalisation. La forme normale de Boyce-Codd exige que, pour toute dépendance fonctionnelle non triviale $X \rightarrow Y$, X soit une superclé ; la troisième forme normale assouplit cette règle pour tolérer une certaine redondance afin que les dépendances restent vérifiables. Les algorithmes de décomposition, appuyés sur la couverture canonique, convertissent les schémas non BCNF en relations BCNF — mais la décomposition en BCNF peut ne pas préserver les dépendances, tandis que la décomposition en 3NF les préserve. La conception devient un choix raisonné, non une habitude.

Les dépendances fonctionnelles — comme ID déterminant name, dept_name et salary dans la relation instructor — sont l'outil formel pour repérer la redondance et guider la décomposition. La BCNF exige que, dans toute dépendance non triviale $X \rightarrow Y$, X soit une superclé ; la 3NF assouplit délibérément cette règle, tolérant une certaine redondance pour que les dépendances restent vérifiables au sein de relations uniques. Les algorithmes de décomposition fondés sur la couverture canonique transforment les schémas non BCNF, et l'arbitrage est réel : la décomposition en BCNF peut ne pas préserver les dépendances, tandis que la décomposition en 3NF les préserve. La conception devient un choix raisonné, non une habitude.

À essayer: Prenez votre table la plus large et listez ce qui détermine quoi ; si une colonne non clé en détermine une autre non clé, vous avez trouvé une candidate à la décomposition.

Là où vivent les octets : disques et RAID

On passe de la logique à la physique : la hiérarchie de stockage comprend le cache, la mémoire centrale, la mémoire flash, le disque magnétique, le disque optique et la bande magnétique. Les disques magnétiques sont des objets physiques avec plateaux, pistes et secteurs, ce qui explique pourquoi le temps d'accès est dominé par le déplacement mécanique. Puis la question que tout administrateur finit par affronter — que se passe-t-il quand un disque meurt ? — rencontre le RAID : les niveaux 0 à 6, y compris des combinaisons comme RAID 1+0, fournissent une redondance répartie sur plusieurs disques. Chaque décision d'indexation et de tampon plus loin dans le livre est tarifiée à l'aune de cette hiérarchie.

La hiérarchie de stockage va du cache et de la mémoire centrale à la mémoire flash, au disque magnétique, au disque optique et à la bande magnétique, la volatilité et le coût par octet déterminant où les données ont leur place. Les disques magnétiques sont des objets physiques — plateaux, pistes, secteurs —, ce qui explique pourquoi le temps d'accès est dominé par le déplacement mécanique. Les niveaux RAID 0 à 6, y compris des combinaisons comme RAID 1+0, fournissent une redondance répartie sur plusieurs disques, transformant des taux de défaillance indépendants en un risque maîtrisé. Chaque décision d'indexation et de tampon plus loin dans le livre est tarifiée à l'aune de cette hiérarchie.

À essayer: Vérifiez quel niveau RAID votre stockage de production ou de sauvegarde utilise réellement, et s'il protège contre la défaillance de deux disques.

Arbres B et seaux : trouver les données vite

Le livre construit d'abord les index ordonnés — primaires et secondaires, denses et épars, empilés en index multiniveaux — puis sa vedette : l'arbre B+, une structure d'arbre équilibrée où tous les nœuds feuilles sont à la même profondeur, qui prend en charge efficacement les requêtes d'intervalle et les recherches par égalité. Le hachage prend l'autre route : le hachage statique utilise un nombre fixe de seaux et souffre quand les données grossissent, tandis que les schémas dynamiques comme le hachage extensible laissent la table de hachage grossir et rétrécir dynamiquement, avec le hachage linéaire comme alternative. Choisir entre eux, c'est choisir entre balayages d'intervalle et recherches ponctuelles.

Les index ordonnés se déclinent en plusieurs variantes — primaires et secondaires, denses et épars, multiniveaux — et l'arbre B+ est le cheval de bataille : un arbre équilibré dont tous les nœuds feuilles sont à la même profondeur, et c'est précisément pour cela qu'il sert aussi bien les recherches par égalité que les requêtes d'intervalle. Le hachage échange l'ordre contre la vitesse : le hachage statique utilise un nombre fixe de seaux, tandis que les schémas dynamiques comme le hachage extensible laissent la table de hachage grossir et rétrécir au gré des données, avec le hachage linéaire comme alternative. Choisir entre eux, c'est choisir entre balayages d'intervalle et recherches ponctuelles.

À essayer: Choisissez votre requête la plus lente et vérifiez si ses colonnes WHERE portent un index ; sinon, ajoutez-en un et chronométrez la différence.

Comment une requête s'exécute réellement : traitement et optimisation

Le traitement des requêtes comprend l'analyse et la traduction, l'optimisation et l'évaluation. Les algorithmes de sélection, de tri et de jointure — boucles imbriquées, boucles imbriquées par blocs, boucles imbriquées indexées, jointure par fusion, jointure par hachage — sont mesurés en coût d'E/S disque, et un mauvais choix peut coûter des ordres de grandeur. L'astuce de l'optimiseur, ce sont les règles d'équivalence qui transforment une expression d'algèbre relationnelle en une autre, plus l'optimisation fondée sur les coûts qui estime le coût des différents plans d'évaluation à l'aide de statistiques comme les histogrammes et la sélectivité — prédisant le prix de chaque plan avant d'en exécuter le moindre.

Le traitement des requêtes comporte trois étapes : analyse et traduction, optimisation et évaluation, les algorithmes de sélection, de tri et de jointure — boucles imbriquées, boucles imbriquées par blocs, boucles imbriquées indexées, jointure par fusion, jointure par hachage — étant jugés en coût d'E/S disque. L'optimisation est la partie intéressante : les règles d'équivalence permettent au système de réécrire une expression d'algèbre relationnelle sous une forme moins coûteuse, et l'optimisation fondée sur les coûts estime le prix des plans à partir de statistiques comme les histogrammes et la sélectivité. Le même SQL peut différer de plusieurs ordres de grandeur en temps d'exécution selon le plan que l'optimiseur retient.

À essayer: Cette semaine, lancez EXPLAIN (ou l'équivalent de votre base de données) sur une requête lente et trouvez la méthode de jointure qu'elle a choisie.

ACID : pourquoi un virement raté ne peut pas perdre votre argent

L'illustration canonique est un virement bancaire du compte A vers le compte B. Une transaction est une unité d'exécution de programme qui accède à divers éléments de données et les met éventuellement à jour, et elle doit satisfaire les propriétés ACID : atomicité (toutes les étapes ont lieu ou aucune), cohérence (les comptes restent équilibrés), isolation (aucune autre transaction ne voit l'état à moitié accompli) et durabilité (une fois confirmé, le virement survit à toute défaillance ultérieure). Le virement bancaire du compte A vers le compte B est l'illustration du livre, et c'est la même garantie que votre banque vous accorde chaque jour.

Une transaction est une unité d'exécution de programme qui accède à divers éléments de données et les met éventuellement à jour, et elle doit satisfaire ACID : atomicité, cohérence, isolation, durabilité. L'illustration canonique est un virement bancaire du compte A vers le compte B — deux mises à jour qui doivent réussir ou échouer ensemble, ne jamais laisser les comptes incohérents, ne jamais être visibles en cours de route par les autres et ne jamais s'évaporer après un crash. Chaque protocole de concurrence et chaque mécanisme de récupération des chapitres suivants existe pour délivrer ces quatre propriétés à grande vitesse.

À essayer: Enveloppez votre prochaine modification de données en plusieurs étapes dans une seule transaction — BEGIN, les deux mises à jour, COMMIT — et vérifiez qu'un échec annule les deux.

Verrous, horodatages et la danse de l'interblocage

Le contrôle de concurrence couvre la sérialisabilité (par conflit et par vue), la barre de correction pour l'exécution concurrente. Les protocoles à base de verrous comme le verrouillage en deux phases (2PL) avec verrous partagés et exclusifs garantissent la sérialisabilité, bien que la détention de verrous puisse mener à des interblocages. Le livre enseigne donc leur gestion : schémas de prévention (wait-die, wound-wait), détection via graphes d'attente (un cycle indique un interblocage) et résolution généralement par annulation d'une des transactions. Au-delà des verrous, les protocoles à base d'horodatage attribuent à chaque transaction un horodatage unique, et les protocoles à base de validation (optimistes) et le contrôle de concurrence multiversion (MVCC) sont également couverts.

La sérialisabilité—par conflit et par vue—est la barre de correction pour l'exécution concurrente, et le verrouillage en deux phases l'atteint avec des verrous partagés et exclusifs, bien que la détention de verrous dans les phases croissante et décroissante puisse produire des interblocages. La gestion des interblocages se divise en prévention (wait-die, wound-wait), détection via graphes d'attente où un cycle indique un interblocage, et résolution par annulation d'une transaction. Les alternatives échangent le verrouillage contre d'autres hypothèses : les protocoles à base d'horodatage donnent à chaque transaction un horodatage unique et

ordonnent le travail en conséquence, tandis que les protocoles à base de validation (optimistes) et le contrôle de concurrence multiversion maintiennent les lecteurs sans verrou.

À essayer: Vérifiez cette semaine les attentes de verrous ou interblocages en cours dans votre base de données—la plupart des systèmes les exposent—et identifiez quelles deux requêtes sont entrées en collision.

Le journal qui survit à tout : la récupération

La récupération à base de journal enregistre chaque mise à jour dans un journal sur stockage stable. La règle qui le rend fiable est la journalisation avec écriture anticipée (WAL) : les enregistrements du journal doivent être écrits sur le stockage stable avant la page de données correspondante. La modification différée et immédiate de la base de données définit les deux stratégies, et les points de contrôle limitent la quantité de journal à rejouer. L'algorithme de récupération ARIES utilise un numéro de séquence de journal (LSN) et trois phases : analyse (identifie les pages sales et les transactions actives), redo (réapplique toutes les mises à jour du journal) et undo (annule les transactions non validées à l'aide d'enregistrements de journal de compensation). La récupération après incident cesse d'être magique et devient un algorithme que vous pourriez implémenter.

La récupération à base de journal enregistre chaque mise à jour dans un journal sur stockage stable, avec la modification différée et immédiate de la base de données comme deux stratégies, les points de contrôle limitant la quantité de journal à rejouer, et la règle de journalisation avec écriture anticipée comme invariant : les enregistrements du journal atteignent le stockage stable avant la page de données correspondante. ARIES transforme le journal en une procédure de récupération précise utilisant des numéros de séquence de journal et trois phases—analyse pour trouver les pages sales et les transactions actives, redo pour réappliquer toutes les mises à jour, undo pour annuler les transactions non validées avec des enregistrements de journal de compensation. La récupération après incident cesse d'être magique et devient un algorithme que vous pourriez implémenter.

À essayer: Vérifiez que votre base de données a la journalisation de pages complètes et des points de contrôle récents activés—ces deux paramètres décident de la vitesse à laquelle vous récupérez.

Une seule machine ne suffit jamais : architectures, parallélisme, distribution

Les architectures de systèmes de base de données incluent les systèmes centralisés et client-serveur, avec la distinction entre serveurs d'applications et serveurs de bases de données, et le concept d'architecture à trois niveaux. Pour le débit brut, les architectures de bases de données parallèles incluent mémoire partagée, disque partagé et rien de partagé, mesurées par l'accélération et la mise à l'échelle, exploitant le parallélisme inter-requêtes et intra-requête. Quand les données s'étendent sur plusieurs sites, les bases de données distribuées utilisent la réplication de données, la fragmentation (horizontale et verticale) et le partitionnement

(round-robin, hachage, plage) ; le protocole de validation en deux phases (2PC) garantit l'atomicité, avec un coordinateur demandant à tous les participants de se préparer avant de valider, de sorte qu'une transaction distribuée est atomique partout ou nulle part.

Les architectures de systèmes de base de données vont du centralisé au client-serveur, distinguant les serveurs d'applications des serveurs de bases de données et ajoutant le modèle à trois niveaux. Les bases de données parallèles choisissent entre les conceptions à mémoire partagée, disque partagé et rien de partagé, mesurées par l'accélération et la mise à l'échelle, exploitant le parallélisme inter-requêtes et intra-requête. Les bases de données distribuées répartissent les données sur plusieurs sites via la réplication, la fragmentation horizontale et verticale, et le partitionnement par round-robin, hachage ou plage—tandis que le protocole de validation en deux phases, avec un coordinateur demandant à tous les participants de se préparer avant de valider, est ce qui maintient une transaction atomique à travers les machines.

À essayer: Esquissez quel niveau de votre système actuel détient la logique métier et lequel détient les données—s'ils sont fusionnés, vous avez trouvé votre goulot d'étranglement de montée en charge.

Les meilleures citations

- "Une base de données est un ensemble de données interreliées, et un système de gestion de base de données (SGBD) est un ensemble de programmes permettant d'accéder à ces données." - Abraham Silberschatz, Henry F. Korth, S. Sudarshan
- "la relation « instructor » possède les attributs ID, name, dept_name et salary, avec ID comme clé primaire" - Abraham Silberschatz, Henry F. Korth, S. Sudarshan
- "la construction « with recursive », avec un exemple retrouvant tous les cours qui sont des prérequis (directement ou indirectement) d'un cours donné" - Abraham Silberschatz, Henry F. Korth, S. Sudarshan
- "la règle du journalisation avec écriture préalable (WAL), qui exige que les enregistrements du journal soient écrits sur le stockage stable avant la page de données correspondante" - Abraham Silberschatz, Henry F. Korth, S. Sudarshan

Limites

C'est un manuel, et il s'en lit comme tel : patient, systématique et aride par moments, avec des exercices qui assurent une bonne part de l'enseignement. L'exemple filé de l'université est utile, mais peut sembler répétitif sur quinze cents pages de contenu. Les détails d'implémentation penchent vers les systèmes en vogue au moment de la rédaction, si bien que certaines précisions produits vieillissent plus vite que la théorie. Et le livre enseigne davantage les principes que le savoir-faire opérationnel — le réglage fin, la planification de capacité en conditions réelles et les patterns cloud-native sont traités trop brièvement. Complétez-le par une pratique concrète, sinon vous saurez ce qu'est un arbre B+ sans jamais en avoir réglé un.

En une phrase

Toute application fiable repose sur des tables, des clés, des transactions et des journaux — et ce livre vous enseigne ces quatre éléments depuis les fondations.

Pour qui

Quiconque écrit du SQL et soupçonne que la base de données fait quelque chose de magique en dessous — développeurs d'applications, ingénieurs data et étudiants en informatique qui veulent savoir pourquoi leur requête est lente et pourquoi un virement bancaire ne s'effectue jamais à moitié. Également les managers qui assistent aux réunions d'architecture en hochant la tête tout en souhaitant secrètement que quelqu'un explique les index, les transactions et la normalisation en mots simples.