



Performances et optimisation : optimiseur et plans abstraits

Adaptive Server® Enterprise

12.5.1

RÉF. DU DOCUMENT : DC38426-01-1251-01
DERNIÈRE MISE À JOUR : août 2003

Copyright © 1989-2004 par Sybase, Inc. Tous droits réservés.

Cette publication concerne le logiciel Sybase et toutes les versions ultérieures qui ne feraiant pas l'objet d'une réédition de la documentation ou de la publication de notes de mise à jour. Les informations contenues dans ce document pourront faire l'objet de modifications sans préavis. Le logiciel décrit est fourni sous contrat de licence et il ne peut être utilisé ou copié que conformément aux termes de ce contrat.

Pour commander des ouvrages supplémentaires ou acquérir des droits de reproduction, si vous habitez aux Etats-Unis ou au Canada, appelez notre Service Clients au (800) 685-8225, télécopie (617) 229-9845.

Les clients ne résidant ni aux Etats-Unis ni au Canada et qui disposent d'un contrat de licence pour les Etats-Unis peuvent joindre notre Service Clients par télécopie. Les autres clients doivent contacter leur filiale Sybase ou leur distributeur local. Les mises à jour du logiciel ne sont fournies qu'à des dates d'édition périodiques. Aucune partie de cette publication ne peut être reproduite, transmise ou traduite sous quelque forme et par quelque moyen que ce soit, électronique, mécanique, manuel, optique ou autre, sans l'autorisation écrite de Sybase, Inc.

Sybase, le logo Sybase, AccelaTrade, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-FORMS, APT-Translator, APT-Library, AvantGo, AvantGo Application Alerts, AvantGo Mobile Delivery, AvantGo Mobile Document Viewer, AvantGo Mobile Inspection, AvantGo Mobile Marketing Channel, AvantGo Mobile Pharma, AvantGo Mobile Sales, AvantGo Pylon, AvantGo Pylon Application Server, AvantGo Pylon Conduit, AvantGo Pylon PIM Server, AvantGo Pylon Pro, Backup Server, BizTracker, ClearConnect, Client-Library, Client Services, Convoy/DM, Copernicus, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRTP, eFulfillment Accelerator, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, EWA, Financial Fusion, Financial Fusion Server, Gateway Manager, GlobalFIX, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, Mail Anywhere Studio, MainframeConnect, Maintenance Express, Manage Anywhere Studio, M-Business Channel, M-Business Network, M-Business Server, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, My AvantGo, My AvantGo Media Channel, My AvantGo Mobile Marketing, MySupport, Net-Gateway, Net-Library, New Era of Networks, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC Net Library, PocketBuilder, Pocket PowerBuilder, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Rapport, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RW-DisplayLib, S-Designor, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, STEP, SupportNow, S.W.I.F.T. Message Format Libraries, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, TradeForce, Transact-SQL, Translation Toolkit, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server et XP Server sont des marques déposées de Sybase, Inc.

Unicode et le logo Unicode sont des marques déposées d'Unicode, Inc.

Tous les autres noms de produit, société ou marque apparaissant dans ce document sont des marques ou marques déposées de leurs propriétaires respectifs.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568-9702, Etats-Unis d'Amérique

Table des matières

Préface	xv	
CHAPITRE 1	Introduction au guide Performances et optimisation	1
CHAPITRE 2	Présentation de l'optimiseur	3
	Définition	4
	Etapes du traitement d'une requête	4
	Fonctionnement de l'optimiseur	5
	Importance de la taille des objets pour l'optimisation de la requête.....	6
	Optimisation des requêtes	7
	Tables sous-jacentes SQL et optimisation	8
	Facteurs examinés pour l'optimisation.....	9
	Ajout de clauses optimisables durant la phase de prétraitement	10
	Conversion des clauses en arguments de recherche équivalents	11
	Conversion d'expressions en arguments de recherche	11
	Fermeture transitive des arguments de recherche.....	12
	Fermeture transitive sur les jointures	13
	Transformation et factorisation de prédictats	14
	Instructions pour créer des arguments de recherche.....	16
	Arguments de recherche et utilisation des index.....	17
	Syntaxe des arguments de recherche.....	18
	Utilisation des statistiques pour les arguments de recherche.....	20
	Utilisation des statistiques sur plusieurs arguments de recherche.....	23
	Valeurs par défaut des arguments de recherche	24
	Arguments de recherche utilisant des variables et des paramètres.....	24
	Syntaxe et traitement des jointures	25
	Traitement des jointures	25
	Statistiques de jointure non disponibles	26
	Valeurs de densité et jointures	26

Plusieurs colonnes de jointures.....	27
Arguments de recherche et jointures sur une table.....	27
Types de données incompatibles et optimisation des requêtes	28
Présentation de la hiérarchie des types de données et des problèmes d'index.....	29
Types de données pour les paramètres et les variables utilisés comme arguments de recherche.....	32
Types de données compatibles pour les colonnes jointes	34
Suggestions concernant les types de données et les comparaisons	35
Conversion imposée sur l'autre côté de la jointure	36
Fractionnement des procédures stockées pour une meilleure estimation du coût	37
Unités principales d'estimation des coûts	38
 CHAPITRE 3	
Outils d'optimisation avancés.....	39
Techniques d'optimisation spéciales.....	39
Spécification des choix de l'optimiseur	40
Spécification de l'ordre des tables dans les jointures	41
Risques liés à l'utilisation de forceplan.....	42
Avant d'utiliser forceplan	43
Spécification du nombre de tables prises en compte par l'optimiseur	43
Spécification d'un index pour une requête	45
Risques	46
Opérations précédant la spécification d'un index.....	47
Spécification de la taille des E/S dans une requête	47
Type d'index et E/S étendues	49
Omission de la spécification de prefetch.....	49
set prefetch on.....	50
Spécification de la stratégie de caches.....	51
Dans les instructions select, delete et update	52
Contrôle des E/S étendues et des stratégies de cache	53
Informations sur les stratégies de cache	53
Service de journaux asynchrones	54
Description de l'architecture du cache de journaux utilisateur	55
ALS.....	56
Utilisation d'ALS	57
Procédures système modifiées	57
Activation et désactivation des jointures par fusion	58
Activation et désactivation de la fermeture transitive de jointures ..	58
Suggestion d'un degré de parallélisation pour une requête.....	60
Exemples de clauses parallel au niveau requête	61

	Optimisation de la concurrence d'accès pour les petites tables	62
	Modification du plan de verrouillage	63
CHAPITRE 4	Outils d'optimisation de la requête	65
	Présentation	65
	Interaction entre les outils	67
	Utilisation de showplan avec noexec	67
	noexec et statistics io	68
	Outils et traitement des requêtes	68
CHAPITRE 5	Méthodes d'accès et évaluation des coûts de requête sur une seule table	69
	Coût du balayage de table	71
	Coût du balayage d'une table verrouillée au niveau de toutes les pages	72
	Coût du balayage d'une table verrouillée au niveau des pages de données seulement.....	72
	De la ligne à la page	75
	Incidence des taux de clusterisation sur les estimations d'E/S étendues	75
	Evaluation du coût d'accès à un index	78
	Requête ne renvoyant qu'une seule ligne	78
	Requête renvoyant plusieurs lignes	78
	Requêtes à intervalles avec des index couvrants	81
	Requêtes à intervalles avec des index non couvrants	82
	Evaluation du coût des requêtes utilisant la clause order by	86
	Sous-ensemble ordonné et tris	87
	Ordre des clés et tris	88
	Evaluation des opérations de tri	90
	Tables APL avec index clusterisés.....	90
	Opérations de tri avec un index couvrant la requête.....	92
	Tris et index non couvrants	93
	Méthodes d'accès et évaluation du coût des clauses or et in	94
	Syntaxe or	94
	Conversion de in (values_list) en clause or.....	94
	Méthodes de traitement des clauses or	95
	Optimisation des agrégats	99
	Utilisation conjointe des agrégats max et min	101
	Mise à jour des opérations	102
	Mises à jour directes	102
	Mises à jour différées	105
	Insertion différée dans les index.....	107
	Restrictions concernant les modes de mise à jour avec des jointures.....	109

CHAPITRE 6

Optimisation des mises à jour	111
Utilisation de sp_sysmon dans l'optimisation des mises à jour	113
Méthodes d'accès et coût des requêtes pour des jointures et des sous-requêtes	115
Estimation et optimisation des jointures	115
Traitement	116
Jointures et densité des index	116
Jointures et incompatibilité des types de données	117
Permutations de jointures	118
Jointures à boucles imbriquées	121
Formules de calcul	123
Définition des tables internes et externes	123
Autojointure	124
Méthodes d'accès et estimation du coût des jointures tri-fusion	126
Déroulement d'une jointure par fusion complète	128
Déroulement d'une jointure par fusion à droite ou à gauche	129
Déroulement d'une jointure par tri	130
Exemple de jointures mixtes	130
Estimation du coût des jointures par fusion	132
Estimation du coût d'une jointure par fusion complète avec des valeurs uniques	133
Exemple : tables APL avec index clusterisés	134
Estimation du coût d'une jointure par fusion complète avec des valeurs en double	134
Evaluation du coût des tris	136
Impossibilité d'utiliser des jointures par fusion	137
Utilisation des processus de travail	138
Recommandations pour améliorer les performances des fusions	138
Activation et désactivation des jointures par fusion	140
Au niveau du serveur	140
Au niveau session	140
Stratégie de redéfinition d'index	141
Optimisation des sous-requêtes	142
Mise à plat des sous-requêtes in, any et exists	142
Mise à plat des sous-requêtes d'expression	148
Matérialisation des résultats de sous-requête	148
Sous-requête introduite par une clause and	151
Sous-requête introduite par une clause or	151

	Mise en mémoire cache des résultats de la sous-requête	152
	Optimisation des sous-requêtes.....	153
	Clauses or et unions dans les jointures	153
CHAPITRE 7	Traitement des requêtes parallèles	155
	Types de requête pouvant bénéficier du traitement parallèle	156
	Modèle de processus de travail d'Adaptive Server	158
	Exécution de requêtes parallèles	160
	Retour des résultats des requêtes parallèles	161
	Types d'accès parallèle aux données	162
	Balayage de table reposant sur un hachage.....	163
	Balayage reposant sur les partitions	164
	Balayage d'index reposant sur un hachage	164
	Traitement parallèle de deux tables dans le cadre d'une jointure	165
	Messages showplan.....	167
	Contrôle du degré de parallélisation	167
	Paramètres de configuration pour le contrôle de la parallélisation	168
	Utilisation des options set pour contrôler la parallélisation d'une session.....	172
	Contrôle de la parallélisation pour une requête.....	173
	Disponibilité des processus de travail et exécution des requêtes.....	174
	Autres paramètres de configuration pour le traitement en parallèle	174
	Commandes associées aux tables partitionnées.....	175
	Equilibre ressources/performances.....	177
	Ressources CPU.....	177
	Ressources disque et E/S	178
	Exemple d'optimisation : saturation de la CPU et des E/S....	178
	Recommandations pour la configuration des requêtes parallèles	179
	Recommandations relatives au matériel	179
	Objectifs de performances et recommandations matérielles	180
	Exemples d'optimisation des requêtes parallèles	181
	Recommandations pour le partitionnement et le degré de parallélisation.....	182
	Expériences avec des sous-ensembles de données	184
	Incidence au niveau du système	185
	Problèmes de verrouillage.....	185
	Problèmes de devices	186
	Conséquences du cache des procédures	186

	Cas dans lesquels les résultats des requêtes parallèles peuvent être différents.....	186
	Requêtes utilisant set rowcount	187
	Requêtes définissant des variables locales	188
	Homogénéisation des résultats	188
CHAPITRE 8	Optimisation des requêtes parallèles	189
	Concept d'optimisation des requêtes parallèles.....	190
	Optimisation du temps de réponse par rapport à la charge de travail globale.....	190
	Conditions d'exécution de l'optimisation	191
	Coûts d'overhead	191
	Facteurs non pris en compte	192
	Méthodes d'accès parallèles.....	192
	Balayage d'une partition en parallèle	193
	Balayage de partitions d'index clusterisés en parallèle (table APL)	196
	Balayage de table en parallèle avec hachage.....	198
	Balayage d'index en parallèle avec hachage	200
	Balayages en parallèle reposant sur des intervalles	202
	Méthodes d'accès en parallèle supplémentaires	204
	Récapitulatif des méthodes d'accès parallèles	205
	Sélection des méthodes d'accès parallèles	206
	Degré de parallélisation des requêtes parallèles	207
	Limite supérieure	207
	Degré optimisé	208
	Jointures à boucles imbriquées.....	210
	Exemples.....	213
	Ajustements lors de l'exécution des processus de travail	216
	Exemples de requête parallèle.....	216
	Balayages effectués sur une seule table.....	217
	Jointures sur plusieurs tables.....	219
	Sous-requêtes	222
	Requêtes nécessitant des tables de travail	223
	Requêtes union	223
	Requêtes avec agrégats	223
	Instructions select into	224
	Ajustements lors de l'exécution des processus de travail	224
	Ajustement d'un plan d'exécution de requête avec Adaptive Server	225
	Incidents des ajustements lors de l'exécution	226
	Identification et gestion des ajustements lors de l'exécution.	226
	Réduction de la probabilité d'ajustements lors de l'exécution.....	227
	Vérification des ajustements avec sp_sysmon.....	228

	Identification des problèmes liés aux performances	
	des requêtes parallèles	229
	Impossibilité d'exécuter la requête en parallèle	229
	Performances en parallèle peu satisfaisantes.....	230
	Assistance technique	230
	Limites d'utilisation des ressources dans les requêtes	
	parallèles	231
CHAPITRE 9	Tri parallèle.....	233
	Commandes pouvant bénéficier du tri parallèle	233
	Présentation des conditions et des ressources.....	234
	Présentation de la méthode de tri parallèle	236
	Création d'un plan de répartition	238
	Partitionnement dynamique des intervalles.....	238
	Tri des intervalles	239
	Fusion des résultats	239
	Configuration des ressources pour le tri parallèle	240
	Processus de travail requis pour le tri parallèle.....	240
	Processus de travail requis pour le tri de la requête select...	244
	Caches, buffers de tri et tris parallèles	245
	Conditions requises au niveau du disque.....	253
	Considérations de reprise	255
	Outils permettant de contrôler et d'optimiser les tris	256
	Utilisation de la commande set sort_resources on.....	256
	Utilisation de sp_msmon pour optimiser la création d'index.....	261
	Utilisation du tri parallèle pour accélérer la commande create index	262
CHAPITRE 10	Optimisation de la prélecture asynchrone	263
	Amélioration des performances à l'aide de la prélecture asynchrone	263
	Amélioration des performances des requêtes grâce à la prélecture des pages	265
	Mécanismes de contrôle de la prélecture dans un environnement multi-utilisateur.....	265
	Ensemble de pages préalablement lues lors d'une reprise...	266
	Ensemble de pages préalablement lues lors de balayages séquentiels	267
	Ensemble de pages préalablement lues lors de l'accès à un index non clusterisé	268
	Ensemble de pages préalablement lues et vérifications dbcc	268
	Tailles maximale et minimale d'un ensemble de pages préalablement lues	269

Désactivation automatique de la prélecture	271
Saturation des zones.....	271
Surcharges du système d'E/S	272
Lectures inutiles	272
Optimisation des objectifs de la prélecture asynchrone	274
Commandes de configuration	275
Autres fonctions d'optimisation des performances	
d'Adaptive Server	276
E/S étendues.....	276
Balayages de lecture-élimination (MRU).....	277
Balayages parallèles et E/S étendues.....	278
Paramètres spéciaux pour les limites de la prélecture	
asynchrone	279
Limites de reprise	279
Limites de dbcc	280
Activités de maintenance pour des performances	
de prélecture élevées	280
Elimination d'anomalies dans les tables sans index	281
Elimination d'anomalies dans les tables avec	
index clusterisés	281
Elimination d'anomalies dans les index non clusterisés.....	281
Analyse des performances et prélecture asynchrone	281
 CHAPITRE 11	
Bases de données temporaires multiples	283
Présentation	283
Après la création d'une base de données temporaire.....	287
Utilisation de sp_tempdb.....	288
Liaison à des bases de données temporaires.....	289
Liaison d'une session	290
Bases de données temporaires multiples dans le système	290
Modifications des tables système.....	290
Variable globale @@tempdbid	291
Fonction tempdb_id()	292
Troncature du journal	292
Annulation et restauration	292
Suppression d'une base de données temporaire.....	293
Alter database	294
Caractéristiques de la mise en mémoire cache	295
Traitement des procédures stockées	296
Optimisation de l'écriture de tempdb.....	297
Considérations en matière de haute disponibilité.....	297
Sauvegarde et chargement des bases de données	
temporaires.....	299
Procédure stockée sp_doption.....	300
Configuration du nombre de bases de données ouvertes....	300

	Procédures modifiées.....	300
	Commandes dbcc modifiées et nouvelles commandes dbcc	301
	Autres modifications	302
	Remarques concernant l'installation	303
	Taille et configuration de bases de données	
	temporaires pour des applications.....	303
	Tables temporaires partageables.....	304
	Mises à jour des procédures stockées créées	
	par l'utilisateur	304
	Retour à une version antérieure.....	305
CHAPITRE 12	Performances de tempdb.....	307
	Incidence de la gestion de tempdb sur les performances	307
	Principales solutions pour améliorer les performances	
	de tempdb.....	308
	Types et utilisations de tables temporaires	308
	Tables véritablement temporaires	309
	Tables utilisateur normales	309
	Tables de travail	310
	Allocation initiale de tempdb	310
	Taille de tempdb.....	311
	Positionnement de tempdb	312
	Libération du device master de segments de tempdb	313
	Utilisation de disques pour améliorer les performances	
	des requêtes parallèles	314
	Liaison de tempdb à son propre cache de données	314
	Commandes pour lier le cache.....	315
	Tables temporaires et verrouillage	315
	Réduction de l'activité de journalisation dans tempdb	316
	Utilisation de select into.....	316
	Utilisation de lignes plus courtes	316
	Optimisation des tables temporaires	317
	Création d'index sur les tables temporaires	319
	Création de procédures imbriquées avec des	
	tables temporaires	319
	Fractionnement des utilisations de tempdb	
	en plusieurs procédures	320
CHAPITRE 13	Curseurs et performances	321
	Définition	321
	Programmation à orientation ensembliste ou séquentielle....	322
	Exemple	323
	Ressources nécessaires à chaque étape	324
	Utilisation de la mémoire et exécution des curseurs	327

	Modes de curseur	327
	Utilisation d'index et conditions requises pour les curseurs.....	328
	Tables verrouillées au niveau de toutes les pages	328
	Tables verrouillées au niveau des pages de données seulement	329
	Comparaison des performances réalisées avec et sans curseur	330
	Exemple de procédure stockée sans curseur	330
	Exemple de procédure stockée avec curseur	331
	Comparaison des performances avec curseur et sans curseur	333
	Verrouillage à l'aide de curseurs en lecture seule	334
	Niveaux d'isolement et curseurs	335
	Tables sans index partitionnées et curseurs.....	336
	Conseils d'optimisation des curseurs	336
	Optimisation des sélections de curseur à l'aide d'un curseur.....	337
	Utilisation d'union au lieu des clauses or ou des listes in.....	337
	Déclaration de l'intention du curseur	338
	Spécification des noms de colonne dans la clause for update	338
	Utilisation de set cursor rows	339
	Curseurs ouverts lors des validations et des annulations	340
	Ouverture de plusieurs curseurs sur une seule connexion ...	340
CHAPITRE 14	Présentation des plans abstraits	341
	Définition	341
	Gestion des plans abstraits.....	343
	Relation entre le texte des requêtes et les plans d'exécution de requête	343
	Limites des options pour influer sur les plans d'exécution de requête	344
	Plans complets et plans partiels.....	344
	Création d'un plan partiel	346
	Groupes de plans abstraits	346
	Association des plans abstraits avec les requêtes.....	347
CHAPITRE 15	Guide des plans abstraits pour les requêtes	349
	Introduction	349
	Langage des plans abstraits	350
	Identification des tables.....	353
	Identification des index.....	354
	Spécification de l'ordre de jointure	354
	Spécification du type de jointure	359

	Spécification des plans partiels et des indications	360
	Création de plans abstraits pour des sous-requêtes.....	362
	Plans abstraits pour des vues matérialisées	369
	Plans abstraits de requêtes contenant des agrégats	369
	Spécification de la stratégie de reformatage	372
	Limites de la stratégie OR	373
	Cas où l'opérateur store n'est pas spécifié	373
	Conseils pour écrire des plans abstraits	373
	Comparaison des plans « avant » et « après ».....	374
	Conséquences de l'activation du mode de capture au niveau du serveur	375
	Temps et espace requis pour copier des plans.....	376
	Plans abstraits pour des procédures stockées	377
	Procédures et appartenance des plans.....	377
	Optimisation et procédures avec des chemins d'exécution variables	378
	Requêtes ad hoc et plans abstraits.....	378
CHAPITRE 16	Création et utilisation des plans abstraits.....	381
	Utilisation des commandes set pour capturer et associer des plans	381
	Activation du mode capture de plans avec set plan dump	382
	Association de requêtes à des plans enregistrés.....	383
	Utilisation du mode remplacement durant la capture de plans	383
	Utilisation simultanée des modes dump, load et replace	385
	Option set plan exists check	386
	Utilisation d'autres options set avec les plans abstraits	387
	Utilisation de showplan.....	387
	Utilisation de noeexec	388
	Utilisation de forceplan	388
	Modes capture et association de plans au niveau du serveur	389
	Création de plans avec SQL	389
	Utilisation de create plan	390
	Utilisation de la clause plan.....	391
CHAPITRE 17	Gestion des plans abstraits avec des procédures système... ..	393
	Procédures système pour gérer les plans abstraits	393
	Gestion d'un groupe de plans abstraits.....	394
	Création d'un groupe.....	394
	Suppression d'un groupe	395
	Informations relatives à un groupe	395
	Modification du nom d'un groupe	398
	Recherche de plans abstraits.....	398

Gestion des plans abstraits individuels	399
Affichage d'un plan.....	399
Copie d'un plan dans un autre groupe	400
Suppression d'un plan abstrait individuel.....	401
Comparaison de deux plans abstraits.....	401
Changement d'un plan existant.....	402
Gestion de tous les plans d'un groupe	403
Copie de tous les plans d'un groupe	403
Comparaison entre tous les plans d'un groupe.....	404
Suppression de tous les plans abstraits d'un groupe.....	407
Importation et exportation de groupes de plans.....	407
Exportation de plans vers une table utilisateur.....	407
Importation de plans depuis une table utilisateur	408
CHAPITRE 18	
Référence de langage des plans abstraits	409
Mots-clés.....	409
Opérandes	409
Tables sous-jacentes de plan abstrait.....	410
Schéma des exemples	410
g_join.....	411
hints.....	414
i_scan.....	415
in	417
lru	419
m_g_join.....	420
mru	422
nested	422
nl_g_join.....	424
parallel.....	426
plan	427
prefetch	429
prop	430
scan.....	431
store	433
subq	434
t_scan.....	437
table	438
union	440
view	441
work_t.....	442
Index	445

Préface

A qui s'adresse ce manuel ?

Ce manuel s'adresse aux administrateurs et aux concepteurs de bases de données, aux développeurs et aux administrateurs système.

Remarque Vous pouvez utiliser votre propre base de données pour tester les modifications et les requêtes. Prenez un cliché de la base de données en question et configuez-la sur une machine de test.

Comment utiliser ce manuel

Ce manuel vous permet d'optimiser, d'améliorer les performances d'Adaptive Server et de résoudre les éventuels problèmes.

Le [Chapitre 1, « Introduction au guide Performances et optimisation »](#), décrit les grandes lignes de ce manuel et des autres manuels de la série Performances et optimisation d'Adaptive Server.

Le [Chapitre 2, « Présentation de l'optimiseur »](#), détaille le processus d'optimisation des requêtes et explique comment sont utilisées les statistiques en vue de rechercher les arguments et les jointures pour les requêtes.

Le [Chapitre 3, « Outils d'optimisation avancés »](#), décrit les outils avancés pour l'optimisation des performances des requêtes.

Le [Chapitre 4, « Outils d'optimisation de la requête »](#), présente les outils d'optimisation des requêtes et une description de leur mode d'interaction.

Le [Chapitre 5, « Méthodes d'accès et évaluation des coûts de requête sur une seule table »](#), décrit la manière dont Adaptive Server accède aux tables des requêtes qui ne se réfèrent qu'à une seule table et comment il évalue les coûts associés aux différentes méthodes d'accès.

Le [Chapitre 6, « Méthodes d'accès et coût des requêtes pour des jointures et des sous-requêtes »](#), explique comment Adaptive Server accède aux tables pendant les jointures et les sous-requêtes et comment il détermine les coûts.

Le [Chapitre 7, « Traitement des requêtes parallèles »](#), présente les concepts et les ressources nécessaires pour le traitement des requêtes parallèles.

Documentation annexe

Le [Chapitre 8, « Optimisation des requêtes parallèles »](#), traite en détail de l'optimisation des requêtes parallèles.

Le [Chapitre 9, « Tri parallèle »](#), traite de l'utilisation du tri parallèle pour les requêtes et la création d'index.

Le [Chapitre 10, « Optimisation de la prélecture asynchrone »](#), explique comment l'option de prélecture asynchrone améliore les performances des requêtes qui exécutent des opérations d'E/S disque d'envergure.

Le [Chapitre 11, « Bases de données temporaires multiples »](#) explique comment Adaptive Server vous permet de créer plusieurs bases de données temporaires, que vous pouvez ensuite utiliser pour créer des objets temporaires, tels que des tables temporaires privées et des tables de travail.

Le [Chapitre 12, « Performances de tempdb »](#), souligne l'importance de la base de données temporaire tempdb et propose des conseils pour améliorer ses performances.

Le [Chapitre 13, « Curseurs et performances »](#), aborde les questions de performances liées aux curseurs.

- Les autres manuels de la série Performances et optimisation sont les suivants :

- *Performances et optimisation : concepts de base*
- *Performances et optimisation : verrouillage*
- *Performances et optimisation : contrôle et analyse*

- Les Notes de mise à jour pour votre plate-forme contiennent les informations de dernière minute qui ne figurent pas dans les manuels.

Une version plus récente des Notes de mise à jour peut être disponible sur le Web. Pour vérifier si des informations importantes sur le produit ou le document ont été ajoutées après la commercialisation du CD-ROM, consultez le site Sybase Technical Library.

- Le *Guide d'installation* pour votre plate-forme décrit les procédures d'installation, de mise à niveau et de configuration pour tous les produits Adaptive Server et Sybase associés.
- Le manuel *Configuration d'Adaptive Server Enterprise* pour votre plate-forme fournit des instructions pour effectuer des tâches spécifiques de configuration d'Adaptive Server.

- Le manuel *Nouvelles fonctionnalités d'Adaptive Server Enterprise* décrit les nouvelles caractéristiques de la version 12.5 d'Adaptive Server, les modifications apportées au système pour leur prise en charge et les modifications susceptibles d'avoir des conséquences sur les applications existantes.
- Le *Guide de l'utilisateur Transact-SQL* présente Transact-SQL, version enrichie du langage de base de données relationnelle de Sybase. Ce guide sert de document de référence pour les utilisateurs qui découvrent les systèmes de gestion de bases de données. Il décrit également les bases de données exemple pubs2 et pubs3.
- Le *Guide d'administration système* fournit des informations détaillées sur l'administration des serveurs et des bases de données. Il contient des instructions relatives à la gestion des ressources physiques, à la sécurité, aux bases de données système et utilisateur, ainsi qu'à la définition des paramètres de conversion de caractères, de la langue et de l'ordre de tri.
- Le *Manuel de référence* fournit des informations détaillées sur l'ensemble des commandes, fonctions, procédures et types de données Transact-SQL. Il propose également la liste des mots réservés de Transact-SQL et des définitions de tables système.
- Le guide *Utilitaires* présente les utilitaires d'Adaptive Server, tels que isql et bcp, qui sont exécutés au niveau du système d'exploitation.
- Le *Guide de référence rapide* fournit une liste complète des noms et syntaxes pour les commandes, les fonctions, les procédures système, les procédures système étendues, les types de données et les utilitaires dans un format poche. Disponible uniquement en version papier.
- Le *Diagramme des tables système* est un poster qui illustre les tables système selon le modèle entité/relation. Disponible uniquement en version papier.
- Le manuel *Error Messages and Troubleshooting Guide* explique comment résoudre les messages d'erreur les plus fréquents et décrit les solutions aux problèmes liés au système auxquels les utilisateurs sont le plus souvent confrontés.
- Le *Guide de l'utilisateur de Component Integration Services* explique l'utilisation de la fonction Adaptive Server Component Integration Services permettant d'établir des connexions à des bases de données distantes Sybase et non Sybase.

-
- Le manuel *Java dans Adaptive Server Enterprise* décrit l'installation et l'utilisation des classes Java dans la base de données d'Adaptive Server comme types de données, fonctions et procédures stockées.
 - Le manuel *XML Services dans Adaptive Server Enterprise* décrit le processeur XML natif et la prise en charge XML Java de Sybase, introduit XML dans la base de données et présente les fonctions de requête et de mappage qui composent XML Services.
 - Le manuel *Utilisation de Sybase Failover en environnement haute disponibilité* fournit les instructions d'utilisation du mode Failover de Sybase pour configurer un Adaptive Server comme serveur compagnon dans un environnement haute disponibilité.
 - Le *Guide de l'utilisateur de Job Scheduler* décrit les instructions d'installation et de configuration de Job Scheduler ainsi que la création et la planification de tâches sur un Adaptive Server local ou distant à l'aide de la ligne de commande ou d'une interface utilisateur graphique (GUI).
 - Le manuel *Utilisation des fonctionnalités DTM* traite de la configuration et de l'utilisation des fonctionnalités DTM d'Adaptive Server ainsi que de la résolution des éventuels problèmes dans les environnements de traitement des transactions distribuées.
 - Le *Guide de l'utilisateur d'EJB Server* explique comment utiliser EJB Server pour déployer et exécuter Enterprise JavaBeans dans Adaptive Server.
 - Le guide *XA Interface Integration Guide for CICS, Encina, and TUXEDO* fournit des instructions sur l'utilisation de l'interface DTM XA de Sybase avec les gestionnaires de transactions X/Open XA.
 - Le *Glossaire* définit les termes techniques utilisés dans la documentation Adaptive Server.
 - Le document *Sybase jConnect for JDBC Programmer's Reference* décrit le produit jConnect for JDBC et explique comment l'utiliser pour accéder aux données stockées dans des systèmes de gestion de bases de données relationnelles.
 - Le *Guide de l'utilisateur de Full-Text Search Specialty Data Store* explique comment utiliser la fonction Full-Text Search avec Verity afin d'effectuer des recherches dans les données d'Adaptive Server Enterprise.
 - Le *Guide de l'utilisateur d'Historical Server* décrit comment utiliser Historical Server pour obtenir des statistiques sur les performances de SQL Server et Adaptive Server.

- Le *Guide de l'utilisateur de Monitor Server* explique comment utiliser Monitor Server afin d'obtenir des statistiques de performances de SQL Server et Adaptive Server.
- Le guide *Monitor Client Library Programmer's Guide* explique comment écrire des applications Monitor Client Library accédant aux données de performances d'Adaptive Server.

Autres sources d'information

Utilisez le CD-ROM Sybase Technical Library ainsi que le site Web Technical Library Product Manuals pour obtenir davantage d'informations sur les produits :

- Le CD-ROM Getting Started, qui accompagne votre logiciel, propose les notes de mise à jour et les guides d'installation au format PDF, ainsi que d'autres documents ou des informations de dernière minute qui n'apparaissent pas sur le CD-ROM Technical Library. Pour lire ou imprimer les documents figurant sur le CD-ROM Getting Started, vous avez besoin du logiciel Acrobat Reader d'Adobe (téléchargeable gratuitement sur le site Web d'Adobe, accessible au moyen du lien indiqué sur le CD-ROM).
- Le Sybooks français contient la documentation traduite. Le CD-ROM Technical Library contient la documentation anglaise. Ces CD-ROM sont fournis avec le logiciel. Le lecteur DynaText, également présent sur le CD-ROM, permet d'accéder aux informations techniques relatives aux produits, dans un format facile à utiliser.

Pour plus d'informations sur l'installation et le démarrage de la Technical Library, reportez-vous au *Technical Library Installation Guide*.

- Le site Web Technical Library Product Manuals constitue la version HTML du CD-ROM Technical Library, accessible à l'aide d'un navigateur Web traditionnel. Outre les manuels relatifs aux produits, vous y trouverez des liens vers les sites EBFs/Updates, Technical Documents, Case Management, Solved Cases, des forums et Sybase Developer Network.

Pour accéder au site Web Technical Library Product Manuals, sélectionnez Product Manual à l'adresse <http://www.sybase.com/support/manuals/>.

Certifications Sybase sur le Web

La documentation technique du site Web Sybase est fréquemment mise à jour.

❖ **Recherche des informations les plus récentes sur les certifications des produits**

- 1 Cliquez sur Technical Documents à l'adresse <http://www.sybase.com/support/techdocs/>.
- 2 Sélectionnez Products dans la barre de navigation située à gauche.
- 3 Sélectionnez le nom d'un produit dans la liste de produits, puis cliquez sur Go.
- 4 Sélectionnez le filtre Certification Report, spécifiez une période et cliquez sur Go.
- 5 Cliquez sur un titre dans Certification Report pour visualiser le rapport.

❖ **Création d'un affichage personnalisé du site Web de Sybase (y compris des pages d'assistance)**

Il faut configurer un profil MySybase. MySybase est un service gratuit qui vous permet de créer une vue personnalisée des pages Web de Sybase.

- 1 Cliquez sur Technical Documents à l'adresse <http://www.sybase.com/support/techdocs/>.
- 2 Cliquez sur MySybase et créez un profil MySybase.

EBF et mises à jour des logiciels de Sybase

❖ **Recherche des informations les plus récentes sur les EBF et les mises à jour des logiciels**

- 1 Cliquez sur la page Sybase Support à l'adresse <http://www.sybase.com/support>.
- 2 Choisissez EBFs/Updates. Saisissez votre nom d'utilisateur et votre mot de passe si vous y êtes invité (pour les comptes Web existants) ou créez un nouveau compte (service gratuit).
- 3 Sélectionnez un produit.
- 4 Spécifiez une période et cliquez sur Go.
- 5 Cliquez sur l'icône Info pour afficher le rapport EBF/Update ou cliquez sur la description du produit pour télécharger le logiciel.

Conventions	Cette section décrit les conventions utilisées dans ce manuel.
Présentation des instructions SQL	SQL est un langage à structure non imposée. Il n'existe aucune règle quant au nombre de mots qui peuvent être placés sur une ligne ou à l'emplacement des fins de ligne. Cependant, pour une meilleure lisibilité, toutes les instructions syntaxiques et tous les exemples de ce manuel sont présentés de sorte que chaque clause d'une instruction commence sur une nouvelle ligne. Les clauses composées de plusieurs parties s'étendent sur les lignes suivantes, qui apparaissent en retrait.
Conventions typographiques et syntaxiques	Les conventions typographiques et syntaxiques utilisées dans ce manuel sont répertoriées dans le tableau 1.0 :

Tableau 1 : Conventions typographiques et syntaxiques de ce manuel

Elément	Exemple
Les noms de commandes, d'options de commande, d'utilitaires, de drapeaux d'utilitaire et autres mots-clés apparaissent en gras .	select sp_configure
Les noms de bases de données, les types de données, les noms de fichiers et les chemins d'accès apparaissent dans la police sans serif.	base de données master
Les variables (termes se substituant aux valeurs à insérer) apparaissent en <i>italique</i> .	select <i>nom_colonne</i> from <i>nom_table</i> where <i>conditions_recherche</i>
Les parenthèses font partie de la commande et doivent être saisies.	compute agrégat_ligne (nom_colonne)
Les accolades indiquent que vous devez choisir au moins une des options énumérées. N'entrez pas d'accolade dans l'instruction.	{comptant, chèque, crédit}
Les crochets indiquent que le choix d'une ou plusieurs des options entre crochets est facultatif. N'entrez pas de crochet dans votre instruction.	[anchois]
La barre verticale indique que vous ne pouvez choisir qu'une seule des options énumérées.	{comptant chèque crédit}

Elément	Exemple
La virgule vous permet de choisir autant d'options que vous le souhaitez, à condition de les séparer par une virgule dans la commande.	[fromage, avocat, crème]
Les points de suspension (...) signifient que vous pouvez <i>répéter</i> le dernier élément autant de fois que nécessaire.	acheter article = prix [comptant chèque crédit] [, article = prix [comptant chèque crédit]]...

Vous devez acheter au moins un article et indiquer son prix. Vous pouvez choisir un mode de paiement : l'un des éléments entre crochets. Vous pouvez également décider d'acheter des articles supplémentaires : autant d'articles que vous le souhaitez. Pour chaque article acheté, indiquez son nom, son prix et (éventuellement) un mode de paiement.

- Les instructions syntaxiques (affichage de la syntaxe et de toutes les options d'une commande) apparaissent comme suit :

```
sp_dropdevice [ nom_device ]
```

ou, dans le cas d'une commande comportant davantage d'options :

```
select nom_colonne
      from nom_table
        where conditions_recherche
```

Dans les instructions syntaxiques, les mots-clés (commandes) figurent en caractères standard et les identificateurs figurent en minuscules : caractères standard pour les mots-clés, en italique pour les mots fournis par l'utilisateur.

- Les exemples de résultats apparaissent comme suit :

```
0736 New Age Books Boston MA
0877 Binnet & Hardley Washington DC
1389 Algodata Infosystems Berkeley CA
```

Format des lettres

Dans ce manuel, la plupart des exemples sont en minuscules. Cependant, vous pouvez ignorer la distinction entre majuscules et minuscules lorsque vous entrez des mots-clés Transact-SQL. Par exemple, SELECT, Select et select sont équivalents. Notez que la distinction majuscules/minuscules faite par Adaptive Server pour les objets de base de données, tels que des noms de table, dépend de l'ordre de tri défini dans Adaptive Server. Pour les jeux de caractères codés sur un octet, la distinction majuscules/minuscules peut être modifiée en reconfigurant l'ordre de tri d'Adaptive Server.

Pour de plus amples informations, reportez-vous au *Guide d'administration système*.

Expressions

Les instructions syntaxiques Adaptive Server utilisent les types d'expression suivants :

Tableau 2 : Types d'expression utilisés dans les instructions syntaxiques

Syntaxe	Définition
<i>expression</i>	Peut inclure des constantes, des littéraux, des fonctions, des identificateurs de colonne, des variables ou des paramètres.
<i>expression logique</i>	Expression renvoyant une valeur TRUE (vrai), FALSE (faux) ou UNKNOWN (inconnu).
<i>expression constante</i>	Expression renvoyant toujours la même valeur, telle que « 5+3 » ou « ABCDE ».
<i>expression_flottante</i>	Toute expression à virgule flottante ou expression convertie implicitement en une valeur flottante.
<i>expression_entier</i>	Toute expression entière ou expression convertie implicitement en une valeur entière.
<i>expression_numérique</i>	Toute expression numérique renvoyant une valeur unique.
<i>expression_caractère</i>	Toute expression renvoyant une valeur unique de type caractère.
<i>expression_binaire</i>	Toute expression renvoyant une valeur unique de type <i>binary</i> ou <i>varbinary</i> .

Exemples

La plupart des exemples de ce manuel proviennent d'une base de données appelée pubtune. Le schéma de cette base de données est le même que celui de la base de données pubs2 mais les tables utilisées dans les exemples contiennent plus de lignes : la table titles en comporte 5 000, la table authors 5 000 et la table titleauthor 6 250. Divers index sont générés pour illustrer différentes fonctionnalités dans de nombreux exemples et ces index sont décrits dans le texte.

La base de données pubtune n'est pas fournie avec Adaptive Server. Etant donné que de nombreux exemples illustrent les résultats de commandes telles que set showplan et set statistics io, l'exécution des requêtes décrites dans ce manuel sur les tables pubs2 n'aboutira pas aux mêmes résultats d'E/S et, dans la plupart des cas, n'aboutira pas aux plans de requête présentés ici.

**Si vous avez
besoin d'aide**

Pour chaque installation Sybase bénéficiant d'un contrat de maintenance, une ou plusieurs personnes désignées sont autorisées à contacter le Support Technique de Sybase. Si vous ne parvenez pas à résoudre un problème en recourant aux manuels ou à l'aide en ligne, veuillez demander à la personne désignée de contacter le Support Technique Sybase ou la filiale Sybase la plus proche.

Introduction au guide Performances et optimisation

L'optimisation des performances d'Adaptive Server Enterprise comprend plusieurs processus destinés à analyser le pourquoi du ralentissement des performances, des conflits, de l'optimisation et de l'usage.

L'optimiseur de l'Adaptive Server prend une requête et cherche la meilleure manière de l'exécuter. L'optimisation s'effectue sur la base des statistiques d'une base de données ou d'une table. Le plan optimisé reste d'application tant que les statistiques ne sont pas mises à jour ou que la requête n'est pas modifiée. Vous pouvez mettre à jour les statistiques de l'ensemble de la table ou effectuer un échantillonnage sur un certain pourcentage de données.

Adaptive Server peut générer un plan abstrait pour une requête et sauvegarder le texte avec son plan abstrait dans la table système `sysqueryplans`. Vous pouvez recourir aux plans abstraits au lieu d'employer des options qui doivent figurer dans le batch ou dans la requête pour orienter les décisions de l'optimiseur. Avec des plans abstraits, vous pouvez jouer sur l'optimisation d'une instruction SQL sans avoir à modifier sa syntaxe.

Ce volume de la série Performances et optimisation propose des informations sur l'optimisation et les plans abstraits dans Adaptive Server.

Les autres manuels de la série Performances et optimisation sont les suivants :

- *Performances et optimisation : concepts de base*

Ce manuel décrit les principes de base en matière de compréhension et d'analyse des questions liées aux performances d'Adaptive Server. Il vous aide à rechercher les endroits susceptibles de nuire aux performances.

- *Performances et optimisation : verrouillage*

Pour protéger les tables, les pages ou les lignes de données utilisées par des transactions actives, Adaptive Server les verrouille. Le verrouillage est un mécanisme de contrôle des concurrences : il garantit l'intégrité des données au sein et au travers des transactions. Cette fonctionnalité est indispensable dans un environnement multi-utilisateur où les accès simultanés aux données sont fréquents.

- *Performances et optimisation : contrôle et analyse*

Adaptive Server utilise des rapports pour contrôler le serveur. Ce manuel explique comment les statistiques sont obtenues et utilisées à des fins de contrôle et d'optimisation. La procédure stockée `sp_sysmon` produit un rapport étendu qui montre les performances d'Adaptive Server.

Vous pouvez également utiliser le Sybase Monitor dans Sybase Central pour obtenir des informations en temps réel sur l'état du serveur.

Chaque manuel a été conçu pour couvrir des informations spécifiques pouvant être utilisées par l'administrateur système et l'administrateur de la base de données.

Présentation de l'optimiseur

Ce chapitre présente l'optimiseur de requêtes d'Adaptive Server et décrit les opérations mises en oeuvre lorsque vous exécutez des requêtes.

Sujet	Page
Définition	4
Importance de la taille des objets pour l'optimisation de la requête	6
Optimisation des requêtes	7
Facteurs examinés pour l'optimisation	9
Ajout de clauses optimisables durant la phase de prétraitement	10
Instructions pour créer des arguments de recherche	16
Arguments de recherche et utilisation des index	17
Syntaxe et traitement des jointures	25
Types de données incompatibles et optimisation des requêtes	28
Fractionnement des procédures stockées pour une meilleure estimation du coût	37
Unités principales d'estimation des coûts	38

Ce chapitre explique comment le système détermine les coûts de chaque clause de requête.

Le [Chapitre 5, « Méthodes d'accès et évaluation des coûts de requête sur une seule table »](#), traite de l'utilisation de ces coûts pour estimer le coût logique, physique et total des E/S pour des requêtes portant sur une seule table.

Le [Chapitre 6, « Méthodes d'accès et coût des requêtes pour des jointures et des sous-requêtes »](#), décrit l'utilisation des coûts lorsque des requêtes joignent deux ou plusieurs tables ou qu'elles incluent des sous-requêtes.

Définition

L'optimiseur examine les requêtes analysées et normalisées ainsi que les informations sur les objets de base de données. Les données fournies en entrée à l'optimiseur comportent une requête SQL analysée et des statistiques sur les tables, les index et les colonnes mentionnés dans la requête. Les résultats fournis par l'optimiseur constituent un **plan d'exécution de requête**.

Un plan d'exécution de requête est un code compilé qui contient l'ensemble ordonné des étapes pour effectuer la requête, y compris les méthodes d'accès à chaque table (balayage de table ou d'index, type de jointure à utiliser, ordre des jointures, etc.).

A partir de statistiques sur les tables et les index, l'optimiseur calcule le coût des méthodes d'accès possibles pour résoudre une requête. Il détermine le meilleur plan d'exécution de requête, c'est-à-dire le plus avantageux en termes d'E/S. Pour bon nombre de requêtes, il existe plusieurs plans possibles. Adaptive Server sélectionne le plan le moins coûteux, le compile et l'exécute.

Etapes du traitement d'une requête

Adaptive Server traite une requête selon les étapes suivantes :

- 1 La requête est analysée et normalisée. L'analyse permet de vérifier que la syntaxe SQL est correcte et la normalisation s'assure que tous les objets référencés dans la requête existent. Les autorisations sont contrôlées afin de vérifier que l'utilisateur a l'autorisation d'accéder à toutes les tables et colonnes de la requête.
- 2 Une phase de prétraitement modifie certains arguments de recherche afin de les améliorer et ajoute des arguments et des clauses de jointure optimisables.
- 3 Au cours de l'optimisation de la requête, le système analyse chacune de ces parties, puis adopte le plan d'exécution le plus approprié. Cette étape d'optimisation comprend :
 - l'analyse de chaque table ;
 - le calcul du coût d'utilisation de chaque index correspondant à un argument de recherche ou à une colonne de jointure ;
 - la sélection de l'ordre et du type de jointures ;
 - la sélection de la méthode d'accès finale.

- 4 Le plan d'exécution de requête choisi est compilé.
- 5 La requête est exécutée et le résultat est renvoyé à l'utilisateur.

Fonctionnement de l'optimiseur

Le rôle de l'optimiseur est de sélectionner pour chaque table la méthode d'accès permettant de réduire le temps total nécessaire au traitement d'une requête. Pour ce faire, il se fonde sur les statistiques disponibles pour les tables interrogées et sur d'autres facteurs, tels que les stratégies de cache, la taille du cache et la taille des E/S. Les statistiques disponibles pour les tables, les index et les colonnes sont les principaux éléments qui interviennent dans la prise de décision de l'optimiseur.

Dans certaines situations, il semble choisir des méthodes d'accès inadéquates. Ce choix inadapté peut résulter d'informations incomplètes ou inexactes (par exemple des statistiques périmées). Dans d'autres cas, une analyse complémentaire et le recours à des options spéciales de traitement peuvent aider à déterminer la cause du problème et fournir des solutions de contournement.

L'optimiseur de requêtes utilise le coût des E/S pour mesurer le coût d'exécution des requêtes. Les coûts significatifs pour le traitement des requêtes sont les suivants :

- E/S physiques, quand les pages doivent être lues dans le disque,
- E/S logiques, quand les pages du cache sont lues pour une requête.

Reportez-vous aux méthodes d'accès et coût de la requête.

Importance de la taille des objets pour l'optimisation de la requête

Pour comprendre les requêtes et le fonctionnement du système, il est essentiel de connaître la taille des tables et des index. Au cours de plusieurs étapes de l'optimisation, vous devez disposer des informations sur la taille pour pouvoir :

- interpréter les résultats renvoyés par `statistics io` pour un plan d'exécution de requête spécifique.

Le [Chapitre 4, « Utilisation des commandes set statistics »](#) du guide *Performances et optimisation : contrôle et analyse* explique comment utiliser `statistics io` pour obtenir des informations sur les E/S exécutées.

- comprendre le plan d'exécution de requête choisi par l'optimiseur. L'optimiseur Adaptive Server se base sur le coût pour évaluer les E/S physiques et logiques requises par chacune des méthodes d'accès disponibles et choisit la méthode la plus rentable. Si vous avez des doutes sur un plan d'exécution de requête, exécutez la commande `dbcc traceon(302)` pour connaître les raisons qui ont conduit l'optimiseur à faire ce choix. Ce résultat comporte des évaluations du nombre de pages.
- déterminer l'emplacement des objets, en fonction de leur taille et leurs E/S prévues. Vous pouvez obtenir de meilleures performances en répartissant les objets de base de données sur plusieurs devices physiques afin que les opérations d'écriture et de lecture soient réparties uniformément.

L'emplacement des objets est décrit au [Chapitre 6, « Gestion de l'emplacement physique des données »](#) du guide *Performances et optimisation : concepts de base*.

- comprendre les variations des performances. Si la taille des objets augmente, leurs caractéristiques de performances risquent de changer. Prenons par exemple une table utilisée intensivement qui est mise à 100 % en mémoire cache. Si la taille de cette table devient trop importante pour le cache, les requêtes qui utilisent la table peuvent subir une baisse soudaine des performances. Ce problème se pose en particulier pour les jointures nécessitant un grand nombre de balayages.

- élaborer des planifications de capacité. Que vous soyez sur le point de concevoir un nouveau système ou de planifier une extension d'un système existant, vous devez déterminer l'espace requis afin d'être en mesure de prévoir les besoins en mémoire et en disques physiques.
- interpréter les résultats renvoyés par Adaptive Server Monitor et `sp_sysmon` sur les E/S physiques.

Pour plus d'informations sur la taille, reportez-vous au *Guide d'administration système*.

Optimisation des requêtes

Pour bien comprendre ce processus et savoir s'il est possible d'améliorer les performances d'une requête, vous devez connaître son mode d'accès aux objets de base de données, la taille des objets et les index des tables.

Les symptômes suivants indiquent généralement des problèmes de performances :

- Une requête s'exécute beaucoup plus lentement que prévu, d'après la taille des index et des tables.
- Une requête s'exécute beaucoup plus lentement que d'autres requêtes du même type.
- Une requête s'exécute beaucoup plus lentement que d'habitude.
- Le traitement d'une requête à l'intérieur d'une procédure stockée est plus long que lorsqu'elle est traitée sous forme d'instruction.
- Le plan d'exécution de requête indique un balayage de table au lieu de l'utilisation d'un index.

Les problèmes de performances peuvent être dus à :

- L'absence de mise à jour récente des statistiques ; dans ce cas, la répartition des données réelles ne correspond pas aux valeurs utilisées par Adaptive Server pour optimiser les requêtes.
- Les lignes à référencer par une transaction ne correspondent pas au contenu des statistiques d'index.
- Un index est utilisé pour accéder à une grande partie de la table.

- Les clauses where sont écrites dans un format impossible à optimiser.
- Il n'existe pas d'index approprié pour une requête importante.
- Une procédure stockée a été compilée avant que des modifications importantes n'aient été apportées aux tables sous-jacentes.

Tables sous-jacentes SQL et optimisation

Les requêtes exprimées sous la forme d'une instruction SQL unique tirent un meilleur parti de l'optimiseur que celles exprimées dans deux instructions SQL ou plus. Les tables sous-jacentes SQL en activent une afin d'exprimer de manière concise et au cours d'une seule étape ce qui aurait exigé autrement plusieurs instructions SQL et tables temporaires, en particulier lorsque des résultats d'agrégats intermédiaires doivent être stockés. Par exemple,

```
select dt_1.* from
  (select sum(total_sales)
    from titles_west group by total_sales)
      dt_1(sales_sum),
  (select sum(total_sales)
    from titles_east group by total_sales)
      dt_2(sales_sum)
  where dt_1.sales_sum = dt_2.sales_sum
```

Dans cet exemple, les résultats d'agrégats sont fournis par les tables sous-jacentes SQL `dt_1` et `dt_2`, entre lesquelles une jointure est calculée. Tout se fait au travers d'une instruction SQL unique.

Pour plus d'informations sur les tables sous-jacentes SQL, reportez-vous au *Guide de l'utilisateur Transact-SQL*.

Facteurs examinés pour l'optimisation

Les plans d'exécution de requête sont constitués de stratégies de recherche et d'un ensemble ordonné d'étapes d'exécution permettant d'extraire les données demandées. Pour développer des plans d'exécution de requête, l'optimiseur prend en considération les éléments suivants :

- La taille de chaque table de la requête, aussi bien en nombre de lignes que de pages de données, et le nombres des pages OAM et d'allocation à lire.
- Les index existant sur les tables et les colonnes utilisées dans la requête, le type d'index, le nombre de niveaux, le nombre de pages de niveau feuille et le taux de clusterisation pour chaque index.
- La couverture de la requête par l'index, qui permet de satisfaire la requête par l'extraction de données à partir des pages feuille de l'index sans accéder aux pages de données. Adaptive Server peut utiliser des index couvrants, même si la requête ne contient aucune clause `where`.
- La densité et la répartition des clés dans les index.
- La taille du cache de données accessible et des E/S gérées par le cache, ainsi que la stratégie de cache à adopter.
- Le coût des lectures physiques et logiques.
- Les clauses de jointure pouvant être optimisées ainsi que l'ordre de jointure le plus approprié, en fonction des coûts et du nombre de balayages de tables requis pour chaque jointure et de l'utilité des index dans la limitation du nombre des E/S.
- La création d'une table de travail (table temporaire interne) ayant un index sur les colonnes de jointure est-elle plus rapide que des balayages de tables répétés quand il n'existe pas d'index exploitable pour la table interne d'une jointure ?
- La requête contient-elle un agrégat `max` ou `min` pouvant utiliser un index pour trouver la valeur sans balayer la table ?
- Les pages de données ou d'index seront-elles utilisées plusieurs fois pour satisfaire une requête telle qu'une jointure, ou est-il possible d'utiliser une stratégie de lecture-élimination lorsque les pages ne doivent être balayées qu'une seule fois ?

Pour chaque plan, l'optimiseur détermine le coût total en calculant les E/S physiques et logiques. Adaptive Server choisit ensuite le plan le moins coûteux.

Les procédures stockées et les triggers sont optimisés lors de la première exécution de l'objet et le plan d'exécution de requête est stocké dans le cache de la procédure. Si d'autres utilisateurs exécutent la même procédure alors que le cache contient un exemplaire inutilisé du plan, le plan d'exécution de requête compilé est copié dans le cache au lieu d'être recompilé.

Ajout de clauses optimisables durant la phase de prétraitement

Une fois la requête analysée et normalisée mais avant le début de l'analyse par l'optimiseur, la requête est prétraitée afin d'augmenter le nombre de clauses optimisables :

- Certains arguments sont convertis.
- Certaines expressions, comme les arguments de recherche, sont prétraitées afin de générer une valeur littérale pouvant être optimisée.
- Une fermeture transitive est appliquée aux arguments de recherche, chaque fois que possible.
- Une fermeture transitive est appliquée aux colonnes de jointure, chaque fois que possible.
- Pour certaines requêtes contenant la clause or, des arguments de recherche supplémentaires sont définis afin de fournir d'autres voies d'optimisation.

Les modifications apportées par le prétraitement sont transparentes ; pour les voir, vous devez examiner les résultats renvoyés par les outils d'optimisation, tels que showplan, statistics io ou dbcc traceon(302). Si vous exécutez des requêtes enrichies d'arguments de recherche optimisables, vous pouvez voir les clauses ajoutées :

- dans les résultats de dbcc traceon(302), au niveau de blocs supplémentaires d'évaluation des coûts pour les clauses optimisables ajoutées ;
- dans les résultats de showplan, où apparaîtront éventuellement des messages du type « Keys are » pour les tables dépourvues d'arguments de recherche ou de jointure.

Conversion des clauses en arguments de recherche équivalents

La phase de prétraitement recherche les clauses qu'il est possible de convertir au format des arguments de recherche (SARG). Celles-ci sont répertoriées dans le tableau 2-1.

Tableau 2-1 : Arguments de recherche équivalents

Clause	Conversion
between	Converti en clauses \geq et \leq . Par exemple, between 10 and 20 est convertie en ≥ 10 et ≤ 20 .
like	<p>Si le premier caractère du masque est une constante, les clauses like peuvent être converties en requêtes de type supérieur à ou inférieur à. Par exemple, like "sm%" devient $\geq "sm"$ and $< "sn"$.</p> <p>Si le premier caractère est un caractère joker, une clause telle que like "%x" ne peut pas utiliser d'index pour accéder aux données, mais des valeurs d'histogramme permettent d'estimer le nombre de lignes correspondantes.</p>
in (<i>liste_valeurs</i>)	Converti en une liste de requêtes or, c'est-à-dire que int_col in (1, 2, 3) devient int_col = 1 or int_col = 2 or int_col = 3. Le nombre maximal d'éléments dans une liste IN est 1025.

Conversion d'expressions en arguments de recherche

Bon nombre d'expressions sont converties en chaînes de recherche littérales avant l'optimisation de la requête. Dans les exemples suivants, les expressions traitées sont présentées telles qu'elles apparaissent dans l'analyse des arguments de recherche fournie par le résultat de dbcc traceon(302) :

Opération	Exemples de clause where	Expression traitée
Conversion implicite	numeric_col = 5	numeric_col = 5.0
Fonctions de conversion	int_column = convert(int, "77")	int_column = 77
Arithmétique	salary = 5000*12	salary = 6000 0
Fonctions mathématiques	width = sqrt(900)	width = 30
Fonctions de chaîne	shoe_width = replicate("E", 5)	shoe_width = "EEEEEE"

Opération	Exemples de clause where	Expression traitée
Concaténation de chaînes	full_name = "Fred" + " " + "Simpson"	full_name = "Fred Simpson"
Fonctions de date	week = datepart(wk, "5/22/99")	week = 21
Remarque getdate() ne peut pas être optimisé.		

Ces conversions permettent à l'optimiseur d'utiliser les valeurs d'histogramme pour une colonne au lieu des valeurs de sélectivité par défaut.

Il existe toute fois des exceptions :

- la fonction getdate,
- la plupart des fonctions système telles que object_id or object_name.

Celles-ci ne sont pas converties en valeurs littérales avant l'optimisation.

Fermeture transitive des arguments de recherche

Lors de la phase de prétraitement, une fermeture transitive est appliquée aux arguments de recherche. Par exemple, la requête suivante joint titles et titleauthor sur title_id et inclut un argument de recherche sur titles.title_id :

```
select au_lname, title
from titles t, titleauthor ta, authors a
where t.title_id = ta.title_id
      and a.au_id = ta.au_id
      and t.title_id = "T81002"
```

Cette requête est optimisée comme si elle incluait également l'argument de recherche sur titleauthor.title_id :

```
select au_lname, title
from titles t, titleauthor ta, authors a
where t.title_id = ta.title_id
      and a.au_id = ta.au_id
      and t.title_id = "T81002"
      and ta.title_id = "T81002"
```

Avec cette clause supplémentaire, l'optimiseur peut utiliser des statistiques relatives aux index sur titles.title_id afin d'estimer le nombre de lignes correspondantes dans la table titleauthor. Les estimations de coût les plus précises améliorent le choix des index et des ordres de jointure.

Fermeture transitive sur les jointures

Lors de la phase de prétraitement, une fermeture transitive est appliquée aux colonnes de jointure afin de détecter les équijoointures normales, si cette fermeture est activée au niveau serveur ou session. La requête suivante spécifie l'équijoindre de t1.c11 et t2.c21, ainsi que l'équijoindre de t2.c21 et t3.c31 :

```
select *
  from t1, t2, t3
 where t1.c11 = t2.c21
   and t2.c21 = t3.c31
   and t3.c31 = 1
```

Sans fermeture transitive, les seuls ordres de jointure pris en considération sont (t1, t2, t3), (t2, t1, t3), (t2, t3, t1) et (t3, t2, t1). En ajoutant la jointure sur t1.c11 = t3.c31, l'optimiseur enrichit la liste des ordres de jointure des possibilités suivantes : (t1, t3, t2) et (t3, t1, t2). La fermeture transitive d'arguments applique la condition spécifiée par t3.c31 = 1 aux colonnes de jointure de t1 et t2.

La recherche transitive n'est utilisée que pour les équijoointures normales, comme le montrent les exemples précédents. Elle n'est pas exécutée sur :

- des non-équijoointures, par exemple t1.c1 > t2.c2,
- des équijoointures qui incluent une expression, par exemple t1.c1 = t2.c1 + 5,
- des équijoointures dépendant d'une clause or,
- des jointures externes, par exemple t1.c11 *= t2.c2 ou left join ou right join,
- des jointures sortant des limites d'une sous-requête,
- des jointures utilisées pour vérifier l'intégrité référentielle ou with check option sur des vues,
- des colonnes ayant des types de données incompatibles.

Activation de la fermeture transitive sur les jointures

Un administrateur système peut activer la fermeture transitive sur des jointures au niveau du serveur à l'aide du paramètre de configuration enable sort-merge joins and JTC. Ce paramètre active aussi les jointures par fusion. Au niveau de la session, set jtc on active la fermeture transitive sur les jointures et il a la priorité sur le paramètre établi au niveau du serveur. Pour plus d'informations sur les types de requêtes qui peuvent bénéficier de la recherche transitive.

Reportez-vous à la section « Activation et désactivation de la fermeture transitive de jointures », page 58.

Transformation et factorisation de prédictats

La transformation et la factorisation de prédictats augmentent le nombre de choix disponibles pour l'optimiseur. Ces processus ajoutent des clauses optimisables dans une requête en récupérant des clauses dans des blocs de prédictats liés par or et en les insérant dans des clauses liées par and. L'ajout de ces clauses permet d'accroître le nombre des méthodes d'accès pour l'exécution de la requête. Les prédictats or initiaux sont maintenus afin de garantir l'exactitude de la requête.

Pendant la transformation des prédictats :

- 1 Les prédictats simples (jointures, arguments de recherche et listes in) qui présentent une correspondance exacte dans chaque clause or sont récupérés. Dans cet exemple, la clause a une exacte correspondante dans chaque bloc et est donc extraite :

```
t.pub_id = p.pub_id
```

Les clauses between sont converties en clauses « supérieur ou égal à » et « inférieur ou égal à » avant la transformation des prédictats. L'exemple ci-dessus utilise la clause between 15 dans les deux blocs de recherche (même si les fins d'intervalle sont différentes). La clause équivalente est donc extraite à l'étape 1 :

```
price >=15
```

- 2 Les arguments de recherche de la même table sont extraits et tous les termes référençant la même table sont traités en tant que prédictat unique au cours de l'expansion. type et price sont des colonnes de la table titles, de sorte que les clauses extraites sont :

```
(type = "travel" and price >=15 and price <= 30)  
or  
(type = "business" and price >= 15 and price <= 50)
```

- 3 Les listes in et les clauses or sont extraites. Si, dans un des blocs, il existe plusieurs listes in pour une table, seule la première est extraite. Dans notre exemple, les listes extraites sont les suivantes :

```
p.pub_id in ("P220", "P583", "P780")  
or  
p.pub_id in ("P651", "P066", "P629")
```

- 4 Ces différentes opérations peuvent se recouper et extraire la même clause ; les données en double sont donc éliminées.
- 5 Chaque terme généré est examiné et l'optimiseur détermine s'il peut servir d'argument de recherche ou de clause de jointure optimisable. Seuls les termes utiles à l'optimisation sont retenus.
- 6 Les clauses supplémentaires sont ajoutées aux clauses existantes déjà spécifiées par l'utilisateur.

Exemple

Toutes les clauses optimisables de cette requête figurent à l'intérieur de clauses or :

```
select p.pub_id, price
from publishers p, titles t
where (
    t.pub_id = p.pub_id
    and type = "travel"
    and price between 15 and 30
    and p.pub_id in ("P220", "P583", "P780")
)
or (
    t.pub_id = p.pub_id
    and type = "business"
    and price between 15 and 50
    and p.pub_id in ("P651", "P066", "P629")
)
```

La transformation des prédictats extrait les clauses liées par and des blocs de clauses liées par or, comme indiqué dans l'exemple. Elle n'extract que les clauses qui apparaissent dans les blocs entre parenthèses. Dans l'exemple ci-dessus, si une clause de l'un des blocs liés par or ne figurait pas dans l'autre clause, elle ne serait pas extraite.

Instructions pour créer des arguments de recherche

Lorsque vous écrivez des arguments de recherche pour vos requêtes, suivez les instructions ci-après :

- Evitez d'indiquer des fonctions, des opérations arithmétiques et d'autres expressions dans la partie des clauses de recherche dans la colonne. Si possible, déplacez les fonctions et les autres opérations vers la partie expression de la clause.
- Evitez les types de données incompatibles pour des colonnes qui seront jointes et pour des variables et paramètres employés comme arguments de recherche.
Pour de plus amples informations, reportez-vous à la section « [Types de données incompatibles et optimisation des requêtes](#) », page 28.
- Utilisez la colonne principale d'un index composé comme argument de recherche. L'optimisation de clés secondaires offre des gains de performance moins importants.
- Utilisez le plus d'arguments de recherche possible pour fournir à l'optimiseur un maximum d'éléments de travail.
- Si une requête comprend plus de 102 prédictats pour une table, placez les clauses les plus utiles au début de la requête car seuls les 102 premiers arguments de recherche de chaque table sont utilisés lors de l'optimisation. (L'ensemble des conditions de recherche est utilisé pour répondre à la requête.)
- Certaines requêtes qui utilisent > (supérieur à) seront plus performantes si vous les réécrivez de sorte qu'elles utilisent >= (supérieur ou égal à). Dans notre exemple, la recherche, avec un index sur int_col, utilise l'index pour trouver la première valeur où int_col est égale à 3, puis continue le balayage pour rechercher la première valeur supérieure à 3. Si int_col est égale à 3 dans un grand nombre de lignes, le serveur doit balayer plusieurs pages pour trouver la première ligne où int_col est supérieure à 3 :

```
select * from table1 where int_col > 3
```

Il sera plus efficace d'écrire la requête comme suit :

```
select * from table1 where int_col >= 4
```

Cette optimisation est plus difficile avec les chaînes de caractères et les données à virgule flottante. Vous devez connaître vos types de données.

- Consultez le résultat de showplan pour connaître les clés et les index utilisés.
- Si vous pensez qu'un index n'est pas utilisé alors qu'il le devrait, vérifiez les résultats fournis par dbcc traceon(302) pour voir si l'optimiseur a pris cet index en compte.

Arguments de recherche et utilisation des index

Il est important de faire la distinction entre les prédictats des clauses where et having susceptibles d'optimiser la requête et ceux qui permettront ultérieurement, lors du traitement de la requête, de filtrer les lignes à renvoyer.

Les arguments de recherche permettent de déterminer le chemin d'accès aux lignes de données lorsqu'une colonne de la clause where correspond à une clé d'index principale. L'index permet pour sa part de localiser et de récupérer les lignes de données qui satisfont à la requête. Une fois que la ligne a été localisée dans le cache de données ou lue sur le disque, les clauses restantes sont appliquées.

Par exemple, si la table authors possède un index sur au_lname et un autre sur city, l'un ou l'autre peut servir à localiser les lignes adéquates :

```
select au_lname, city, state
from authors
where city = "Washington"
and au_lname = "Catmull"
```

L'optimiseur utilise des statistiques, notamment les histogrammes, le nombre de lignes d'une table, les niveaux d'index et les taux de clusterisation des pages d'index et de données afin de déterminer l'index le plus avantageux pour accéder aux données. L'index ainsi retenu est sélectionné et utilisé pour exécuter la requête, puis l'autre clause est appliquée aux lignes de données une fois qu'elles ont été localisées.

Syntaxe des arguments de recherche

Les arguments de recherche sont des expressions qui peuvent se présenter sous l'une des formes suivantes :

```
<colonne> <opérateur> <expression>
<expression> <opérateur> <colonne>
<colonne> is null
```

où :

- *colonne* représente uniquement un nom de colonne. Si des fonctions, des expressions ou une concaténation sont ajoutées au nom de colonne, il est impossible d'utiliser un index sur la colonne.
- *opérateur* correspond à l'un des opérateurs suivants :
`=, >, <, >=, <=, !>, !<, <>, !=, is null`
- *expression* est soit une constante, soit une expression produisant une constante. L'optimiseur n'utilise pas les statistiques d'index de la même manière, selon que la valeur de l'expression est connue ou non au moment de la compilation :
 - Si l'*expression* est une constante connue ou si elle peut être convertie en constante connue lors du prétraitement, elle peut être comparée aux valeurs d'histogramme stockées pour un index afin de fournir une évaluation précise du nombre de lignes.
 - Si la valeur de *expression* n'est pas connue au moment de la compilation, l'optimiseur utilise la densité totale pour évaluer le nombre de lignes que la requête doit renvoyer. La valeur des variables définies dans un batch de requête ou les paramètres définis dans une procédure stockée ne peuvent pas être connus avant l'exécution.
 - Si le type de données de l'*expression* est incompatible avec celui de la colonne, il est impossible d'utiliser un index et il n'est pas pris en considération.

Pour de plus amples informations, reportez-vous à la section « [Types de données incompatibles et optimisation des requêtes](#) », page 28.

Opérateurs d'inégalité

Les opérateurs d'inégalité < > et != sont des cas particuliers. Si la colonne est indexée, l'optimiseur vérifie s'il existe des index non clusterisés couvrant la requête et, si tel est le cas, il effectue un balayage d'index sans correspondance. Cependant, si l'index ne couvre pas la requête, l'accès à la table s'effectue via un balayage de table.

Exemples d'arguments de recherche

Voici quelques exemples de clauses entièrement optimisables. S'il existe des statistiques sur ces colonnes, elles faciliteront l'estimation du nombre de lignes que la requête doit renvoyer. De même, s'il existe des index sur les colonnes, ils peuvent servir à accéder aux données :

```
au_lname = "Bennett"
price >= $12.00
advance > $10000 and advance < $20000
au_lname like "Ben%" and price > $12.00
```

Les arguments suivants ne sont pas des arguments de recherche optimisables :

```
advance * 2 = 5000 /*expression on column side
not permitted */
substring(au_lname,1,3) = "Ben" /* function on
column name */
```

Ces deux clauses peuvent être optimisées si elles sont écrites dans ce format :

```
advance = 5000/2
au_lname like "Ben%"
```

Prenons comme exemple la requête suivante, avec le seul index sur au_lname :

```
select au_lname, au_fname, phone
      from authors
     where au_lname = "Gerland"
           and city = "San Francisco"
```

La clause peut servir d'argument de recherche :

```
au_lname = "Gerland"
```

- Il existe un index sur au_lname.
- Il n'y a pas de fonctions ni d'autres opérations sur le nom de la colonne.

- L'opérateur est un opérateur d'argument de recherche adéquat.
- Le type de données de la constante est le même que celui de la colonne.
`city = "San Francisco"`

Cette clause répond à tous les critères indiqués ci-dessus, à l'exception du premier : il n'y a pas d'index sur la colonne city. Dans ce cas, l'index sur au_lname est utilisé pour la requête. Toutes les pages contenant un nom correspondant à cet argument sont mises en mémoire cache ; chaque ligne correspondante est examinée pour vérifier si la ville répond au critère indiqué.

Utilisation des statistiques pour les arguments de recherche

Quand vous créez un index, des statistiques sont générées et stockées dans des tables système. Certaines sont utiles pour déterminer le coût des arguments de recherche et des jointures, notamment :

- Statistiques pour l'index : nombre de pages et de lignes, niveaux de l'index, nombre de pages de niveau feuille, taille moyenne des lignes de niveau feuille.
- Statistiques sur les données de la colonne :
 - Un histogramme pour la colonne principale de l'index. Les histogrammes servent à déterminer la sélectivité de l'argument de recherche, à savoir le nombre de lignes de la table qui correspondent à une valeur donnée.
 - Des valeurs de densité, qui mesurent la densité des clés de l'index.
- Taux de clusterisation qui mesurent la fragmentation du stockage des données et l'efficacité des E/S étendues.

Une partie seulement de ces statistiques (nombre de pages de niveau feuille par exemple) est prise en considération lors du traitement de la requête. Les autres sont mises à jour uniquement lorsque vous exécutez la commande `update statistics` ou lorsque vous supprimez puis recréez l'index. Pour afficher ces statistiques, utilisez `optdiag`.

Reportez-vous au [Chapitre 6, « Affichage des tables de statistiques avec optdiag »](#) du guide *Performances et optimisation : contrôle et analyse*.

Cellules d'histogramme

Lorsque vous générez un index, un histogramme est créé sur la première colonne de cet index. Il stocke des informations sur la répartition des valeurs dans la colonne. La commande `update statistics` vous permet également de générer des statistiques sur les clés mineures d'un index composé et sur les colonnes utilisées dans des clauses de recherche non indexées.

L'histogramme d'une colonne réunit les données dans un ensemble d'opérations ou de cellules. Le nombre de ces cellules peut être spécifié lors de la création de l'index ou lors de l'exécution de la commande `update statistics`. Pour chaque cellule, l'histogramme enregistre une valeur de colonne et un poids.

Il existe dans les histogrammes deux types de cellules :

- Une **cellule de fréquences**, représentant une valeur qui indique une forte proportion d'éléments en double dans la colonne. Le poids de ce type de cellule multiplié par le nombre de lignes de la table donne le nombre de lignes de la table qui correspondent à la valeur de la cellule. Si une colonne ne contient que peu de valeurs dupliquées, l'histogramme ne se compose que de cellules d'intervalles.
- Des **cellules d'intervalles**, représentant un intervalle de valeurs. Le poids des cellules d'intervalles et leur densité servent à évaluer le nombre de lignes à renvoyer lorsque des valeurs d'arguments de recherche figurent dans l'intervalle spécifié par une cellule.

Pour plus d'informations sur les histogrammes, reportez-vous à la section « [Affichages d'histogramme](#) », page 157 du guide *Performances et optimisation : contrôle et analyse*.

Valeurs de densité

La densité est la proportion moyenne de clés dupliquées dans l'index. Elle varie entre 0 et 1. Un index comportant N lignes dont les clés sont uniques a une densité égale à $1/N$; un index dont les clés sont toutes des valeurs dupliquées les unes des autres a une densité de 1.

Pour les index comportant plusieurs clés, des valeurs de densité sont calculées et stockées pour chaque ensemble ordonné de clés dans l'index. Par exemple, pour un index sur les colonnes A, B, C et D, des densités sont stockées pour :

- A
- A, B
- A, B, C
- A, B, C, D

Densité des cellules d'intervalles et densité totale

Pour chaque sous-ensemble ordonné, deux densités sont stockées :

- la densité de la cellule d'intervalles, utilisée avec les arguments de recherche ;
- la densité totale, utilisée pour les jointures.

La densité de l'intervalle de cellules correspond au nombre moyen de duplications de toutes les valeurs représentées par les cellules d'intervalles dans l'histogramme. La densité totale représente le nombre moyen de doublons pour toutes les valeurs, qu'il s'agisse des cellules de fréquence ou d'intervalles. La densité totale est utilisée pour estimer le nombre de lignes qualifiées pour les jointures et les arguments de recherche dont la valeur n'est pas connue lors de l'optimisation de la requête.

Utilisation des densités et des histogrammes par l'optimiseur

Lorsque l'optimiseur analyse un argument de recherche, il utilise les valeurs de l'histogramme, les densités et le nombre de lignes de la table afin de déterminer combien de lignes correspondent à la valeur spécifiée dans l'argument :

- Si la valeur de l'argument de recherche correspond à une cellule de fréquence, le nombre estimé de lignes correspondantes est égal au poids de la cellule de fréquence multiplié par le nombre de lignes de la table. La requête ci-après inclut une valeur avec un nombre élevé de données dupliquées, de sorte qu'il y a correspondance avec une cellule de fréquence :

```
where authors.city = "New York"
```

Si le poids de la cellule de fréquence est #0,015606 et si la table authors contient 5 000 lignes, l'optimiseur évalue le résultat renvoyé par la requête à $5\ 000 * 0,015606$, soit 78 lignes.

- Si la valeur de l'argument de recherche figure dans une cellule d'intervalles, l'optimiseur utilise la densité de cette dernière pour évaluer le nombre de lignes. Prenons par exemple une requête sur une valeur city comprise dans une cellule d'intervalle, dont la densité est de 0,000586 pour la colonne : le résultat renvoyé sera $5\ 000 * 0,000586 = 3$ lignes.
- Pour les requêtes sur des intervalles, l'optimiseur ajoute les poids de toutes les cellules couvertes par l'intervalle de valeurs. Lorsque le début ou la fin de l'intervalle tombe dans une cellule d'intervalles, l'optimiseur procède par interpolation pour déterminer le nombre de lignes de cette cellule qui sont comprises dans l'intervalle.

Utilisation des statistiques sur plusieurs arguments de recherche

Lorsqu'il existe plusieurs arguments de recherche sur la même table, l'optimiseur a recours à des statistiques afin de combiner la sélectivité des arguments.

La requête qui suit spécifie des arguments de recherche pour deux colonnes de la table :

```
select title_id
  from titles
 where type = "news"
   and price < $20
```

Avec un index sur type, price, les estimations de sélectivité varient selon que des statistiques ont été créées ou non pour la colonne price :

- S'il n'existe de statistiques que pour la colonne type, l'optimiseur utilise le poids de la cellule de fréquence pour type et une sélectivité par défaut pour price. La sélectivité pour type est de #0,106600 et la sélectivité par défaut pour une requête à intervalles ouverts est de 33 %. Le nombre estimé de lignes que la requête doit renvoyer s'obtient par la multiplication suivante : 0,106600 * 0,33, soit 0,035178. Si la table contient 5 000 lignes, l'estimation est égale à 171 lignes.

Pour connaître les valeurs par défaut utilisées en l'absence de statistiques, reportez-vous au [tableau 2-2](#).

- Si des statistiques sont ajoutées à la colonne price, l'histogramme permet d'estimer à 0,133334 le nombre de lignes correspondant à l'argument de recherche sur price. En multipliant cette valeur par la sélectivité de type, on obtient 0,014213 et l'estimation est donc de 71 lignes.

Le nombre réel de lignes renvoyées est de 53 pour cette requête, ce qui montre que les statistiques supplémentaires ont amélioré la précision. Pour cette requête simple contenant une seule table, la meilleure précision de la sélectivité n'a pas entraîné le changement de la méthode d'accès, à savoir le recours à l'index sur type, price. Cependant, dans certaines requêtes portant sur une seule table, l'ajout de statistiques peut amener l'optimiseur à choisir entre un balayage de la table ou l'utilisation d'autres index. Dans des requêtes de jointure, la présence de statistiques plus précises sur chaque table peut donner lieu à des ordres de jointure plus performants.

Valeurs par défaut des arguments de recherche

Lorsqu'il n'existe pas de statistiques pour un argument de recherche ou lorsque la valeur d'un argument n'est pas connue au moment de l'optimisation, l'optimiseur utilise des valeurs par défaut. Ces valeurs sont présentées dans le [tableau 2-2](#).

Tableau 2-2 : Approximations de densité pour des arguments de recherche inconnus

Type d'opération	Opérateur	Approximations de densité
Egalité	=	Densité totale, s'il existe des statistiques pour la colonne, ou 10 %
Intervalle ouvert	<, <=, > ou >=	33 %
Intervalle fermé	between	25 %

Arguments de recherche utilisant des variables et des paramètres

Dans la mesure où l'optimiseur établit une estimation avant l'exécution de la requête, il lui est impossible de connaître la valeur d'une variable définie dans le batch ou la procédure. Si la valeur d'une variable est inconnue au moment de la compilation, l'optimiseur utilise les valeurs par défaut reprises dans le [tableau 2-2](#).

Par exemple, la valeur de `@city` est définie dans ce batch :

```
declare @city varchar(25)
select @city = city from publishers
      where pub_name = "Brave Books"
select au_lname from authors where city = @city
```

L'optimiseur utilise la densité totale, à savoir 0,000879, et estime que 4 lignes seront renvoyées. Le nombre réel de lignes peut cependant être bien plus élevé.

Ce même type de problème se pose lorsque vous définissez les valeurs de variables au sein d'une procédure stockée. Dans ce cas, vous améliorerez les performances en fractionnant la procédure : définissez la variable dans la première procédure puis appelez la seconde procédure en transmettant les variables en tant que paramètres. La seconde procédure peut ainsi être optimisée convenablement.

Pour obtenir un exemple, reportez-vous à la section « [Fractionnement des procédures stockées pour une meilleure estimation du coût](#) », page 37.

Syntaxe et traitement des jointures

Le format des clauses de jointure est le suivant :

```
table1.column_name <operator> table2.column_name
```

Les opérateurs de jointure sont les suivants :

```
=, >, >=, <, <=, !>, !<, !=, <>, *=, =*
```

Et :

```
table1 [ left | right ] join table2
        on column_name = column_name
table1 inner join table2
        on column_name = column_name
```

Lorsque des jointures sont optimisées, l'optimiseur ne peut prendre en considération que les index sur des noms de colonnes. Lorsque des opérateurs ou des expressions sont associés au nom de colonne, l'optimiseur ne choisit pas un index sur la colonne comme méthode d'accès. Si les colonnes de la jointure ont des types de données incompatibles, l'optimiseur ne peut envisager qu'un index sur une des colonnes.

Traitement des jointures

Lorsque l'optimiseur crée un plan d'exécution pour une requête de jointure :

- Il évalue les index de chaque table en estimant les E/S requises pour l'utilisation d'un index et pour un balayage de table.
- Il détermine l'ordre de jointure, en fonction du coût total évalué pour les différents ordres possibles. Il évalue les coûts des jointures à boucle imbriquée et des jointures tri-fusion.
- S'il n'existe aucun index exploitable sur la table interne d'une jointure, l'optimiseur peut choisir de construire un index temporaire ; on parle dans ce cas de **redéfinition d'index**.

Reportez-vous à la section « [Stratégie de redéfinition d'index](#) », page 141.

- Il détermine la taille des E/S et la stratégie de cache.
- Il compare aussi le coût de l'exécution parallèle et en série, si le traitement parallèle des requêtes est opérationnel.

Pour de plus amples informations, reportez-vous au [Chapitre 8](#),
« Optimisation des requêtes parallèles ».

Les facteurs qui déterminent les coûts pour des sélections sur table unique, tels qu'une indexation appropriée, la sélectivité des arguments de recherche ou la densité des clés, sont plus stratégiques encore pour les jointures.

Statistiques de jointure non disponibles

S'il n'existe pas de statistiques sur une colonne dans une jointure, l'optimiseur applique des valeurs par défaut :

Type d'opérateur	Exemples	Sélectivité par défaut
Egalité	t1.c1 = t1.c2	1/lignes dans une petite table
Inégalité	t1.c1 > t1.c2 t1.c1 >= t1.c2 t1.c1 < t1.c2 t1.c1 <= t1.c2	33 %

Par exemple, dans la requête suivante, s'il n'existe pas de statistiques sur la colonne city, l'optimiseur utilise la valeur 1/500 comme sélectivité de jointure pour les deux tables ; stores contient 500 lignes et authors 5 000 lignes :

```
select au_fname, au_lname, stor_name
      from authors a, stores s
     where a.city = s.city
```

Valeurs de densité et jointures

Lorsqu'il existe des statistiques sur une colonne de jointure, la densité totale sert à calculer le nombre de lignes qui correspondent à chaque clé de jointure. Si la table authors contient 5 000 lignes et si la densité totale pour la colonne city est 0,000879, l'optimiseur estime que la requête va renvoyer $5\ 000 * 0,000879 = 4$ lignes de la table authors chaque fois qu'une jointure sur la colonne city correspond à une ligne de l'autre table.

Plusieurs colonnes de jointures

Lorsqu'une requête de jointure spécifie plusieurs colonnes de jointure sur deux tables et qu'il existe un index composé sur les colonnes, la densité totale composée est utilisée. Par exemple, si authors et publishers ont chacune un index sur city, state, la densité totale composée pour city, state est utilisée pour chaque table de la requête :

```
select au_lname, pub_name
from authors a, publishers p
where a.city = p.city
and a.state = p.state
```

Arguments de recherche et jointures sur une table

Lorsqu'il existe des arguments de recherche et des jointures sur une table, la sélectivité des colonnes est combinée lors de l'évaluation du coût afin de fournir une estimation plus précise du nombre de lignes.

Les exemples suivants joignent authors et stores sur les deux colonnes city et state. Un argument de recherche existant sur authors.state, une fermeture transitive d'argument de recherche l'ajoute également pour stores.state :

```
select au_fname, au_lname, stor_name
from authors a, stores s
where a.city = s.city
and a.state = s.state
and a.state = "GA"
```

S'il existe un index sur la colonne city de chaque table mais pas de statistiques pour state, l'optimiseur utilise la sélectivité par défaut de l'argument de recherche (10 %) combinée à la densité totale de city. Il en résulte une surestimation du nombre de lignes qui correspondent à l'argument de recherche de cette requête ; en revanche, pour une colonne state comportant davantage de lignes en correspondance avec un argument de recherche, il y aurait sous-estimation. Lorsque des statistiques existent pour la colonne state de chaque table, l'estimation du nombre de lignes qualifiées est meilleure et l'évaluation globale du coût de la requête de jointure est également plus pointue.

Types de données incompatibles et optimisation des requêtes

Dans une requête, l'impossibilité d'utiliser les index comme prévu tient souvent à des incompatibilités entre les types de données. Ces incompatibilités se produisent dans les cas suivants :

- avec des clauses de recherche utilisant des variables ou des paramètres de procédure stockée dont les types de données sont différents de celui de la colonne, par exemple :

```
where int_col = @money_parameter
```

- dans des requêtes de jointure dont les colonnes jointes ont des types de données différents, par exemple :

```
where tableA.int_col = tableB.money_col
```

Les incompatibilités entre les types de données entraînent des difficultés lorsqu'elles empêchent l'optimiseur de prendre un index en considération. Les problèmes les plus courants sont dus à :

- des comparaisons entre les types d'entier, int, smallint et tinyint
- des comparaisons entre les types money et smallmoney
- des comparaisons entre datetime et smalldatetime
- des comparaisons entre des types numeric et decimal de précision et d'échelle différentes
- des comparaisons entre des types numeric ou decimal et des colonnes integer ou money

Pour éviter ces difficultés, utilisez le même type de données (y compris la précision et l'échelle) pour les colonnes susceptibles d'être jointes lorsque vous créez des tables. Utilisez un type de données correspondant pour les variables ou les paramètres de procédure stockée employés comme arguments de recherche. Les sections suivantes décrivent les règles et les mesures appliquées lorsque les types de données diffèrent. Elles sont accompagnées de quelques conseils pour résoudre les problèmes.

Présentation de la hiérarchie des types de données et des problèmes d'index

La hiérarchie des types de données contrôle l'utilisation des index lorsque des arguments de recherche ou des colonnes de jointure ont des types de données différents. La requête suivante imprime les valeurs de cette hierarchy et les noms des types de données correspondants :

```
select hierarchy, name from systypes order by 1
hierarchy name
-----
1 floatn
2 float
3 datetimn
4 datetime
5 real
6 numericn
7 numeric
8 decimaln
9 decimal
10 moneyn
11 money
12 smallmoney
13 smalldatetime
14 intn
15 int
16 smallint
17 tinyint
18 bit
19 univarchar
20 unichar
21 reserved
22 varchar
22 sysname
22 nvarchar
23 char
23 nchar
24 varbinary
24 timestamp
25 binary
26 text
27 image
```

Si vous avez créé des types de données utilisateur, ils figurent aussi dans les résultats de la requête, avec les valeurs hiérarchiques correspondantes.

La règle générale veut qu'en cas de types de données différents, la valeur `systypes.hierarchy` détermine s'il est possible ou non d'utiliser un index.

- Pour les arguments de recherche, l'index est pris en compte lorsque le type de données de la colonne est identique à la valeur `hierarchy` du paramètre ou de la variable ou qu'il la précède immédiatement dans la liste.
- Pour une jointure, l'index est pris en compte uniquement sur la colonne dont la valeur `systypes.hierarchy` est identique à celle de l'autre colonne ou qu'elle la précède dans la hiérarchie.
- Quand les types de données `char` et `unichar` sont utilisés ensemble, `char` est converti en `unichar`.

Les exceptions sont les suivantes :

- Comparaisons entre les types de données `char` et `varchar`, `unichar` et `univarchar`, ou entre `binary` et `varbinary`. Par exemple, bien que leurs valeurs hiérarchiques soient respectivement égales à 23 et 22, les colonnes `char` et `varchar` sont traitées comme ayant le même type de données au moment de la prise en compte de l'index. En effet, l'index est pris en considération pour les deux colonnes de la jointure :

```
where t1.char_column = t2.varchar_column
```

Les colonnes `char` qui acceptent les valeurs `NULL` sont stockées comme `varchar`, mais les index restent utilisables sur les deux colonnes.

- Le type `NULL` de la colonne n'a aucune influence. En effet, bien que `float` et `floatn` aient des valeurs hiérarchiques différentes, elles sont traitées comme si elles avaient le même type de données.
- Dans les comparaisons des types `decimal` ou `numeric`, la précision et l'échelle sont également prises en compte. Il s'agit notamment de comparaisons de types `numeric` ou `decimal` entre eux et de comparaisons des types `numeric` ou `decimal` avec d'autres types de données, tels que `int` ou `money`.

Pour de plus amples informations, reportez-vous à la section
[« Comparaison des types de données numeric et decimal », page 31.](#)

Comparaison des types de données numeric et decimal

Lorsqu'une requête joint des colonnes de type numeric ou decimal, il est possible d'utiliser un index lorsque les deux conditions suivantes sont remplies :

- l'échelle de la colonne susceptible de figurer dans la jointure égale ou excède celle de l'autre colonne de la jointure, et
- la longueur de la partie entière de la colonne égale ou excède la longueur de la partie entière de l'autre colonne.

Voici quelques exemples où l'utilisation d'index est possible :

Types de données dans les jointures	Utilisation possible d'index
numeric(12,4) et numeric(16,4)	Index pris en compte uniquement pour le type numeric(16,4) ; la partie entière de numeric(12,4) est plus petite.
numeric(12,4) et numeric(12,8)	Aucun des index n'est pris en compte car la partie entière de numeric(12,8) est plus petite et l'échelle de numeric(12,4) est plus petite.
numeric(12,4) et numeric(12,4)	Les deux index sont pris en considération.

Comparaison de types numériques avec d'autres types de données

Lors de la comparaison des colonnes numeric et decimal avec des colonnes d'un autre type tel que money ou int :

- numeric et decimal précèdent les colonnes integer et money dans la hiérarchie, de sorte que l'index sur la colonne numeric ou decimal est le seul pris en compte ;
- les conditions relatives à la précision et à l'échelle doivent être remplies pour que l'index numeric ou decimal soit pris en compte. L'échelle de la colonne numeric doit être égale ou supérieure à celle de la colonne integer ou money et le nombre de chiffres dans la partie entière de la colonne numeric doit être égal ou supérieur au nombre maximum de chiffres utilisables dans la colonne integer ou money.

La précision et l'échelle des types integer et money sont indiquées dans le tableau 2-3.

Tableau 2-3 : Précision et échelle des types integer et money

Type de données	Précision, échelle
tinyint	3,0
smallint	5,0
int	10,0
smallmoney	10,4
money	19,4

Types de données pour les paramètres et les variables utilisés comme arguments de recherche

Lorsque vous déclarez des types de données pour des variables ou des paramètres de procédure stockée utilisés comme arguments de recherche, vous devez les faire correspondre au type de la colonne dans la déclaration de la variable ou du paramètre afin qu'il soit possible d'utiliser un index.

Par exemple :

```
declare @int_var int
select @int_var = 50
select *
from t1
where int_col = @int_var
```

L'utilisation de l'index dépend de l'ordre des types de données dans la hiérarchie. L'index sur une colonne n'est utilisable que si le type de données de celle-ci précède dans la hiérarchie le type de données de la variable.

Par exemple, int précède smallint et tinyint dans la hiérarchie. Les types d'entier sont les suivants :

hierarchy	name
15	int
16	smallint
17	tinyint

Si une variable ou un paramètre a le type de données smallint ou tinyint, il est possible d'utiliser un index sur une colonne int dans la requête. En revanche, un index sur une colonne tinyint n'est pas envisageable si le paramètre est de type int.

De même, money précède int. Si une variable ou un paramètre money est comparé à une colonne int, il est impossible d'utiliser un index sur la colonne int.

Cela évite que des problèmes se posent en cas de troncature ou d'overflow. Ainsi, dans la requête suivante, il ne serait pas intéressant ni correct de chercher à tronquer à 5 la valeur money pour utiliser un index sur *int_col* :

```
declare @money_var money
select @money_var = $5.12
select * from t1 where int_col = @money_var
```

Résolution des problèmes d'incompatibilité des types de données dans les arguments de recherche

S'il y a incompatibilité de types de données dans un argument de recherche sur une colonne indexée, la requête peut utiliser un autre index s'il existe d'autres arguments de recherche, ou procéder à un balayage de la table. Les résultats de showplan indiquent la méthode d'accès et les clés pour chaque table d'une requête.

L'outil de diagnostic dbcc traceon(302) permet également de savoir si un index est pris en compte. Par exemple, l'utilisation d'une variable de type integer comme argument de recherche sur une colonne *int_col* génère les résultats suivants :

```
Selecting best index for the SEARCH CLAUSE:
t1.int_col = unknown-value
```

```
SARG is a local variable or the result of a function
or an expression, using the total density to estimate
selectivity.
```

```
Estimated selectivity for int_col,
selectivity = 0.020000.
```

L'emploi d'un type de données incompatible tel que money pour une variable utilisée comme argument de recherche sur une colonne de type integer ne génère pas le message « Selecting best index for the SEARCH CLAUSE » dans les résultats de dbcc traceon(302), indiquant ainsi que l'index n'est pas pris en compte et ne peut pas être utilisé. Si vous ne pouvez pas utiliser un index comme prévu, examinez d'abord, pour tenter de résoudre le problème, la section concernant l'évaluation du coût dans les résultats de dbcc traceon(302).

Le message « unknown-value » et le recours à la densité totale pour évaluer le nombre de lignes correspondant à cet argument de recherche s'expliquent par le fait que la valeur de la variable a été définie dans le batch ; il ne s'agit donc pas d'un problème d'incompatibilité de types de données.

Pour de plus amples informations, reportez-vous à la section « [Arguments de recherche utilisant des variables et des paramètres](#) », page 24.

Types de données compatibles pour les colonnes jointes

L'optimiseur ne prend en considération un index sur des colonnes jointes que lorsque les types de colonnes sont identiques ou lorsque le type de la colonne de jointure précède dans la hiérarchie le type de l'autre colonne. Par conséquent, l'optimiseur n'envisage l'utilisation de l'index que sur l'une des colonnes de la jointure, limitant ainsi le choix des ordres de jointure.

Par exemple, la requête ci-après joint des colonnes de type decimal et int :

```
select *
  from t1, t2
 where t1.decimal_col = t2.int_col
```

decimal précédant int dans la hiérarchie, l'optimiseur peut envisager l'emploi d'un index sur t1.decimal_col, mais pas sur t2.int_col. Il en résultera probablement un balayage de la table t2, puis l'utilisation de l'index sur t1.decimal_col.

Le [tableau 2-4](#) montre comment, avec certains types de données qui posent souvent problème, la hiérarchie influe sur le choix de l'index.

Tableau 2-4 : Index possibles dans le cas de types de colonnes incompatibles

Types de colonnes de jointure	Index pris en compte sur la colonne de type
money et smallmoney	money
datetime et smalldatetime	datetime
int et smallint	int
int et tinyint	int
smallint et tinyint	smallint

Résolution des problèmes d'incompatibilité des types de données dans les jointures

Si vous pensez qu'un index n'est pas pris en considération sur un côté de la jointure en raison d'une incompatibilité de types de données, utilisez l'outil de diagnostic dbcc traceon(302). Dans les résultats, cherchez le message « Selecting best index for the JOIN CLAUSE ». Si les types de données sont compatibles, ce bloc de texte apparaît pour chaque jointure, par exemple :

```
Selecting best index for the JOIN CLAUSE:  
t1.int_col = t2.int_col
```

Et un peu plus loin pour l'autre table de la jointure :

```
Selecting best index for the JOIN CLAUSE:  
t2.int_col = t1.int_col
```

Si la requête compare des types de données incompatibles, par exemple une colonne decimal avec une autre de type int, ce bloc de texte n'apparaît qu'une fois :

```
Selecting best index for the JOIN CLAUSE:  
t1.decimal_col = t2.int_col
```

Cela signifie que pour évaluer le coût de la jointure, l'optimiseur n'a pas envisagé l'utilisation d'un index avec t2.int_col en tant que colonne externe.

Suggestions concernant les types de données et les comparaisons

Pour éviter des problèmes d'incompatibilité de type de données :

- Lorsque vous créez des tables, utilisez les mêmes types de données pour les colonnes qui seront jointes.
- Si des colonnes de deux tables souvent jointes ont des types de données différents, voyez s'il n'est pas possible de modifier le type de l'une des colonnes avec la commande alter table...modify.
- Utilisez le type de données de la colonne chaque fois que vous déclarez des variables ou des paramètres de procédure stockée qui seront employés comme arguments de recherche.

- Envisagez de définir des types de données utilisateur. Une fois les définitions créées à l'aide de `sp_addtype`, elles sont utilisables dans des commandes telles que `create table`, `alter table` et `create procedure`, ainsi que dans des déclarations de types de données.
- Pour certaines requêtes dont les incompatibilités de types de données nuisent aux performances, essayez d'effectuer une conversion afin que les index soient utilisables sur l'autre table de la jointure. La section ci-après décrit cette opération.

Conversion imposée sur l'autre côté de la jointure

Si une jointure entre des types de données différents est inévitable et qu'elle nuit aux performances, vous pouvez imposer la conversion vers l'autre partie de la jointure, dans certaines requêtes. Dans la requête suivante, il est impossible d'utiliser un index sur `smallmoney_col` ; aussi la requête procède-t-elle à un balayage de la table `huge_table` :

```
select *
from tiny_table, huge_table
where tiny_table.money_col =
      huge_table.smallmoney_col
```

Les performances s'améliorent si l'index sur `huge_table.smallmoney_col` est utilisable. L'exécution de la fonction `convert` sur la colonne `money` de la petite table permet d'utiliser l'index sur la grande table et un balayage de la petite table est effectué :

```
select *
from tiny_table, huge_table
where convert(smallmoney,tiny_table.money_col) =
      huge_table.smallmoney_col
```

Cette solution de contournement part du principe qu'il n'existe pas dans `tinytable.money_col` de valeurs suffisamment élevées pour risquer de générer des erreurs lors de la conversion au type `smallmoney`. Cependant, si certaines valeurs dépassent le maximum autorisé pour `smallmoney`, vous pouvez ajouter un argument de recherche spécifiant les valeurs maximales admises dans une colonne `smallmoney` :

```
select smallmoney_col, money_col
from tiny_table, huge_table
where convert(smallmoney,tiny_table.money_col) =
      huge_table.smallmoney_col
and tiny_table.money_col <= 214748.3647
```

La conversion de données à virgule flottante et numériques peut modifier la signification de certaines requêtes. La requête suivante compare les nombres entiers et ceux à virgule flottante :

```
select *
  from tab1, tab2
 where tab1.int_column = tab2.float_column
```

Dans la requête ci-dessus, il est impossible d'utiliser un index sur int_column. Cette conversion force l'accès de l'index à tab1, mais renvoie aussi des résultats différents de ceux de la requête qui n'exécute pas la fonction convert :

```
select *
  from tab1, tab2
 where tab1.int_col = convert(int, tab2.float_col)
```

Par exemple, si int_column est égal à 4 et float_column à 4,2, la requête modifiée convertit implicitement celle-ci à 4 et renvoie une ligne que ne retourne pas la requête initiale. Il est alors possible de recourir à la solution décrite plus haut, en ajoutant cette auto-jointure :

```
and tab2.float_col = convert(int, tab2.float_col)
```

Cette opération suppose que toutes les valeurs tab2.float_col peuvent être converties en int sans générer d'erreur de conversion.

Fractionnement des procédures stockées pour une meilleure estimation du coût

L'optimiseur ne peut pas utiliser les statistiques sur l'instruction select finale de la procédure suivante, car il ne peut connaître la valeur de @city que lors de l'exécution :

```
create procedure au_city_names
  @pub_name varchar(30)
as
  declare @city varchar(25)
  select @city = city
  from publishers where pub_name = @pub_name
  select au_lname
    from authors
   where city = @city
```

L'exemple suivant montre la procédure divisée en deux. La première appelle la seconde :

```
create procedure au_names_proc
    @pub_name varchar(30)
as
    declare @city varchar(25)
    select @city = city
        from publishers
        where pub_name = @pub_name
    exec select_proc @city
create procedure select_proc @city varchar(25)
as
    select au_lname
        from authors
        where city = @city
```

Lorsque la seconde procédure s'exécute, Adaptive Server connaît la valeur de `@city` et peut optimiser l'instruction `select`. Bien sûr, si vous modifiez la valeur de `@city` dans la seconde procédure avant qu'elle ne soit utilisée dans l'instruction `select`, l'optimiseur risque de choisir un plan inapproprié, parce qu'il optimise la requête à partir de la valeur de `@city` telle qu'elle est au démarrage de la procédure. Si `@city` prend une valeur différente à chaque exécution de la deuxième procédure, conduisant ainsi à des plans d'exécution de requête très différents, vous devez utiliser `with recompile`.

Unités principales d'estimation des coûts

Lorsque l'optimiseur évalue le coût d'une requête, les deux taux pris en considération sont le coût des E/S physiques, avec la lecture des pages sur le disque, et le coût des E/S logiques, avec la recherche des pages dans le cache de données. L'optimiseur définit à 18 le coût d'une E/S physique et à 2 celui d'une E/S logique. Il s'agit d'unités de coût relatives, qui ne correspondent pas à des unités de temps comme des millisecondes ou des impulsions d'horloge. Ces unités sont employées dans les formules décrites dans ce chapitre, avec les coûts des E/S physiques d'abord, puis les coûts des E/S logiques. Le coût total d'accès à une table est exprimé de la manière suivante :

Coût = toutes les E/S physiques * 18 + toutes les E/S logiques * 2

Outils d'optimisation avancés

Ce chapitre décrit les options de traitement de requêtes qui influencent le choix de l'ordre de jointure, l'index, la taille des E/S et la stratégie de cache.

Sujet	Page
Techniques d'optimisation spéciales	39
Spécification des choix de l'optimiseur	40
Service de journaux asynchrones	54
Spécification de l'ordre des tables dans les jointures	41
Spécification du nombre de tables prises en compte par l'optimiseur	43
Spécification d'un index pour une requête	45
Spécification de la taille des E/S dans une requête	47
Spécification de la stratégie de caches	51
Contrôle des E/S étendues et des stratégies de cache	53
Service de journaux asynchrones	54
Activation et désactivation des jointures par fusion	58
Activation et désactivation de la fermeture transitive de jointures	58
Suggestion d'un degré de parallélisation pour une requête	60
Optimisation de la concurrence d'accès pour les petites tables	62

Techniques d'optimisation spéciales

La lecture des informations présentées dans le volume *Performances et optimisation : concepts de base* vous aidera à comprendre ce chapitre. Utilisez ces outils avec prudence car certains d'entre eux permettent d'annuler les décisions prises par l'optimiseur d'Adaptive Server et peuvent réduire considérablement les performances en cas de mauvaise utilisation. Il est primordial que vous compreniez leur impact sur les performances de vos requêtes et les éventuelles retombées sur le système.

L'optimiseur avancé d'Adaptive Server, basé sur l'évaluation des coûts, fournit d'excellents plans d'exécution de requête dans la plupart des cas. Cependant, quelquefois, l'optimiseur ne choisit pas l'index garantissant des performances optimales ou opte pour un ordre de jointure entraînant des performances en dessous du seuil optimal ; vous devez alors définir les méthodes d'accès pour la requête. Les options décrites dans ce chapitre vous permettent d'effectuer ces opérations.

De plus, lors d'une opération d'optimisation, il est utile de pouvoir observer les effets d'un ordre de jointure, d'une taille d'E/S ou d'une stratégie de cache différents. Certaines de ces options vous permettent de spécifier le traitement de requête ou la stratégie d'accès sans reconfiguration coûteuse.

Adaptive Server fournit des outils et des clauses de requête relatifs à l'optimisation des requêtes et des outils d'analyse de requêtes avancés vous permettant de comprendre les choix de l'optimiseur.

Remarque Ce chapitre propose des solutions de contournement de certains problèmes d'optimisation. Si ces problèmes persistent, contactez le Support Technique de Sybase.

Spécification des choix de l'optimiseur

Adaptive Server vous permet de spécifier les choix d'optimisation en incluant des commandes dans le batch de requête ou dans le texte d'une requête :

- l'ordre des tables dans une jointure,
- le nombre de tables évaluées simultanément pendant l'optimisation des jointures,
- l'index utilisé pour l'accès à une table,
- la taille des E/S,
- la stratégie de cache,
- le degré de parallélisation.

Dans certains cas, l'optimiseur ne choisit pas le plan optimal. Or, comme il arrive que le plan choisi ne soit que légèrement plus coûteux que le plan optimal, vous devez comparer le coût des choix imposés à celui d'un plan moins performant.

Les commandes spécifiant l'ordre de jointure, l'index, la taille des E/S ou la stratégie de cache, associées aux commandes d'affichage des résultats de requêtes telles que `statistics io` et `showplan` vous permettent de déterminer les raisons des choix de l'optimiseur.

Avertissement ! Utilisez avec prudence les options décrites dans ce chapitre. Les plans d'exécution de requête imposés peuvent être impropre dans certains cas et engendrer de très faibles performances. Si vous incluez ces options dans vos applications, veillez à vérifier régulièrement leurs plans d'exécution de requête, leurs statistiques des E/S, ainsi que les autres données relatives aux performances.

Ces options sont généralement destinées à être utilisées comme outils d'optimisation et d'expérimentation, et non comme solutions à long terme aux problèmes d'optimisation.

Spécification de l'ordre des tables dans les jointures

Adaptive Server optimise les ordres de jointure afin de réduire les E/S. Dans la plupart des cas, l'ordre choisi par l'optimiseur ne correspond pas à celui des clauses `from` de votre commande `select`. Pour imposer à Adaptive Server d'interroger des tables dans l'ordre où elles sont répertoriées, utilisez la commande suivante :

```
set forceplan [on|off]
```

L'optimiseur continue à choisir la meilleure méthode d'accès pour chaque table. Si vous utilisez `forceplan` en spécifiant un ordre de jointure, l'optimiseur risque d'utiliser des index sur les tables, différents de ceux qu'il aurait choisis avec un autre ordre de table, ou de ne pas pouvoir utiliser les index existants.

Vous pouvez utiliser cette commande comme aide au débogage si d'autres outils d'analyse de requête indiquent que l'optimiseur n'a peut-être pas choisi l'ordre de jointure optimal. Vérifiez toujours que l'ordre que vous imposez réduit les E/S et les lectures logiques à l'aide de `set statistics io on`, et en comparant les E/S avec et sans `forceplan`.

Si vous utilisez `forceplan`, les contrôles réguliers de maintenance des performances doivent vérifier que les requêtes et les procédures qui l'utilisent requièrent toujours cette option pour améliorer les performances.

Vous pouvez inclure `forceplan` dans le texte des procédures stockées.

`set forceplan` impose uniquement l'ordre de jointure et non le type de jointure. Si aucune commande n'est spécifiée pour le type de jointure ; vous pouvez désactiver les jointures par fusion au niveau de la session ou du serveur.

Pour de plus amples informations, reportez-vous à la section « [Activation et désactivation des jointures par fusion](#) », page 58.

Risques liés à l'utilisation de `forceplan`

Imposer un ordre de jointure comporte les risques suivants :

- Une mauvaise utilisation peut aboutir à des requêtes extrêmement coûteuses. Testez toujours soigneusement la requête à l'aide de `statistics io`, avec et sans `forceplan`.
- La maintenance est nécessaire. Vous devez en effet vérifier régulièrement les requêtes et les procédures stockées qui comportent `forceplan`. De plus, dans les versions ultérieures d'Adaptive Server, les problèmes qui vous ont conduit à imposer des index seront peut-être éliminés ; vous devrez donc contrôler toutes les requêtes utilisant des plans d'exécution imposés à chaque installation d'une nouvelle version.

Avant d'utiliser forceplan

Avant d'utiliser forceplan :

- Vérifiez les informations affichées par showplan pour déterminer si les clés d'index sont utilisées comme prévu.
- Repérez les autres problèmes d'optimisation à l'aide de dbcc traceon(302).
- Exécutez update statistics sur l'index.
- Utilisez update statistics pour ajouter des statistiques relatives aux arguments de recherche dans des clauses de recherche non indexées, particulièrement, pour ceux relatifs aux clés mineures d'un index composé.
- Si la requête joint plus de quatre tables, contrôlez si l'utilisation de set table count améliore l'ordre de jointure.

Reportez-vous à la section « [Spécification du nombre de tables prises en compte par l'optimiseur](#) », page 43.

Spécification du nombre de tables prises en compte par l'optimiseur

Adaptive Server optimise les jointures en prenant en compte les permutations de deux à quatre tables en même temps, comme indiqué dans la section « [Estimation et optimisation des jointures](#) », page 115. Si vous pensez qu'un ordre de jointure incorrect est choisi pour une requête de jointure, vous pouvez utiliser l'option set table count pour augmenter le nombre de tables prises en compte simultanément. La syntaxe est la suivante :

```
set table count valeur_ent
```

Les valeurs correctes sont comprises entre 0 et 8 ; 0 restaure le fonctionnement par défaut.

Par exemple, si vous choisissez l'optimisation de quatre tables simultanément, utilisez :

```
set table count 4
```

`dbcc traceon(310)` indique le nombre de tables examinées simultanément. Reportez-vous à la section « [dbcc traceon\(310\) et coûts du plan d'exécution de requête final](#) », page 199 du guide *Performances et optimisation : contrôle et analyse* pour de plus amples informations.

Plus cette valeur est faible, moins l'optimiseur risque de prendre en compte tous les ordres de jointure possibles. L'augmentation du nombre de tables prises en compte simultanément pendant la définition de l'ordre de jointure peut accroître considérablement le temps d'optimisation d'une requête.

La durée d'optimisation de la requête étant augmentée avec chaque table supplémentaire, l'option `set table count` est plus utile lorsque les gains de temps d'exécution effectifs résultant de l'amélioration de l'ordre de jointure l'emportent sur le temps d'optimisation supplémentaire.

Voici quelques exemples :

- si vous pensez qu'un ordre de jointure amélioré permet de réduire les temps d'exécution et d'optimisation de la requête, en particulier lorsqu'il s'agit de procédures stockées susceptibles d'être exécutées plusieurs fois après l'insertion d'un plan dans leur cache ;
- lors de la sauvegarde des plans abstraits pour une utilisation ultérieure.

Utilisez `statistics time` pour contrôler le temps d'analyse et de compilation et `statistics io` pour vérifier que l'ordre de jointure amélioré réduit les E/S physiques et logiques.

Si l'augmentation de la valeur de `table count` améliore l'optimisation des jointures mais augmente le temps CPU de manière inacceptable, écrivez de nouveau la clause `from` dans la requête en spécifiant les tables dans l'ordre de jointure indiqué dans les informations affichées par `showplan` et utilisez `forceplan` pour lancer la requête. Les contrôles réguliers de maintenance des performances doivent vérifier que l'ordre de jointure que vous imposez continue à améliorer les performances.

Spécification d'un index pour une requête

Vous pouvez spécifier l'index à utiliser pour une requête à l'aide de la clause (index *nom_index*) dans les instructions select, update et delete. Vous pouvez également imposer un balayage de table dans une requête en spécifiant le nom de la table concernée. La syntaxe est la suivante :

```
select liste_sélection
      from nom_table [alias]
            (index {nom_index | nom_table } )
            [, nom_table ...]
      where ...

delete nom_table
      from nom_table [alias]
            (index {nom_index | nom_table })...

update nom_table set nom_col = valeur
      from nom_table [alias]
            (index {nom_index | nom_table })...
```

Par exemple :

```
select pub_name, title
      from publishers p, titles t (index date_type)
      where p.pub_id = t.pub_id
      and type = "business"
      and pubdate > "1/1/93"
```

Spécifier un index dans une requête peut être utile lorsque vous pensez que l'optimiseur choisit un plan d'exécution de requête entraînant des performances en dessous du seuil optimal. Lorsque vous utilisez cette option :

- Vérifiez toujours statistics io pour la requête afin de voir si l'index choisi requiert moins d'E/S que celui choisi par l'optimiseur.
- Testez une plage complète de valeurs valides pour les clauses de requête, surtout lors de l'optimisation de requêtes :
 - Optimisation des requêtes dans des tables avec une mauvaise répartition des données.
 - Exécution de requêtes séquentielles, les méthodes d'accès à ces requêtes étant sensibles à la taille de la plage.

Utilisez cette option seulement après avoir effectué un test vérifiant que la requête s'exécute plus rapidement avec l'option d'index spécifiée.

Une fois que vous avez inclus cette option d'index dans des applications, assurez-vous régulièrement que le plan résultant reste supérieur aux autres choix effectués par l'optimiseur.

Remarque Si un index non clusterisé porte le même nom que la table, l'indication d'un nom de table entraîne l'utilisation de l'index non clusterisé. Vous pouvez forcer un balayage de table à l'aide de select *liste_selection from nom_table (0)*.

Risques

La spécification des index comporte les risques suivants :

- Les modifications de la répartition des données peuvent réduire l'efficacité de l'index imposé par rapport aux autres choix.
- La suppression de l'index implique l'apparition de messages dans les requêtes et les procédures qui indiquent que l'index qu'elles spécifiaient n'existe plus. La requête est optimisée à l'aide de la méthode d'accès disponible la plus performante.
- La maintenance augmente, puisque toutes les requêtes utilisant cette option doivent être contrôlées régulièrement. De plus, dans les versions ultérieures d'Adaptive Server, les problèmes qui vous ont conduit à imposer des index seront peut-être éliminés ; vous devrez donc contrôler toutes les requêtes utilisant des index imposés à chaque installation d'une nouvelle version.
- L'index doit déjà exister au moment où la requête qui l'utilise est optimisée. En effet, vous ne pouvez pas créer un index pour l'utiliser ensuite dans une requête du même batch.

Opérations précédant la spécification d'un index

Avant la spécification d'un index dans les requêtes :

- Recherchez le message « Keys are » dans les informations affichées par showplan afin de vous assurer que les clés d'index sont utilisées comme prévu.
- Repérez les autres problèmes d'optimisation à l'aide de dbcc traceon(302).
- Exécutez update statistics sur l'index.
- Si l'index est un index composé, exécutez update statistics dans les clés mineures de l'index, si elles sont utilisées comme arguments de recherche. Vous pouvez ainsi améliorer considérablement les estimations de coût de l'optimiseur. La création de statistiques sur d'autres colonnes souvent utilisées dans les clauses de recherche peut également améliorer l'estimation.

Spécification de la taille des E/S dans une requête

Si votre Adaptive Server est configuré pour des E/S étendues dans le cache de données par défaut ou dans des caches de données nommés, l'optimiseur peut utiliser des E/S étendues pour :

- les requêtes qui balaient des tables entières,
- les requêtes séquentielles utilisant des index clusterisés, comme les requêtes utilisant $>$, $<$, $> x$ et $< y$, between et like "chaîne_caractère%",
- les requêtes qui analysent un grand nombre de pages d'index.

Si le cache utilisé par une table ou un index est configuré pour des E/S de 16 Ko, une seule opération d'E/S peut lire jusqu'à huit pages simultanément. Chaque cache de données nommé peut comporter plusieurs zones ayant chacune une taille d'E/S différente. Lorsque vous spécifiez la taille des E/S dans une requête, celles-ci s'effectuent dans la zone configurée pour cette taille. Pour plus d'informations sur la configuration de caches de données nommés, reportez-vous au *Guide d'administration système*.

Pour indiquer une taille d'E/S différente de celle choisie par l'optimiseur, ajoutez la spécification prefetch à la clause index d'une instruction select, delete ou update. La syntaxe est la suivante :

```
select liste_sélection  
      from nom_table  
        ( [index {nom_index | nom_table} ]  
          prefetch taille)  
        [, nom_table ...]  
      where ...
```

```
delete nom_table from nom_table  
  ( [index {nom_index | nom_table} ]  
    prefetch taille)
```

...

```
update nom_table set nom_col = valeur  
      from nom_table  
        ( [index {nom_index | nom_table} ]  
          prefetch taille)
```

...

La taille de prélecture correcte dépend de la taille de la page. S'il n'existe aucune zone de la taille spécifiée dans le cache de données utilisé par l'objet, l'optimiseur choisi la taille disponible la plus appropriée.

S'il existe un index clusterisé sur au_lname, cette requête effectue une E/S de 16 Ko pendant l'analyse des pages de données :

```
select *  
  from authors (index au_names prefetch 16)  
    where au_lname like "Sm%"
```

Si une requête effectue des E/S étendues et que vous souhaitez vérifier ses performances d'E/S avec des E/S de 2 Ko, vous pouvez spécifier une taille de 2 Ko :

```
select type, avg(price)  
      from titles (index type_price prefetch 2)  
        group by type
```

Remarque Les références aux E/S étendues sont sur un serveur de pages logiques dont la taille est de 2 Ko. Si la taille des pages de votre serveur est de 8 Ko, l'unité de base de l'E/S sera de 8 Ko. Si elle est de 16 Ko, l'unité de base de l'E/S sera de 16 Ko.

Type d'index et E/S étendues

La spécification d'une taille d'E/S avec prefetch peut avoir des répercussions sur les pages de données ainsi que sur les pages d'index au niveau feuille. Le [tableau 3-1](#) illustre ces conséquences.

Tableau 3-1 : Méthodes d'accès et prélecture

Méthode d'accès	E/S étendues effectuées sur
Balayage de table	Pages de données
Index clusterisé	Pages de données seulement pour tables verrouillées au niveau de toutes les pages Pages de données et pages d'index de niveau feuille pour des tables verrouillées au niveau des pages de données seulement
Index non clusterisé	Pages de données et pages de niveau feuille d'index non clusterisé

showplan indique la taille des E/S utilisées pour les pages de données et les pages de niveau feuille.

Reportez-vous à la section [« Messages de taille des E/S », page 117](#) du guide *Performances et optimisation : contrôle et analyse* pour de plus amples informations.

Omission de la spécification de `prefetch`

En général, lorsque vous spécifiez une taille d'E/S dans une requête, l'optimiseur l'intègre au plan de la requête. Cependant, il arrive que cette spécification ne soit pas prise en compte, au niveau de la requête globale ou d'une seule requête d'E/S étendue.

Les E/S étendues ne peuvent pas être utilisées pour une requête si :

- Le cache n'est pas configuré pour des E/S de la taille spécifiée. L'optimiseur utilise la taille disponible la plus appropriée.
- `sp_cachestrategy` a été utilisée pour désactiver les E/S étendues pour la table ou l'index.

Les E/S étendues ne peuvent pas être utilisées pour un seul buffer si :

- certaines pages incluses dans cette requête d'E/S se trouvent dans une autre zone du cache ;
- la page se trouve sur le premier extent d'une unité d'allocation. Cet extent contient la page d'allocation pour l'unité d'allocation et seulement sept pages de données ;
- aucun buffer n'est disponible dans la zone correspondant à la taille d'E/S requise.

Lorsqu'il n'est pas possible d'effectuer une E/S étendue, Adaptive Server réalise des E/S de 2 Ko sur la page nécessaire ou les pages de l'extent nécessaires à la requête.

Pour savoir si la spécification de prélecture est prise en compte, utilisez `showplan` pour afficher le plan d'exécution de requête et `statistics io` pour afficher les résultats des E/S de la requête. La procédure système `sp_sysmon` affiche les E/S étendues demandées et refusées pour chaque cache.

Reportez-vous à la section « [Gestion des caches de données](#) », page 294 du guide *Performances et optimisation : contrôle et analyse*.

set prefetch on

Par défaut, une requête utilise des E/S étendues lorsqu'une zone d'E/S étendues est configurée et que l'optimiseur estime que l'utilisation d'E/S de ce type réduirait le coût de la requête. Pour désactiver les E/S étendues le temps d'une session, utilisez :

```
set prefetch off
```

Pour les réactiver, utilisez la commande suivante :

```
set prefetch on
```

Si les E/S étendues sont désactivées pour un objet à l'aide de la procédure système `sp_cachestrategy`, la commande `set prefetch on` ne permet pas de les réactiver.

Si les E/S étendues sont désactivées pour une session à l'aide de `set prefetch off`, vous ne pouvez pas les réactiver en spécifiant une taille de prélecture dans une instruction `select`, `delete` ou `insert`.

La commande `set prefetch` prend effet dans le batch où elle est exécutée ; vous pouvez ainsi l'inclure dans une procédure stockée pour qu'elle s'applique à l'exécution des requêtes de la procédure.

Spécification de la stratégie de caches

Pour les requêtes qui balaien les pages de données d'une table ou le niveau feuille d'un index non clusterisé (requêtes restreintes), l'optimiseur Adaptive Server choisit l'une des deux stratégies de remplacement de cache suivantes : la stratégie de lecture-élimination (MRU) ou la stratégie LRU.

Reportez-vous à la section « [Présentation des stratégies de caches](#) », [page 188](#) du guide *Performances et optimisation : concepts de base* pour plus d'informations sur ces stratégies.

L'optimiseur est susceptible de choisir la stratégie de lecture-élimination (MRU) pour :

- toute requête effectuant des balayages de tables ;
- toute requête à intervalle utilisant un index clusterisé ;
- toute requête couverte balayant le niveau feuille d'un index non clusterisé ;
- une table interne dans une jointure à boucles imbriquées, si cette table est plus grande que le cache ;
- la table externe d'une jointure à boucles imbriquées, puisqu'elle n'est lue qu'une fois ;
- les deux tables dans une jointure par fusion.

Vous pouvez appliquer la stratégie de cache aux objets :

- en spécifiant `lru` ou `mru` dans une instruction `select`, `update` ou `delete` ;
- en utilisant `sp_cachestrategy` pour désactiver ou réactiver la stratégie `mru`.

Si vous spécifiez la stratégie MRU et qu'une page se trouve déjà dans le cache de données, cette page est placée à l'extrémité MRU du cache et non dans le marqueur de vidage.

La spécification de la stratégie de caches concerne uniquement les pages de données et les pages de niveau feuille des index. Les pages racine et les pages intermédiaires utilisent toujours la stratégie LRU.

Dans les instructions **select**, **delete** et **update**

Vous pouvez utiliser lru ou mru (lecture-élimination) dans une commande select, delete ou update pour spécifier la taille d'E/S pour la requête :

```
select liste_sélection
      from nom_table
            (index nom_index prefetch taille [lru|mru])
            [, nom_table ...]
      where ...
```

```
delete nom_table from nom_table (index nom_index
                                         prefetch taille [lru|mru]) ...
```

```
update nom_table set nom_col = valeur
      from nom_table (index nom_index
                     prefetch taille [lru|mru]) ...
```

La requête suivante ajoute la stratégie de remplacement LRU à la spécification d'E/S de 16 Ko.

```
select au_lname, au_fname, phone
      from authors (index au_names prefetch 16 lru)
```

Pour plus d'informations sur la spécification d'une taille prefetch, reportez-vous à la section « [Spécification de la taille des E/S dans une requête](#) », page 47.

Contrôle des E/S étendues et des stratégies de cache

Les bits d'état de la table sysindexes indiquent si la stratégie de prélecture des E/S étendues ou celle de lecture-élimination (MRU) doit s'appliquer à une table ou à un index. Par défaut, les deux stratégies sont activées. Pour les désactiver ou les réactiver, utilisez la procédure système sp_cachestrategy. La syntaxe est la suivante :

```
sp_cachestrategy nom_base, [nomPropriétaire.]nom_table
[, nom_index | "text only" | "table only"
 [, { prefetch | mru }, { "on" | "off"}]]
```

Cette commande désactive la stratégie de prélecture des E/S étendues pour l'index au_name_index de la table authors :

```
sp_cachestrategy pubtune,
authors, au_name_index, prefetch, "off"
```

La commande suivante réactive la stratégie de lecture-élimination pour la table titles :

```
sp_cachestrategy pubtune,
titles, "table only", mru, "on"
```

Seul un administrateur système ou le propriétaire de l'objet peut modifier ou visualiser l'état de la stratégie de cache d'un objet.

Informations sur les stratégies de cache

Pour visualiser la stratégie de cache active pour un objet donné, exécutez sp_cachestrategy en indiquant le nom de la base de données et celui de l'objet :

sp_cachestrategy pubtune, titles	object name	index name	large IO	MRU
	titles	NULL	ON	ON

Les informations affichées par showplan indiquent la stratégie de cache utilisée pour chaque objet, y compris les tables de travail.

Service de journaux asynchrones

ALS accroît l'évolutivité dans Adaptive Server et offre un débit plus élevé dans les sous-systèmes de journalisation aux systèmes multiprocesseur symétriques haut de gamme.

Vous ne pouvez pas utiliser ALS si vous disposez de moins de 4 moteurs. Si vous essayez d'activer ALS avec moins de 4 moteurs en ligne, un message d'erreur s'affiche.

Activation d'ALS

Vous pouvez activer, désactiver ou configurer ALS à l'aide de la procédure stockée `sp_dboption`.

```
sp_dboption <db Name>, "async log service",
"true|false"
```

Exécution d'un point de reprise

Après avoir lancé `sp_dboption`, vous devez exécuter un checkpoint dans la base de données pour laquelle vous paramétrez l'option ALS :

```
sp_dboption "mydb", "async log service", "true"
use mydb
checkpoint
```

Vous pouvez utiliser `checkpoint` pour identifier une ou plusieurs bases de données ou utiliser une clause `all`.

```
checkpoint [all | [nom_base[, nom_base[, nom_base.....]]]]
```

Désactivation d'ALS

Avant de désactiver ALS, assurez-vous qu'il n'y a pas d'utilisateurs actifs dans la base de données. S'il y en a, un message d'erreur s'affiche lorsque vous exécutez le point de reprise :

```
sp_dboption "mydb", "async log service", "false"
use mydb
checkpoint
-----
Error 3647: Cannot put database in single-user mode.
Wait until all users have logged out of the database
and issue a CHECKPOINT to disable "async log
service".
```

S'il n'y a pas d'utilisateurs actifs dans la base de données, l'exemple suivant permet de désactiver ALS :

```
sp_dboption "mydb", "async log service", "false"
use mydb
checkpoin]
-----
```

Affichage d'ALS

Vous pouvez voir si ALS est activé dans une base de données spécifique en lançant la commande `sp_helpdb`.

```
sp_helpdb "mydb"
-----
mydb           3.0 MB sa          2
    July 09, 2002
    select into/bulkcopy/pllsort, trunc log on
    chkpt,
    async log service
```

Pour plus d'informations sur ces procédures stockées, reportez-vous à la section « [Procédures système modifiées](#) », page 57.

Description de l'architecture du cache de journaux utilisateur

L'architecture de journalisation d'Adaptive Server propose un cache de journaux utilisateur ou ULC, grâce auquel chaque tâche dispose de son propre cache de journaux. Aucune autre tâche ne peut être écrite dans ce cache et la tâche continue d'écrire dans le cache de journaux utilisateur chaque fois qu'une transaction génère un enregistrement de journal.

Lorsque la transaction sauvegardée ou annulée ou que le cache de journaux utilisateur est plein, ce dernier est vidé dans le cache de journaux commun, qui est partagé par toutes les tâches en cours et est ensuite écrit sur le disque.

Le vidage de l'ULC est la première étape d'une opération de sauvegarde ou d'annulation. Il fait appel aux étapes suivantes, qui sont toutes susceptibles de provoquer des retards ou d'augmenter les conflits :

- 1 Placement d'un verrou sur la dernière page du journal.
- 2 Allocation de nouvelles pages de journal si nécessaire.
- 3 Copie des enregistrements du journal de l'ULC dans le cache de journaux.

Pour pouvoir exécuter les étapes 2 et 3, vous devez placer un verrou sur la dernière page du journal, afin d'empêcher d'autres tâches d'écrire dans le cache de journaux ou d'exécuter des opérations de sauvegarde ou d'annulation.

4 Videz le cache de journaux sur le disque.

L'étape 4 exige un balayage répété du cache de journaux afin d'exécuter des commandes write sur les buffers modifiés.

Ce balayage répété peut provoquer un conflit au niveau du verrou d'attente du cache de journaux auquel le journal est lié. Lorsque la charge de transaction est importante le conflit peut être significatif.

ALS

Vous pouvez activer ALS sur n'importe quelle base de données respectant au moins l'une des exigences de performances suivantes, tant que votre système dispose d'au moins 4 moteurs en ligne :

- Nombreux conflits au niveau de la dernière page du journal.

La dernière page de journal est en conflit lorsque le résultat de la commande `sp_sysmon` dans la section « Task Management Report » renvoie une valeur très élevée. Par exemple :

Tableau 3-2 : Page de journal en conflit

Gestion des tâches	per sec	per xact	décompte	% du total
Conflit de sémaphores de journal	58,0	0,3	34 801	73,1

- Bande passante sous-utilisée dans le device de journal

Remarque Vous ne devez utiliser ALS que lorsque vous identifiez une base de données avec des exigences élevées en matière de transactions, dans la mesure où la définition d'ALS pour plusieurs bases de données risque de provoquer des variations inattendues au niveau du débit et des temps de réponses. Si vous voulez configurer ALS sur plusieurs bases de données, vérifiez d'abord que le débit et les temps de réponse sont satisfaisants.

Utilisation d'ALS

Deux threads balaiennent les tampons modifiés (tampons pleins de données qui n'ont pas encore été écrites sur le disque), copient les données et les écrivent dans le journal. Il s'agit des threads suivants :

- le videur de cache de journaux utilisateur (ULC) ;
- le scripteur de journaux.

Videur ULC

Le videur ULC est un thread de tâche système conçu pour vider le cache de journaux utilisateur dans une tâche du cache de journaux général.

Lorsque la tâche est prête à être sauvegardée, l'utilisateur entre une requête de commit dans la file d'attente du videur. Chaque entrée dispose d'un pointeur, via laquelle le videur ULC peut accéder à l'ULC de la tâche qui a mis la requête dans la file d'attente. La tâche du videur ULC surveille en permanence la file d'attente du videur, en supprimant des requêtes de la queue et en les traitant en vidant les pages ULC dans le cache de journaux.

Scripteur de journaux

Une fois que le videur ULC a fini de vider les pages ULC dans le cache de journaux, il place la requête de la tâche dans une file d'attente de réveil. Le scripteur de journaux contrôle la chaîne de tampons modifiés dans le cache de journaux et envoie une commande d'écriture s'il trouve des tampons modifiés. Il surveille en outre la file d'attente de réveil à la recherche de tâches dont les pages sont toutes écrites sur le disque. Dans la mesure où le scripteur contrôle la chaîne de tampons modifiés, il sait quand un tampon est prêt à être écrit sur le disque.

Procédures système modifiées

Deux procédures stockées sont modifiées pour activer ALS :

- sp_dboption ajoute une option qui permet d'activer et de désactiver ALS.
- sp_helpdb ajoute une colonne afin d'afficher ALS.

Pour plus d'informations sur l'utilisation de ces procédures stockées, reportez-vous au *Manuel de référence*.

Activation et désactivation des jointures par fusion

Par défaut, les jointures par fusion sont désactivées au niveau du serveur. Lorsque les jointures par fusion sont désactivées, le serveur ne tient compte que des jointures par boucles imbriquées et les jointures par fusion ne sont pas prises en considération. Pour activer les jointures par fusion au niveau du serveur, définissez `enable sort-merge joins and JTC` à 1.

Ce paramètre active la fermeture transitive de jointures.

La commande `set sort_merge on` permet de prendre le pas sur le serveur pour utiliser des jointures par fusion au cours d'une session ou pour une procédure stockée.

Pour activer les jointures par fusion, utilisez :

```
set sort_merge on
```

Pour désactiver les jointures par fusion, utilisez :

```
set sort_merge off
```

Pour plus d'informations sur la configuration des jointures par fusion, reportez-vous au *Guide d'administration système*.

Activation et désactivation de la fermeture transitive de jointures

Par défaut, la fermeture transitive de jointures est désactivée au niveau du serveur, car elle peut augmenter le temps d'optimisation. Vous pouvez activer la fermeture transitive de jointures au niveau de la session à l'aide de la commande `set jtc on`. La commande au niveau de la session prend le pas sur le paramétrage au niveau du serveur pour le paramètre de configuration `enable sort-merge joins and JTC`.

Pour les requêtes qui s'exécutent rapidement, même lorsqu'elles concernent plusieurs tables, la fermeture transitive de jointures peut augmenter le temps d'optimisation sans beaucoup améliorer le coût d'exécution. Par exemple, avec la fermeture transitive de jointures activée pour cette requête, le nombre de jointures possibles est multiplié pour chaque table supplémentaire :

```
select * from t1, t2, t3, t4, ... tN  
where t1.c1 = t2.c1  
and t1.c1 = t3.c1  
and t1.c1 = t4.c1  
...  
and t1.c1 = tN.c1
```

Toutefois, pour les jointures dans les très grosses tables, le temps d'optimisation supplémentaire pour estimer le coût des ordres de jointure ajoutées peut générer un ordre de jointure qui améliore grandement le temps de réponse.

Vous pouvez utiliser la commande `set statistics time` pour voir le temps nécessaire à l'optimisation de la requête. Si l'exécution de requêtes avec `set jtc on` augmente considérablement le temps d'optimisation, mais améliore également l'exécution d'une requête en sélectionnant un meilleur ordre de jointures, vérifiez les résultats de `showplan` ou de `dbcc traceon(302, 310)`. Ajoutez explicitement les ordres de jointure utiles au texte de la requête. Vous pouvez exécuter la requête sans fermeture transitive de jointures et améliorer ainsi le temps d'exécution, sans subir l'augmentation du temps d'optimisation nécessaire à l'examen de tous les ordres possibles de jointure générés par la fermeture transitive de jointures.

Vous pouvez aussi activer la fermeture transitive de jointure et sauvegarder des plans abstraits pour les requêtes qui s'en trouvent améliorées. Si vous exécutez alors ces requêtes en activant le chargement des plans sauvegardés, le plan d'exécution sauvegardé est utilisé pour optimiser la requête, ce qui réduit considérablement le temps d'optimisation.

Pour plus d'informations sur l'utilisation des plans abstraits, reportez-vous aux chapitres relatifs aux plans abstraits de ce guide.

Pour plus d'informations sur la configuration des fermetures transitives de jointures, reportez-vous au *Guide d'administration système*.

Suggestion d'un degré de parallélisation pour une requête

Les extensions `parallel` et `degré_parallélisation` associées à la clause `from` d'une commande `select` permettent aux utilisateurs de restreindre le nombre de processus de travail utilisés dans un balayage.

Pour effectuer un balayage de partition parallel, le `degré_parallélisation` doit être égal ou supérieur au nombre de partitions. Pour un balayage d'index parallèle, spécifiez n'importe quelle valeur pour le `degré_parallélisation`.

La syntaxe de l'instruction `select` est la suivante :

```
select...
[from {tablename}
  [(index nom_index
    [parallel [degré_parallélisation | 1]
    [prefetch taille] [lru|mru]]],
  {tablename} [(nom_index]
    [parallel [degré_parallélisation | 1]
    [prefetch taille] [lru|mru]])] ...]
```

Le [tableau 3-3](#) indique comment combiner les mots-clés `index` et `parallel` afin d'effectuer des balayages en série ou parallèles.

Tableau 3-3 : Conseils de l'optimiseur pour exécuter des balayages en série ou parallèles

Pour spécifier ce type de balayage :	Utilisez la syntaxe suivante :
Balayage d'une partition en parallèle	(<code>index nom_table parallel N</code>)
Balayage d'index parallèle	(<code>index nom_index parallel N</code>)
Balayage de tables en série	(<code>index nom_table parallel 1</code>)
Balayage d'index en série	(<code>index nom_index parallel 1</code>)
En parallèle, l'optimiseur choisissant la table ou l'index à lire	(<code>parallel N</code>)
En série, l'optimiseur choisissant la table ou l'index à lire	(<code>parallel 1</code>)

Lorsque vous spécifiez le degré de parallélisation d'une table dans une jointure par fusion, celui-ci affecte le degré de parallélisation utilisé tant pour le balayage de la table que pour la jointure par fusion.

Vous ne pouvez pas utiliser l'option `parallel` si vous avez désactivé le traitement en parallèle au niveau de la session en utilisant la commande `set parallel_degree 1` ou au niveau du serveur en utilisant le paramètre de configuration `parallel degree`. L'option `parallel` ne peut pas annuler ces paramètres.

Si vous spécifiez un `degré_parallélisation` supérieur au degré de parallélisation maximum configuré, Adaptive Server ignore le conseil.

L'optimiseur ne tient pas compte des conseils qui spécifient un degré de parallélisation si l'une des conditions suivantes est vraie :

- La clause `from` est utilisée dans la définition d'un curseur.
- L'option `Parallel` est utilisée dans la clause `from` d'un des blocs de requête interne d'une sous-requête et l'optimiseur ne transfère pas la table dans le bloc de requête externe lors de la mise à plat de la sous-requête.
- La table est une vue, une table système ou une table virtuelle.
- La table est la table interne d'une jointure externe.
- La requête spécifie `exists`, `min` ou `max` pour la table.
- La valeur 1 est attribuée au paramètre de configuration `max scan parallel degree`.
- Un index clusterisé non partitionné est spécifié ou est la seule option parallèle.
- Un index non clusterisé est couvert.
- La requête est traitée selon la stratégie OR.

Pour plus d'informations sur la stratégie OR, reportez-vous à la section « [Méthodes d'accès et évaluation du coût des clauses or et in](#) », [page 94](#).

- L'instruction `select` est utilisée pour une mise à jour ou une insertion.

Exemples de clauses `parallel` au niveau requête

Pour spécifier le degré de parallélisation pour une seule requête, insérez `parallel` après le nom de la table. Dans cet exemple, la requête s'exécute en mode série :

```
select * from titles (parallel 1)
```

L'exemple ci-après spécifie l'index à utiliser dans la requête et fixe le degré de parallélisation à 5 :

```
select * from titles
  (index title_id_clix parallel 5)
where ...
```

Pour imposer un balayage de table, utilisez le nom de la table, et non celui de l'index.

Optimisation de la concurrence d'accès pour les petites tables

Pour les tables verrouillées au niveau des données seulement (tables DOL) de 15 pages ou moins, Adaptive Server ne choisit pas un balayage de table s'il existe un index utile sur la table. Il choisirra toujours l'index le moins coûteux qui correspond à tout argument de recherche optimisable de la requête. Le verrouillage requis pour un balayage d'index fournit une concurrence supérieure et réduit le risque d'interblocage, bien qu'un nombre légèrement supérieur d'E/S que pour un balayage de table puisse être nécessaire.

Si la concurrence sur les petites tables ne pose pas de problème, et que vous préfériez optimiser les E/S, vous pouvez désactiver cette optimisation à l'aide de `sp_chgattribute`. Cette commande désactive l'optimisation de concurrence d'accès de la table :

```
sp_chgattribute tiny_lookup_table,  
    "concurrency_opt_threshold", 0
```

Une fois l'optimisation de concurrence désactivée, l'optimiseur peut sélectionner des balayages de tables s'ils requièrent moins d'E/S.

Vous pouvez aussi augmenter le seuil d'optimisation de concurrence d'une table. Cette commande règle le seuil d'optimisation de concurrence pour une table à 30 pages :

```
sp_chgattribute lookup_table,  
    "concurrency_opt_threshold", 30
```

La valeur maximum du seuil d'optimisation de concurrence est 32 767. La valeur -1 force l'optimisation de concurrence d'accès, quelle que soit la taille de la table. Ceci peut être utile si un balayage de table est préféré à un accès indexé et que le verrouillage provoque une augmentation de conflit ou des interblocages.

Le paramétrage actuel est stocké dans `systabstats.conopt_thld` et imprimé dans la sortie `optdiag`.

Modification du plan de verrouillage

L'optimisation de la concurrence d'accès ne concerne que les tables DOL. Le [tableau 3-4](#) illustre l'impact de la modification du plan de verrouillage.

Tableau 3-4 : Effets de alter table sur les paramètres d'optimisation de concurrence d'accès

Modification du plan de verrouillage	Effet sur les valeurs stockées
de toutes les pages à un verrouillage des données seulement	paramétré à 15, paramétrage par défaut
des données seulement à toutes les pages	paramétré à 0
d'un plan de données seulement à un autre	valeur définie retenue

Outils d'optimisation de la requête

Ce chapitre présente les outils dont vous disposez pour optimiser vos requêtes.

Sujet	Page
Présentation	65
Interaction entre les outils	67
Outils et traitement des requêtes	68

Les outils présentés dans ce chapitre sont décrits plus en détail dans les chapitres qui suivent.

Présentation

Adaptive Server fournit des outils de diagnostic et d'information qui vous aident à optimiser vos requêtes et donc à améliorer leurs performances, parmi lesquels :

- Un choix d'outils pour contrôler ou estimer la taille des tables et des index. Ces outils sont décrits au [Chapitre 11, « Détermination de la taille des tables et des index »](#) du guide *Performances et optimisation : concepts de base*.
- `set statistics io` on affiche le nombre de lectures et d'écritures logiques et physiques requises pour chaque table dans une requête. Si les limites d'utilisation des ressources sont activées, cette commande affiche également le coût total réel des E/S. `set statistics io` est décrit au [Chapitre 4, « Utilisation des commandes set statistics »](#), du guide *Performances et optimisation : contrôle et analyse*.
- `set showplan` on affiche les étapes effectuées pour chaque requête dans un batch. Cette commande est souvent utilisée avec `set noexec on`, en particulier pour des requêtes qui renvoient un grand nombre de lignes.

Reportez-vous au [Chapitre 5, « Utilisation de set showplan »](#), du guide *Performances et optimisation : contrôle et analyse*.

- `set statistics subquerycache` on affiche, pour chaque sous-requête, le nombre de présences et d'absences dans le cache, ainsi que le nombre de lignes du cache.

Pour obtenir des exemples, reportez-vous à la section « [Mise en mémoire cache des résultats de la sous-requête](#) », page 152.

- `set statistics time` on affiche le temps pris par l'analyse et la compilation de chaque commande.

Reportez-vous à la section « [Contrôle de la durée de compilation et d'exécution](#) », page 66 du guide *Performances et optimisation : contrôle et analyse* pour de plus amples informations.

- `dbcc traceon (302)` et `dbcc traceon(310)` proposent des informations complémentaires sur le choix d'un plan particulier, informations souvent utilisées lorsque l'optimiseur choisit un plan qui semble incorrect.

Reportez-vous au [Chapitre 7, « Optimisation avec dbcc traceon »](#), du guide *Performances et optimisation : contrôle et analyse*.

- L'utilitaire `optdiag` affiche des statistiques sur les tables, les index et les colonnes.

Reportez-vous au [Chapitre 6, « Affichage des tables de statistiques avec optdiag »](#), du guide *Performances et optimisation : contrôle et analyse*.

- Le [Chapitre 3, « Outils d'optimisation avancés »](#), du guide *Performances et optimisation : optimiseur et plans abstraits* décrit les outils utilisables pour appliquer le choix de l'index, l'ordre de jointure et les autres options d'optimisation des requêtes. Parmi ces outils :
 - `set forceplan` force la requête à utiliser les tables dans l'ordre spécifié par la clause `from`.
 - `set table count` augmente le nombre de tables prises en compte en une seule fois par l'optimiseur lorsqu'il détermine l'ordre de jointure.
 - les clauses `select`, `delete`, `update` avec (`index...prefetch...` `mru_lru...parallel`) spécifient l'index, la taille d'E/S ou la stratégie de cache à utiliser pour la requête.
 - `set prefetch` définit prefetch pour expérimenter l'optimisation des requêtes.
 - `set sort_merge` désactive les jointures tri-fusion.

- set parallel_degree indique le degré de parallélisation pour une requête.
- sp_cachestrategy définit les bits d'état pour activer ou désactiver les stratégies de prélecture et de lecture-élimination du cache.

Interaction entre les outils

Le résultat de commandes telles que showplan, statistics io, etc. est généré au cours de l'exécution des procédures stockées. Les procédures système utilisées pour vérifier la structure des tables ou les index pendant les tests des stratégies d'optimisation peuvent générer des sorties volumineuses lors de l'impression des informations de diagnostic. Vous pouvez choisir de créer des copies de sauvegarde des schémas des tables et des informations sur les index ou d'utiliser des fenêtres distinctes pour l'exécution de procédures système telles que sp_helpindex.

Pour des requêtes et des batchs longs, vous pouvez choisir de sauvegarder le résultat des commandes showplan et statistics io dans des fichiers. Pour ce faire, paramétrez l'indicateur « echo input » sur isql. La syntaxe est la suivante :

```
isql -P mot_de_passe -e -i fichier_entrée -o fichier_résultat
```

Utilisation de **showplan** avec **noexec**

showplan est souvent utilisé avec set noexec on, ce qui empêche l'exécution des instructions SQL. Exécutez showplan ou d'autres commandes set, avant noexec. Dès que vous émettez la commande set noexec on, la seule commande qu'Adaptive Server peut exécuter est set noexec off. L'exemple suivant indique l'ordre correct :

```
set showplan on
set noexec on
go
select au_lname, au_fname
      from authors
     where au_id = "A137406537"
go
```

noexec et statistics io

Alors que la combinaison des commandes showplan et noexec est utile, noexec interrompt l'exécution de statistics io. En effet, la commande statistics io cherche les E/S disque réelles mais lorsque noexec est activée, aucune E/S n'est effectuée ; par conséquent, les rapports ne sont pas imprimés.

Outils et traitement des requêtes

Plusieurs de ces outils (les commandes set, par exemple) ont une incidence sur les décisions prises par l'optimiseur. showplan et dbcc traceon(302, 310) montrent la prise de décision de l'optimiseur. dbcc traceon(302,310) affiche des informations intermédiaires en cours d'analyse et dbcc traceon(310) imprime les statistiques du plan final. showplan indique la décision finale sur les méthodes d'accès et l'ordre de jointure.

statistics io et statistics time donnent des informations sur le mode d'exécution de la requête : statistics time mesure le temps écoulé entre l'étape d'analyse jusqu'à la fin de la requête. statistics io imprime les E/S réellement effectuées pendant l'exécution de la requête.

noexec permet d'obtenir des informations telles que les sorties de showplan ou de dbcc traceon(302,310) sans exécuter réellement la requête.

Méthodes d'accès et évaluation des coûts de requête sur une seule table

Ce chapitre présente les méthodes utilisées par Adaptive Server pour accéder aux lignes des tables. Il décrit les différents types de requêtes qui ne se réfèrent qu'à une seule table, ainsi que les méthodes d'accès possibles et les coûts associés.

Sujet	Page
Coût du balayage de table	71
De la ligne à la page	75
Evaluation du coût d'accès à un index	78
Evaluation du coût des requêtes utilisant la clause order by	86
Méthodes d'accès et évaluation du coût des clauses or et in	94
Optimisation des agrégats	99
Mise à jour des opérations	102

Le [Chapitre 2, « Présentation de l'optimiseur »](#), explique comment, à l'aide d'arguments de recherche et de clauses de jointure, l'optimiseur évalue le nombre de lignes que doit renvoyer la requête. Il décrit également la manière dont l'optimiseur utilise ces estimations et d'autres statistiques pour déterminer le nombre de pages à lire dans le cadre de la requête, ainsi que le nombre des E/S physiques et logiques nécessaires.

Ce chapitre traite des requêtes qui ne se réfèrent qu'à une seule table.

Pour les requêtes portant sur plusieurs tables, reportez-vous au [Chapitre 6, « Méthodes d'accès et coût des requêtes pour des jointures et des sous-requêtes »](#).

Pour les requêtes parallèles, reportez-vous au [Chapitre 8, « Optimisation des requêtes parallèles »](#).

Ce chapitre contient des informations sur le traitement des requêtes, utilisables de plusieurs manières :

- Il fournit une présentation générale des méthodes d'accès qu'utilise Adaptive Server pour traiter diverses requêtes, accompagnée d'illustrations et d'exemples. Ces informations vous aideront à comprendre comment sont exécutés certains types de requête et comment améliorer les performances en ajoutant des index ou des statistiques sur les colonnes apparaissant dans les requêtes.
- Ce chapitre décrit également les opérations mises en oeuvre par l'optimiseur pour estimer le volume des E/S physiques et logiques effectuées dans les requêtes. Ces descriptions vous seront utiles pour déterminer si le nombre d'E/S et les temps de réponse sont raisonnables pour une requête donnée. Elles sont utilisables avec les outils d'optimisation des performances suivants :
 - `optdiag` permet d'afficher les statistiques relatives aux tables, aux index et aux valeurs des colonnes.

Reportez-vous au [Chapitre 6, « Affichage des tables de statistiques avec optdiag »](#) du volume Contrôle et analyse.

- `showplan` affiche la méthode d'accès (balayage de table, balayage d'index, type de stratégie OR, etc.) pour une requête.

Reportez-vous au [Chapitre 5, « Utilisation de set showplan »](#) du guide *Performances et optimisation : contrôle et analyse*.

- `statistics io` affiche les E/S physiques et logiques pour chaque table d'une requête.
- Fournit des formules détaillées, très similaires aux formules utilisées réellement par Adaptive Server. Ces formules devraient être utilisées avec les outils d'optimisation décrit dans le guide *Performances et optimisation : contrôle et analyse*.
- Avec `optdiag`, vous pouvez afficher les statistiques nécessaires à l'application des formules. Reportez-vous au [Chapitre 6, « Affichage des tables de statistiques avec optdiag »](#).
- `dbcc traceon(302)` affiche les tailles, les densités, les sélectivités et les taux de clusterisation sur lesquels s'appuient les estimations du nombre d'E/S logiques, tandis que `dbcc traceon(310)` indique le coût final de la requête pour chaque table, avec une estimation des E/S physiques. Reportez-vous au [Chapitre 7, « Optimisation avec dbcc traceon »](#).

Bien souvent, vous n'aurez besoin de ces formules que pour résoudre des problèmes de mise au point des requêtes. Il peut en effet être intéressant de savoir pourquoi une requête or effectue un balayage de table ou pourquoi un index qui vous semblait utile est ignoré par la requête.

Ce chapitre peut également vous aider à déterminer la limite des performances d'une requête spécifique. Lorsque vous connaissez le nombre de pages d'index et de données que doit lire la requête et s'il est impossible de réduire davantage le nombre des E/S avec une couverture par index, c'est que vous avez atteint le niveau de performances optimal pour l'analyse de la requête et la sélection d'un index. Vous pouvez également examiner d'autres aspects, tels que la configuration du cache, les options de requêtes parallèles ou le positionnement d'un objet.

Coût du balayage de table

Lorsqu'une requête doit balayer une table, Adaptive Server lit chaque page sur le disque pour la transférer dans le cache, puis il vérifie les valeurs des données (s'il y a une clause `where`) et renvoie les lignes qualifiées.

Des balayages de tables sont effectués :

- Lorsqu'il n'existe aucun index sur les colonnes utilisées dans les clauses de recherche.
- Lorsque l'optimiseur détermine que l'utilisation de l'index est plus coûteuse que le balayage de la table. En effet, il peut estimer plus avantageux de lire directement les pages de données au lieu de lire des pages d'index puis les pages de données correspondantes pour chaque ligne à renvoyer.

Le coût d'un balayage de table dépend de la taille de celle-ci et de la taille des E/S.

Remarque Les références aux E/S étendues sont sur un serveur de pages logiques dont la taille est de 2 Ko. Si la taille des pages de votre serveur est de 8 Ko, l'unité de base de l'E/S sera de 8 Ko. Si elle est de 16 Ko, l'unité de base de l'E/S sera de 16 Ko.

Coût du balayage d'une table verrouillée au niveau de toutes les pages

Lors du balayage d'une table verrouillée au niveau de toutes les pages (table APL) avec des E/S de 2 Ko, le coût à prendre en compte est celui d'une E/S physique et d'une E/S logique pour chaque page :

$$\begin{aligned}\text{Coût du balayage de table} = & \quad \text{nombre de pages * 18} \\ & + \text{nombre de pages * 2}\end{aligned}$$

Si la table utilise un cache d'E/S étendues, Adaptive Server calcule le nombre des E/S physiques à effectuer en divisant le nombre de pages par la taille de l'E/S et en appliquant un ajustement basé sur le taux de clusterisation de la page de données. Comme il est impossible d'effectuer des E/S étendues sur les pages de données du premier extent de l'unité d'allocation, chacune de ces pages sera lue par tranche de 2 Ko.

Le coût d'E/S logique est égal à une E/S logique pour chaque page de la table. La formule est la suivante :

$$\begin{aligned}\text{Coût du balayage de table} = & \quad (\text{pages}/\text{pages par E/S}) * \\ & \text{ajustement de clusterisation * 18} + \text{nombre de pages * 2}\end{aligned}$$

Pour plus d'informations sur les taux de clusterisation, reportez-vous à la section « [Incidence des taux de clusterisation sur les estimations d'E/S étendues](#) », page 75.

Remarque Dans une table APL, Adaptive Server n'analyse pas le nombre de pages figurant dans le premier extent d'une unité d'allocation ; l'optimiseur n'inclut donc pas cette E/S supplémentaire dans son estimation.

Coût du balayage d'une table verrouillée au niveau des pages de données seulement

Les tables DOL ne contiennent pas de chaînages des pages, contrairement aux tables APL. Pour effectuer une analyse de table DOL, Adaptive Server :

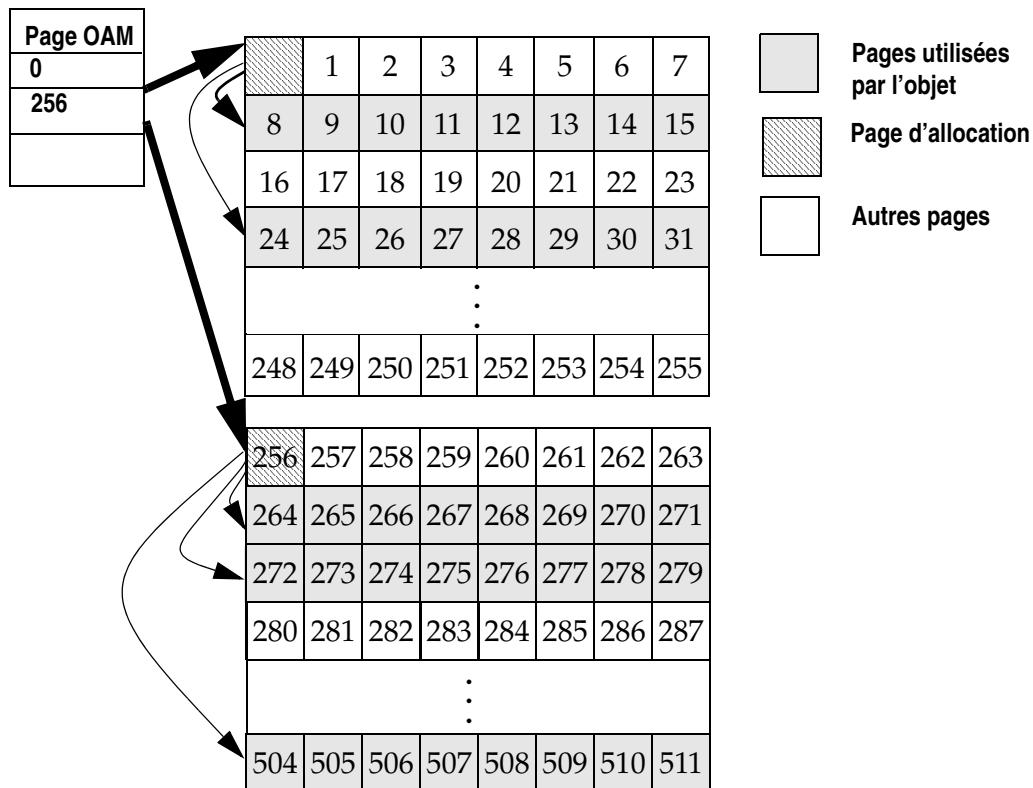
- lit la ou les pages OAM (table d'allocation d'objets) de la table,
- utilise les pointeurs sur les OAM pour accéder aux pages d'allocation,

- utilise les pointeurs sur les pages d'allocation pour localiser les extents utilisés par la table,
- effectue une E/S étendue ou une E/S de 2 Ko sur les pages de l'extent.

Le coût total du balayage d'une table DOL englobe le coût des E/S logiques et physiques pour toutes les pages de la table, plus le coût des E/S logiques et physiques des pages OAM et d'allocation.

La [figure 5-1](#) représente les pointeurs des pages OAM vers les pages d'allocation et des pages d'allocation vers les extents.

Figure 5-1 : Séquence des pointeurs pour les balayages OAM



La formule utilisée pour calculer le coût d'un balayage OAM avec des E/S de 2 Ko est celle-ci :

$$\begin{aligned}\text{Coût du balayage OAM} = & \quad (\text{pages_alloc_OAM} + \text{nbre_pages} + \\& \quad \text{num_pages}) * 18 \\& + (\text{pages_alloc_OAM} + \text{nbre_pages}) * 2\end{aligned}$$

Lorsque des E/S étendues sont possibles, l'optimiseur ajoute le coût des E/S de 2 Ko nécessaires pour lire les pages du premier extent de chaque unité d'allocation au coût des E/S de 16 Ko effectuées sur les pages des autres extents. Le nombre d'E/S physiques correspond au nombre de pages de la table, modifié selon un ajustement basé sur le taux de clusterisation des pages de données pour la table.

Pour plus d'informations sur les taux de clusterisation, reportez-vous à la section « [Incidence des taux de clusterisation sur les estimations d'E/S étendues](#) », page 75.

Les coûts d'E/S logiques correspondent au coût d'une E/S par page plus le coût des E/S logiques permettant de lire les pages OAM et d'allocation. La formule utilisée pour calculer le coût d'un balayage OAM avec des E/S étendues est celle-ci :

$$\begin{aligned}\text{Coût du balayage OAM} = & \quad \text{pages_alloc_OAM} * 18 \\& + \text{pages dans 1er extent} * 18 \\& + \text{pages dans autres extents / pages} \\& \quad \text{par E/S * ajustement de clusterisation} * 18 \\& + \text{pages_alloc_OAM} * 2 \\& + \text{pages dans table} * 2\end{aligned}$$

optdiag indique le nombre de pages pour chacune des valeurs nécessaires.

Lorsqu'une table DOL contient des lignes redirigées, le coût des E/S permettant la lecture de ces lignes s'ajoute au coût des E/S logiques et physiques pour le balayage de la table.

Reportez-vous à la section « [Tables APL sans index](#) », page 181 du guide *Performances et optimisation : concepts de base* pour plus d'informations sur la redirection de ligne.

De la ligne à la page

Lorsque l'optimiseur évalue le coût d'utilisation d'un index pour répondre à une requête, il estime d'abord le nombre des lignes qualifiées puis le nombre des pages à lire.

Les exemples proposés au [Chapitre 2, « Présentation de l'optimiseur »](#), montrent comment Adaptive Server estime le nombre de lignes pour un argument de recherche ou une jointure en utilisant des statistiques. Après quoi il estime le nombre de pages de données et de pages feuille d'index que doit lire la requête :

- Pour une table, l'optimiseur divise le nombre de lignes par le nombre de pages afin de déterminer le nombre moyen de lignes que contient une page de données.
- Pour évaluer le nombre moyen de lignes par page au niveau feuille d'un index, il divise le nombre de lignes de la table par le nombre de pages feuille de l'index.

Une fois qu'il a évalué le nombre de pages, il utilise les taux de clusterisation des pages de données et des pages d'index afin d'ajuster le nombre de pages estimé pour les requêtes qui effectuent des E/S étendues, et il utilise les taux de clusterisation des lignes de données afin de calculer le nombre de pages de données pour les requêtes qui se servent d'index non couvrants.

Incidence des taux de clusterisation sur les estimations d'E/S étendues

Lorsque la clusterisation est importante, les E/S étendues sont efficaces. A mesure que les taux de clusterisation diminuent, l'efficacité des E/S étendues décline rapidement. Pour affiner les estimations des E/S, l'optimiseur utilise un ensemble de taux de clusterisation :

- Pour une table, le taux de clusterisation des pages de données mesure le remplissage et la séquence des pages sur les extents.
- Pour un index, le taux de clusterisation des pages de données mesure l'efficacité des E/S étendues effectuées pour accéder à la table utilisant cet index.

- Le taux de clusterisation des pages d'index mesure le remplissage et la séquence des pages d'index de niveau feuille sur les extents d'index.

Remarque Le taux de clusterisation des lignes de données, autre valeur utilisée dans l'optimisation des requêtes, permet de calculer le nombre de pages de données consultées lors d'un balayage avec un index spécifique. Ce taux n'intervient pas dans l'évaluation du coût des E/S étendues.

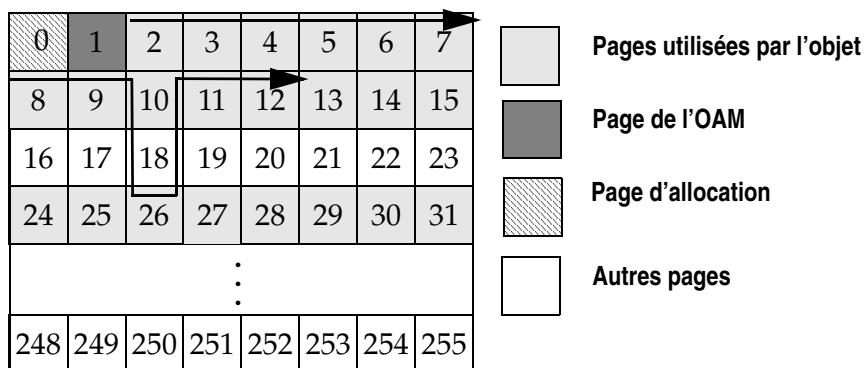
optdiag affiche les taux de clusterisation pour les tables et les index.

Taux de clusterisation des pages de données

Le taux de clusterisation des pages de données d'une table mesure l'efficacité des E/S étendues lors des balayages de tables. Son utilisation varie légèrement selon le plan de verrouillage.

Tables APL

Pour les tables APL, le processus de balayage, qu'il s'applique à la table ou à plusieurs pages à l'aide d'un index clusterisé, suit les pointeurs de page suivante figurant sur chaque page de données. Immédiatement après création de l'index clusterisé, le taux de clusterisation est de 1 et les pages sont classées par numéro au sein des extents. Cependant, à la suite de mises à jour et de fractionnement des pages, le chaînage de celles-ci peut être fragmenté, comme le montre la [figure 5-2](#), où la page 10 a été fractionnée ; les pointeurs de pages renvoient de la page 10 à la page 26 sur un autre extent, puis à la page 11.

Figure 5-2 : Chaînage des pages sur plusieurs extents dans une table APL

Le taux de clusterisation des pages de données d'une table APL mesure l'efficacité des E/S étendues lors des balayages de tables et d'index clusterisés.

Tables DOL

Pour les tables DOL, le taux de clusterisation des pages de données mesure le remplissage des pages dans les extents. Un taux de clusterisation de 1 signifie que toutes les pages des extents sont remplies et en ordre. Si des extents contiennent des pages inutilisées, ce taux est inférieur à 1.

optdiag fournit deux taux de clusterisation des pages de données pour les tables DOL ayant des index clusterisés. La valeur concernant la table est utilisée lors des balayages de tables. La valeur concernant l'index clusterisé est utilisée lors des balayages avec un index.

Taux de clusterisation des pages d'index

Le taux de clusterisation des pages d'index mesure le remplissage et la séquence des pages d'index de niveau feuille dans des extents, pour des index non clusterisés et des index clusterisés de tables DOL. Dans des requêtes qui doivent lire plusieurs pages feuille, le niveau feuille de l'index est analysé à l'aide de pointeurs de page précédente et suivante. Si la requête doit lire plusieurs pages feuille, des E/S de 16 Ko sont effectuées sur ces pages pour lire un seul extent à la fois. Le taux de clusterisation des pages d'index mesure la fragmentation du chaînage des pages de niveau feuille d'un index.

Evaluation du coût d'accès à un index

Lorsqu'une requête contient des arguments de recherche sur des index exploitables, elle n'accède qu'aux pages d'index et de données qui vérifient les arguments de recherche. Adaptive Server compare le coût total des E/S sur les pages d'index et de données avec le coût d'un balayage de table, puis il choisit la méthode la plus économique.

Requête ne renvoyant qu'une seule ligne

Une requête qui ne renvoie qu'une ligne en utilisant un index effectue une opération d'E/S par niveau d'index, plus une lecture pour la page de données. Pour l'optimiseur, le coût total de cette requête englobe le coût d'une E/S physique et celui d'une E/S logique pour chaque page d'index et de données. Le coût d'une requête ponctuelle est le suivant :

$$\begin{aligned} \text{Coût d'une requête ponctuelle} = & \quad (\text{nombre de niveaux d'index} + \\ & \quad \text{page de données}) * 18 \\ & + (\text{nombre de niveaux d'index} + \\ & \quad \text{page de données}) * 2 \end{aligned}$$

optdiag affiche le nombre de niveaux d'index.

La page racine et les pages intermédiaires des index fréquemment utilisés se trouvent souvent en mémoire cache. Dans ce cas, l'E/S physique réelle évite une ou deux lectures.

Requête renvoyant plusieurs lignes

Une requête qui renvoie plusieurs lignes peut être optimisée de différentes manières, selon le type de l'index et le nombre de lignes renvoyées. Voici quelques exemples :

- Requêtes avec des arguments de recherche qui correspondent à plusieurs valeurs, telles que :

```
select title, price
from titles
where pub_id = "P099"
```

- Requêtes à intervalles, telles que :

```
select title, price
from titles
where price between $20 and $50
```

Pour les requêtes qui renvoient un grand nombre de lignes en utilisant la clé principale de l'index, les index clusterisés et les index couvrants non clusterisés sont très performants :

- Si une table APL contient un index clusterisé sur les arguments de recherche, l'index sert à positionner le balayage à la première ligne qualifiée. Les autres lignes répondant à la requête sont lues par balayage avant des pages de données.
- Si un index non clusterisé ou un index clusterisé d'une table DOL couvre la requête, l'index sert à positionner le balayage à la première ligne qualifiée de la page d'index de niveau feuille ; les autres lignes qualifiées sont lues par balayage avant des pages feuille de l'index.

Si l'index ne couvre pas la requête, le recours à un index clusterisé (table DOL) ou à un index non clusterisé exige l'accès aux pages de données qui contiennent chaque ligne d'index vérifiant les arguments de recherche. Or, les lignes qualifiées peuvent être disséminées dans plusieurs pages de données, ou au contraire être regroupées dans un tout petit nombre de pages, notamment si l'index utilisé est un index clusterisé de table DOL. L'optimiseur utilise des taux de clusterisation des lignes de données en vue d'évaluer le nombre d'E/S physiques et logiques nécessaires pour lire toutes les pages de données qualifiées.

Requêtes à intervalles utilisant des index clusterisés (verrouillage APL)

Afin d'évaluer le nombre d'E/S physiques que doit effectuer une requête à intervalles utilisant un index clusterisé dans une table APL, l'optimiseur ajoute les E/S physiques et logiques pour chaque niveau d'index et les E/S physiques et logiques nécessaires pour lire les pages de données adéquates. Les pages de données étant lues dans l'ordre du chaînage, un ajustement de clusterisation est appliqué pour mieux estimer l'efficacité des E/S étendues. La formule est la suivante:

$$\text{Pages de données} = \frac{\text{nombre de lignes qualifiées}}{\text{lignes de données par page}}$$

$$\begin{aligned}\text{Coût de requête à intervalles} = & \text{ nombre de niveaux d'index * 18} \\ & + \text{ pages de données / pages par E/S *} \\ & \text{ajustement de clusterisation * 18} \\ & + \text{ nombre de niveaux d'index * 2} \\ & + \text{ pages de données * 2}\end{aligned}$$

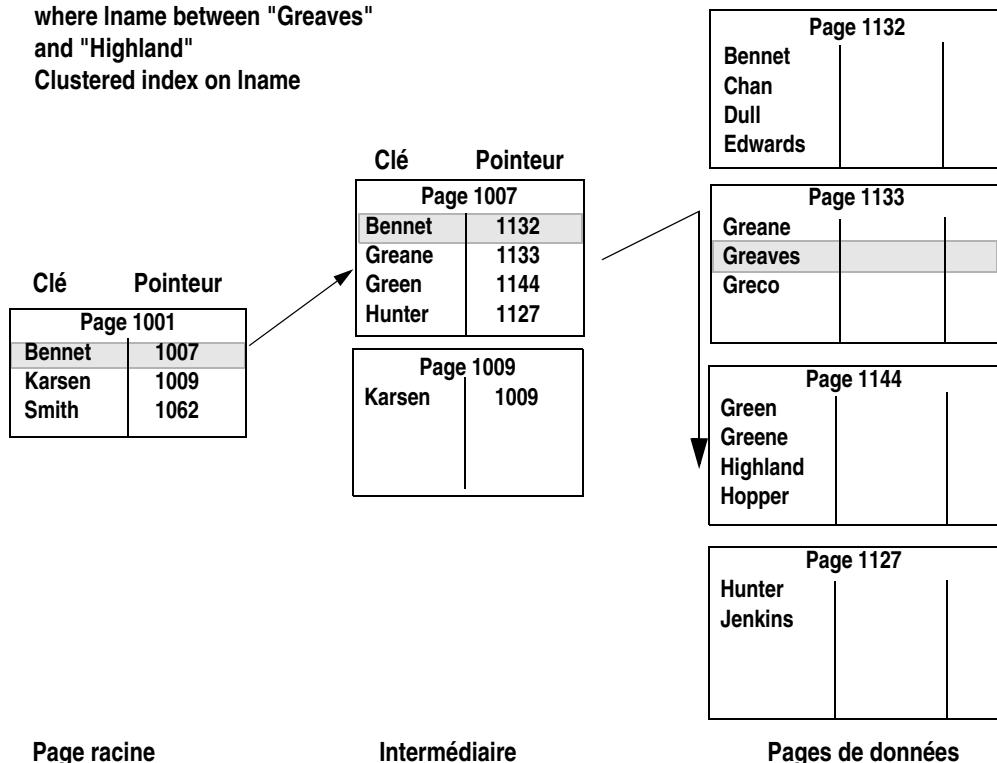
Si une requête renvoie 500 lignes et que la table compte 10 lignes par page, la requête doit lire 50 pages de données, plus une page d'index pour chaque niveau d'index. Si la requête utilise des E/S de 2 Ko, elle a besoin de 50 E/S pour les pages de données. Si la requête utilise des E/S de 16 Ko, ces 50 pages de données ont besoin de 7 E/S.

L'ajustement de clusterisation s'appuie sur le taux de clusterisation des pages de données pour affiner l'estimation du nombre d'E/S étendues nécessaires, en fonction du degré de fragmentation des pages dans les extents de la table.

La [figure 5-3](#) montre comment une requête à intervalles utilisant un index clusterisé positionne la recherche sur la première ligne qualifiée sur les pages de données. Les pointeurs de page suivante permettent d'effectuer un balayage avant sur les pages de données jusqu'à ce que des lignes non qualifiées soient repérées.

Figure 5-3 : Requête à intervalles sur une table APL

```
select fname, lname, id
from employees
where lname between "Greaves"
and "Highland"
Clustered index on lname
```



Requêtes à intervalles avec des index couvrants

Les performances des requêtes à intervalles utilisant des index couvrants sont excellentes pour les raisons suivantes :

- L'index sert à positionner la recherche sur la première ligne qualifiée au niveau feuille de l'index.
- Le nombre de lignes contenues dans une page d'index est supérieur au nombre de lignes contenues dans une page de données correspondante, ce qui réduit le nombre de pages à lire.
- Les pages d'index restent dans le cache plus longtemps que les pages de données, ce qui réduit le nombre d'E/S physiques nécessaires.
- Si le cache utilisé par l'index permet des E/S étendues, une E/S peut lire jusqu'à 8 pages de niveau feuille.
- L'accès aux pages de données est inutile.

Les index non clusterisés et les index clusterisés des tables de données DOL ont un niveau feuille au-dessus du niveau de données, de sorte qu'ils permettent une couverture.

Le coût d'utilisation d'un index couvrant la requête est déterminé par :

- le nombre de niveaux non feuille d'index,
- le nombre de lignes renvoyées par la requête,
- le nombre de lignes par page au niveau feuille de l'index,
- le nombre de pages feuille lues par E/S,
- le taux de clusterisation de page d'index, utilisé pour ajuster les estimations des E/S étendues lorsque les pages d'index ne sont pas stockées séquentiellement sur les extents.

La formule ci-dessous indique les coûts :

$$\begin{aligned} \text{Pages de niveau feuille} &= \text{nombre de lignes qualifiées / lignes de niveau feuille par page} \\ \text{Coût de balayage de couverture} &= \text{nombre de niveaux d'index} * 18 \\ &\quad + (\text{pages de niveau feuille} / \text{pages par E/S}) * \text{ajustement de clusterisation} * 18 \\ &\quad + \text{nombre de niveaux d'index} * 2 \\ &\quad + \text{pages de niveau feuille} * 2 \end{aligned}$$

Par exemple, si une requête doit lire 1 200 pages feuille à raison de 40 lignes par page, elle lira 30 pages de niveau feuille. Si des E/S étendues sont possibles, cette lecture nécessite 4 E/S. Si des insertions ont provoqué des ruptures de pages au niveau feuille de l'index, l'ajustement de clusterisation augmente le nombre estimé d'E/S étendues.

Requêtes à intervalles avec des index non couvrants

Lorsqu'un index non clusterisé ou un index clusterisé d'une table DOL ne couvre pas la requête, Adaptive Server :

- utilise l'index pour localiser la première ligne qualifiée au niveau feuille de l'index non clusterisé ;
- suit le pointeur sur la page de données pour cet index et la lit ;
- trouve la ligne suivante sur la page d'index et localise sa page de données, puis continue le processus jusqu'à ce que toutes les clés correspondantes aient été utilisées.

Pour chaque clé consécutive, la ligne de données peut se trouver sur la même page que la ligne de la clé précédente ou sur une autre page. La clusterisation des valeurs de clé pour chaque index est mesurée à l'aide du *taux de clusterisation des lignes de données*. Ce taux est appliqué pour estimer le nombre d'E/S physiques et logiques.

Lorsque le taux de clusterisation des lignes de données est à 1,0, la clusterisation est élevée. Immédiatement après la création d'un index clusterisé, ces taux ont toujours une valeur haute, égale à 1,00000 ou 0,999997, par exemple. Les lignes des pages de données sont stockées dans le même ordre que les lignes de l'index. Le nombre d'E/S physiques et logiques nécessaires pour lire les pages de données est (globalement) égal au nombre de lignes à renvoyer, divisé par le nombre de lignes par page. Pour une table de 10 lignes par page, une requête qui doit renvoyer 500 lignes va lire 50 pages lorsque le taux de clusterisation des lignes de données est 1.

Lorsque ce taux est très bas, les lignes de données sont disséminées dans les pages, sans relation avec l'ordre des clés. Les index non clusterisés ont souvent des taux de clusterisation des lignes de données bas, puisqu'il n'existe pas de relation entre l'ordre des clés d'index et l'ordre des lignes de données sur les pages. Lorsque ce taux est égal à 0 ou proche de 0, le nombre d'E/S physiques et logiques requises peut aller jusqu'à 1 E/S sur la page de données pour chaque ligne à renvoyer. Une requête devant renvoyer 500 lignes devra donc lire 500 pages ou presque si le taux de clusterisation des lignes de données est proche de 0 et si les lignes sont largement disséminées sur les pages de données. Dans une table volumineuse, les performances restent encore acceptables, mais sur une table de moins de 500 pages, l'optimiseur choisit la solution la moins coûteuse, à savoir un balayage de table.

La taille du cache de données intervient également dans le calcul des E/S physiques. Si le taux de clusterisation des lignes de données est très bas et le cache petit, des pages peuvent être supprimées du cache avant d'avoir été réutilisées. Si le cache est de grande taille, l'optimiseur estime qu'il peut contenir certaines pages et il tente alors de les y trouver.

Taille du jeu de résultats et utilisation d'un index

Une requête à intervalles qui renvoie peu de lignes fonctionne très bien avec un index ; cependant, cette requête, si elle doit renvoyer de nombreuses lignes, n'utilise pas nécessairement un index, car il peut être plus coûteux d'effectuer les E/S physiques et logiques sur un grand nombre de pages d'index, ajouté à un grand nombre de pages de données. Plus le taux de clusterisation des lignes de données est bas, plus le coût d'utilisation de l'index est élevé.

Au niveau feuille d'un index non clusterisé ou d'un index clusterisé d'une table DOL, les clés sont stockées séquentiellement. Pour un argument de recherche sur une valeur qui fournit 100 lignes qualifiées, les lignes du niveau feuille de l'index tiennent peut-être sur un ou deux pages d'index. Les lignes de données réelles risquent cependant de se trouver toutes sur des pages de données différentes. Les requêtes suivantes montrent l'incidence des taux de clusterisation des lignes de données sur les estimations des E/S. La table authors utilise le verrouillage des lignes de données (datarows) et comporte les index suivants :

- un index clusterisé sur au_lname ;
- un index non clusterisé sur state.

Chacune de ces requêtes renvoie près de 100 lignes :

```
select au_lname, phone
from authors
where au_lname like "E%"
select au_id, au_lname, phone
from authors
where state = "NC"
```

Le tableau suivant indique le taux de clusterisation des lignes de données pour chaque index et l'estimation de l'optimiseur concernant le nombre de lignes à renvoyer ainsi que le nombre de pages requises.

Argument de recherche sur	Taux de clusterisation de ligne de données	Estimation du nombre de lignes	Estimation du nombre de pages	Taille des E/S de données
au_lname	.999789	101	8	16 Ko
state	.232539	103	83	2 Ko

Les principales informations sur la table sont :

- La table contient 262 pages.
- Chaque page contient 19 lignes de données.

Bien que chaque requête présente ses clauses de recherche dans un format correct et que chaque clause corresponde à un index, seule la première utilise l'index : pour l'autre requête, un balayage de table revient moins cher que l'utilisation de l'index. Avec 262 pages, le coût d'un balayage de la table est :

$$\begin{aligned} \text{Coût de balayage de table} &= (262 / 8) = 37 * 18 = 666 \\ &+ 262 * 2 = 524 \end{aligned}$$

1190

Analyse approfondie de l'évaluation du coût d'un argument de recherche

Lorsque l'on étudie attentivement les tables, les taux de clusterisation et les arguments de recherche, le choix d'un balayage de table s'explique par les raisons suivantes :

- L'estimation relative à l'index clusterisé sur au_lname inclut juste 8 E/S physiques :
 - 6 E/S (de 16 Ko) sur les pages de données, car le taux de clusterisation des lignes de données indique une clusterisation élevée ;
 - 2 E/S pour les pages d'index (il y a 128 lignes par page feuille) ; une E/S de 16 Ko est également effectuée pour les pages d'index de niveau feuille.
- La requête utilisant l'argument de recherche sur state doit lire beaucoup plus de pages de données puisque le taux de clusterisation des lignes de données est bas. L'optimiseur choisit des E/S de 2 Ko sur les pages de données. 83 E/S physiques représentent plus du double des E/S physiques requises pour un balayage de table (avec des E/S de 16 Ko).

Evaluation du coût des balayages d'index non couvrants

La formule de base qui sert à évaluer les E/S pour des requêtes accédant aux données via un index non couvrant est la suivante :

Pages de niveau feuille = nombre de lignes qualifiées / lignes de niveau feuille par page

Pages de données = nombre de lignes qualifiées *
 ajustement de clusterisation de ligne de données

Coût du balayage = nombre de niveaux d'index non niveau feuille * 18
 + (pages de niveau feuille / pages par E/S) *
 ajustement de clusterisation de pages de données * 18
 + (pages de données / pages par E/S) *
 ajustement de clusterisation de pages de données * 18
 + nombre de niveaux d'index non niveau feuille * 18
 + pages de niveau feuille * 2
 + nombre de lignes de qualification *
 ajustement de clusterisation de pages de données * 2

Evaluation des lignes redirigées

Si une table DOL contient des lignes redirigées, le coût des E/S supplémentaires pour y accéder s'ajoute à celui des balayages d'index sans couverture. Pour calculer ce coût, l'optimiseur multiplie le nombre de lignes redirigées qui sont dans la table par le pourcentage de lignes de la table renvoyées par la requête. Le coût supplémentaire est :

Coût de ligne redirigée = % de lignes renvoyées * Nombre de lignes redirigées dans la table

Evaluation du coût des requêtes utilisant la clause order by

Les requêtes qui effectuent un tri pour order by peuvent créer et trier un index, ou utiliser l'index pour renvoyer des lignes en observant l'ordre de l'index. Par exemple, l'optimiseur peut choisir l'une des méthodes d'accès suivantes pour une requête comportant une clause order by :

- En l'absence d'argument de recherche exploitable : il utilise un balayage de table suivi d'un tri de la table de travail.
- Avec un argument de recherche sélectif ou une jointure sur un index qui ne correspond pas à la clause order by : il utilise un balayage d'index suivi d'un tri de la table de travail.
- Avec un argument de recherche ou une jointure sur un index qui correspond à la clause order by : il procède à un balayage de cet index, sans table de travail ni tri.

Des tris sont toujours nécessaires lorsque les colonnes figurant dans le jeu de résultats constituent un surensemble des clés d'index. Par exemple, si l'index sur authors comprend au_fname et au_lname et si la clause order by comprend également au_id, un tri est nécessaire.

S'il existe des arguments de recherche sur des index correspondant à la clause order by et d'autres arguments sur des index qui ne supportent pas le tri, l'optimiseur évalue les deux méthodes d'accès. Si la table de travail et le tri sont nécessaires, le coût des E/S requises par ces opérations s'ajoute au coût du balayage d'index. Si un index permet d'éviter le tri, dbcc traceon(302) imprime un message pendant l'évaluation du coût de l'argument ou de la jointure.

Reportez-vous à la section « [Messages d'avertissement pour éviter les tris](#) », [page 189](#) du guide *Performances et optimisation : contrôle et analyse* pour de plus amples informations.

Parallèlement à la disponibilité des index, deux facteurs essentiels déterminent le recours à l'index :

- La clause `order by` doit spécifier un sous-ensemble ordonné des clés d'index.
- La clause `order by` et l'index doivent avoir un ordre d'organisation des clés croissant/décroissant compatible.

Sous-ensemble ordonné et tris

Pour qu'une requête utilise un index afin d'éviter un tri, les clés spécifiées dans la clause `order by` doivent être un sous-ensemble ordonné des clés d'index. Par exemple, si l'index spécifie les clés A, B, C, D :

- Les clauses `order by` suivantes peuvent utiliser l'index :
 - A
 - A, B
 - A, B, C
 - A, B, C, D
- Un autre ensemble de colonnes ne peut pas utiliser l'index.
Par exemple, les clés suivantes ne sont pas des sous-ensembles ordonnés :
 - A, C
 - B, C, D

Ordre des clés et tris

Les clauses order by et les commandes qui créent des index peuvent utiliser les critères d'ordre asc ou desc (croissant ou décroissant) :

- Pour la création d'index, les critères asc et desc précisent l'ordre de stockage des clés dans l'index.
- Dans la clause order by, ces critères indiquent l'ordre de renvoi des colonnes dans les résultats.

Pour éviter un tri avec un index spécifique, il convient que les critères asc ou desc dans la clause order by soient exactement les mêmes que ceux utilisés lors de la création de l'index, ou qu'ils soient exactement contraires.

Spécification de l'ordre croissant ou décroissant des clés d'index

Les requêtes qui mêlent ordres croissants et décroissants dans une clause order by n'effectuent pas un tri séparé si l'index a été créé avec la même combinaison d'ordres ou avec un ordre qui est l'exact opposé de celui indiqué dans la clause order by. Les index font l'objet d'une lecture avant ou arrière, suivant les pointeurs de chaînage des pages de niveau feuille.

Par exemple, cette commande crée un index sur la table titles avec pub_id en ordre croissant et pubdate en ordre décroissant :

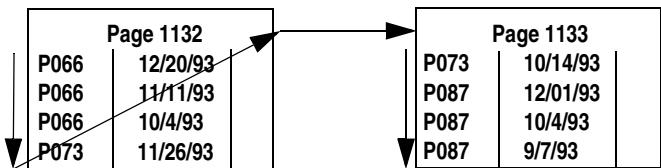
```
create index pub_ix  
on titles (pub_id asc, pubdate desc)
```

Les lignes sont ordonnées sur les pages comme indiqué dans la [figure 5-4](#). Si l'ordre croissant et décroissant de la requête correspond à l'ordre de création de l'index, le résultat est une lecture avant qui, à partir du début de l'index ou de la première ligne qualifiée, renvoie les lignes de chaque page dans l'ordre et se sert des pointeurs appropriés pour lire les pages.

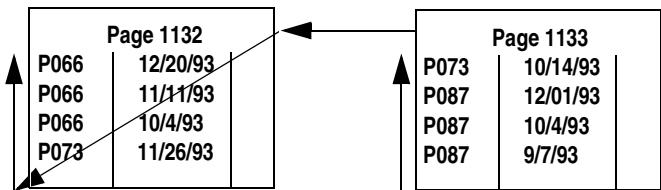
Si l'ordre de la requête est l'exact opposé de celui de création de l'index, le résultat est une lecture arrière qui, à partir de la dernière page de l'index ou de la page contenant la dernière ligne qualifiée, renvoie les lignes de chaque page dans l'ordre décroissant et se sert des pointeurs appropriés pour lire les pages précédentes.

Figure 5-4 : Balayages avant et arrière d'un index

Balayage avant : lit les lignes de la page dans l'ordre, puis utilise les pointeurs page suivante



Balayage arrière : lit les lignes de la page dans l'ordre inverse, puis utilise les pointeurs page précédente



La requête suivante, qui utilise l'index présenté à la figure 5-4, effectue un balayage avant :

```
select *
from titles
order by pub_id asc, pubdate desc
```

La requête suivante, qui utilise l'index présenté à la figure 5-4, effectue un balayage arrière :

```
select *
from titles
order by pub_id desc, pubdate asc
```

Pour les deux requêtes ci-après, effectuées sur la même table, un tri est nécessaire car la clause `order by` ne correspond pas à l'ordre spécifié pour l'index :

```
select *
from titles
order by pub_id desc, pubdate desc
select *
from titles
order by pub_id asc, pubdate asc
```

Remarque Les opérations de tri parallèle sont optimisées de manière différente lorsqu'il s'agit de tables partitionnées. Pour de plus amples informations, reportez-vous à la section Chapitre 9, « Tri parallèle ».

Evaluation des opérations de tri

Lorsqu'il optimise les requêtes pour un tri, Adaptive Server effectue les opérations suivantes :

- Il calcule le coût des ressources affectées à l'utilisation d'un index correspondant à l'ordre de tri requis, lorsque cet index existe.
- Il calcule le coût des E/S physiques et logiques nécessaires pour créer une table de travail et trier chaque index, lorsque l'ordre des index ne correspond pas à l'ordre de tri. Il calcule le coût des E/S physiques et logiques pour balayer la table, créer une table de travail et effectuer le tri.

Le cumul du coût de création et de tri de la table de travail avec le coût requis pour accéder à l'index justifie l'utilisation d'un index supportant la clause `order by`. Cependant, si l'on compare des index très sélectifs mais non ordonnés à des index ordonnés mais non sélectifs, il apparaît que :

- les coûts d'accès sont bas pour l'index le plus sélectif, de même que les coûts associés au tri,
- les coûts d'accès sont élevés pour l'index moins sélectif et ils peuvent excéder les coûts d'accès associés à l'index le plus sélectif et au tri.

Tables APL avec index clusterisés

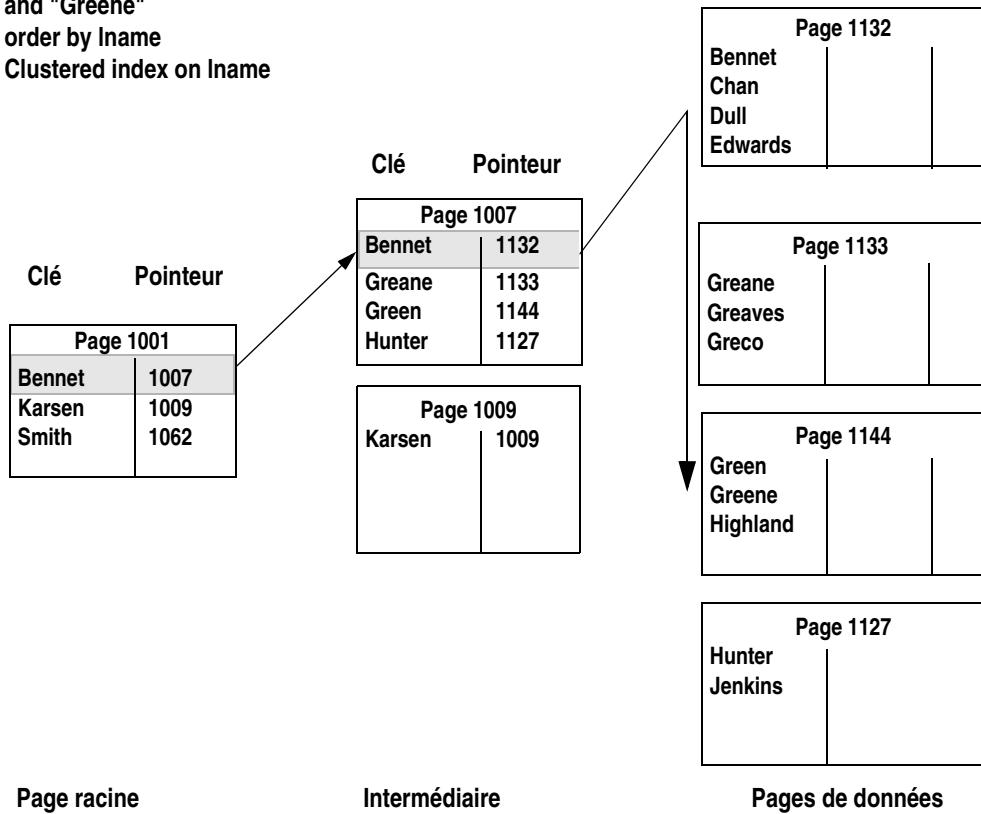
Sur les tables APL avec des index clusterisés, les requêtes avec une clause `order by` correspondant aux clés d'index sont performantes dans les cas suivants :

- Il y a également un argument de recherche qui utilise l'index, la clé d'index positionne la recherche sur la page de données contenant la première ligne qualifiée.
- Le balayage suit les pointeurs de page suivante jusqu'à ce que toutes les lignes qualifiées aient été trouvées.
- Aucun tri n'est nécessaire.

Dans la [figure 5-5](#), l'index a été créé dans l'ordre croissant et la clause `order by` ne spécifie pas l'ordre ; par conséquent, l'ordre croissant est utilisé par défaut.

Figure 5-5 : Opération de tri utilisant un index clusterisé sur une table APL

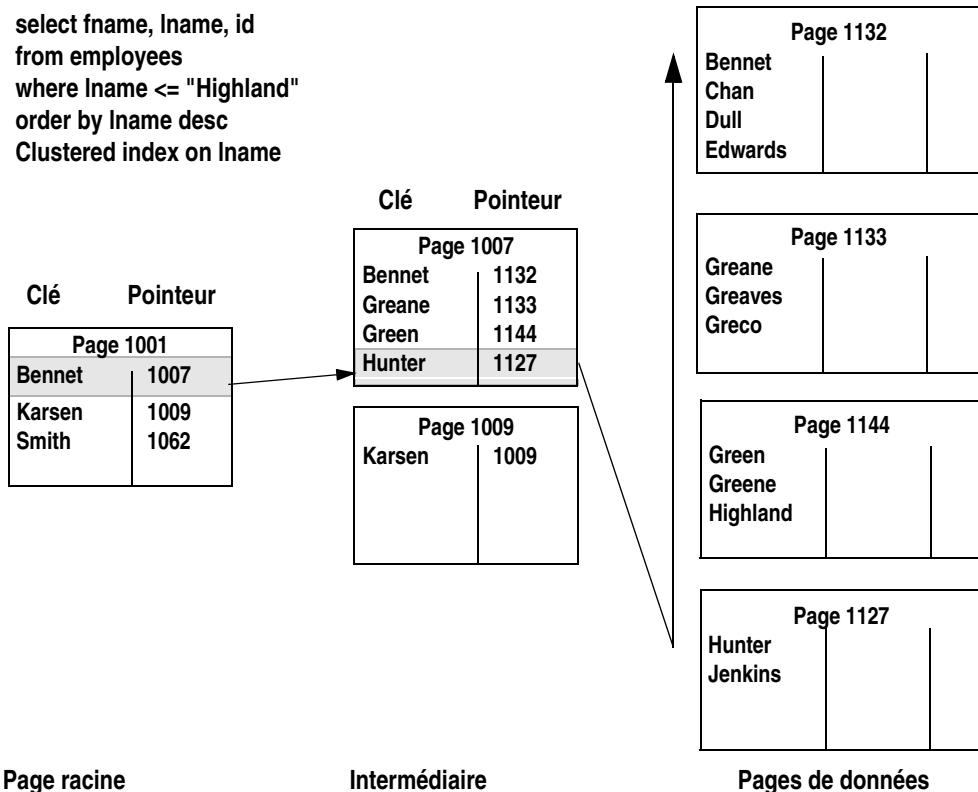
```
select fname, lname, id
from employees
where lname between "Dull"
and "Greene"
order by lname
Clustered index on lname
```



Les requêtes qui requièrent un ordre de tri décroissant (par exemple, `order by title_id desc`) peuvent éviter le tri en balayant les pages dans l'ordre inverse. Si la totalité de la table est nécessaire pour une requête sans clause `where`, Adaptive Server suit les pointeurs d'index jusqu'à la dernière page, puis effectue une lecture arrière à l'aide des pointeurs de page précédente. Si la clause `where` comporte une clé d'index, l'index sert à positionner la recherche et les pages sont balayées vers l'arrière, comme le montre la [figure 5-6](#).

Figure 5-6 : Requête order by desc utilisant un index clusterisé

```
select fname, lname, id
from employees
where lname <= "Highland"
order by lname desc
Clustered index on lname
```



Opérations de tri avec un index couvrant la requête

Lorsqu'un index couvre la requête et que les colonnes order by constituent un sous-ensemble ordonné de clés d'index, les lignes sont renvoyées directement depuis les pages feuille de l'index non clusterisé. Si les colonnes ne forment pas un sous-ensemble ordonné de clés d'index, une table de travail est créée et triée.

Avec un index non clusterisé sur au_lname, au_fname, au_id de la table authors, cette requête peut renvoyer les données directement à partir des pages de niveau feuille :

```
select au_id, au_lname
from authors
order by au_lname, au_fname
```

Tris et index non couvrants

Avec un index non couvrant, Adaptive Server détermine si l'utilisation de l'index qui supporte les conditions de tri est moins coûteuse qu'un balayage de table ou que l'utilisation d'un index plus sélectif avec insertion et tri des lignes dans une table de travail. Le coût d'utilisation de l'index dépend du nombre de lignes et du taux de clusterisation des lignes de données.

Balayages arrière et jointures

Si deux tables ou plus sont jointes et que la clause `order by` définit un ordre décroissant pour les clés d'index sur les tables jointes, ces tables et ces index peuvent faire l'objet d'un balayage arrière, ce qui évite la création et le tri d'une table de travail. Si toutes les colonnes d'une table sont dans l'ordre croissant alors que celles des autres tables sont dans l'ordre décroissant, le balayage de la première s'effectue dans l'ordre croissant et celui des autres dans l'ordre décroissant.

Interblocages et balayages décroissants

Les balayages arrière peuvent entraîner des interblocages avec des requêtes effectuant des modifications via des balayages avant, ainsi qu'avec des requêtes effectuant des ruptures et des réductions de page, sauf lorsque les balayages arrière sont exécutés au niveau d'isolement de transaction 0.

Le paramètre `allow backward scans` permet de configurer l'optimiseur pour qu'il utilise la stratégie de balayage arrière. La valeur par défaut 1 active les balayages arrière.

Pour plus d'informations sur ce paramètre, reportez-vous au *Guide d'administration système*.

Reportez-vous également à la section « [Balayage d'index](#) », page 280 pour obtenir des informations sur le nombre de balayage croissants et décroissants effectués et à la section « [Interblocages par type de verrou](#) », page 291 du guide *Performances et optimisation : contrôle et analyse* pour plus d'informations sur la détection des interblocages.

Méthodes d'accès et évaluation du coût des clauses or et in

Lorsqu'une requête sur une table unique contient des clauses or ou une clause in (values_list), elle peut être optimisée de plusieurs façons, en fonction de la présence d'index, de la sélectivité des arguments de recherche, de l'existence d'autres arguments de recherche et de la possibilité ou non que les clauses renvoient des lignes dupliquées.

Syntaxe or

Lorsqu'une requête utilise une clause or, l'optimiseur peut choisir une stratégie or plus coûteuse non seulement pour les balayages de tables, mais aussi pour les balayages d'index car la stratégie offre une meilleure concurrence d'accès au verrouillage.

Les clauses or prennent l'une des formes suivantes :

```
where nom_colonne1 = <valeur>
      or nom_colonne1 = <valeur>
```

...

ou :

```
where nom_colonne1 = <valeur>
      or nom_colonne2 = <value>
```

...

Conversion de in (values_list) en clause or

La préconversion des listes in en clauses or transforme ainsi cette requête :

```
select title_id, price
      from titles
      where title_id in ("PS1372", "PS2091", "PS2106")
```

en :

```
select title_id, price
      from titles
      where title_id = "PS1372"
            or title_id = "PS2091"
            or title_id = "PS2106"
```

Méthodes de traitement des clauses or

Une requête sur une seule table qui inclut des clauses or est une conjonction de plusieurs requêtes. Même si certaines lignes remplissent plusieurs conditions, chaque ligne ne doit être renvoyée qu'une seule fois. Selon les index et les clauses de la requête, les requêtes or peuvent être résolues de l'une des façons suivantes :

- Si des clauses liées par or ne sont pas indexées, la requête doit utiliser un balayage de table. S'il existe un index sur type, mais pas sur advance, la requête effectue un balayage de table :

```
select title_id, price
from titles
where type = "business" or advance > 10000
```

- S'il se peut qu'une ou plusieurs clauses or correspondent aux valeurs de la même ligne, la requête est résolue selon la *stratégie OR*, qui équivaut à l'utilisation d'un *index dynamique*. La stratégie OR sélectionne les ID des lignes correspondantes dans une table de travail et trie celle-ci pour supprimer les ID de lignes dupliqués. Par exemple, il peut exister des lignes pour lesquelles les deux conditions suivantes sont vraies :

```
select title_id
from titles
where pub_id = "P076" or type > "business"
```

S'il existe un index sur pub_id et un autre sur type, la stratégie OR est applicable.

Pour de plus amples informations, reportez-vous à la section « [Index dynamique \(stratégie OR\)](#) », page 97.

Remarque La *stratégie OR* (plusieurs balayages d'index avec correspondance) n'est prise en compte que pour les prédictats d'égalité. Elle est exclue pour les prédictats d'intervalle même si elle remplit d'autres conditions. A titre d'exemple, si une instruction select contient les expressions suivantes :

```
where bar between 1 and 5
      or bar between 10 and 15
```

Elle ne sera pas utilisée pour la *stratégie OR*.

- S'il est impossible que les clauses or sélectionnent la même ligne, la requête peut être résolue par plusieurs balayages d'index avec correspondance, également appelées **stratégie OR spéciale**. La stratégie OR spéciale n'a besoin ni de table de travail ni de tri. Les clauses or de cette requête ne peuvent pas sélectionner deux fois la même ligne :

```
select title_id, price
from titles
where pub_id = "P076" or pub_id = "P087"
```

Avec un index sur pub_id, cette requête est résolue à l'aide de deux balayages d'index avec correspondance.

Pour de plus amples informations, reportez-vous à la section « [Balayages d'index multiples avec correspondance \(stratégie OR spéciale\)](#) », page 99.

- Les coûts d'accès à l'index pour chaque clause or s'ajoutent, ainsi que le coût du tri si nécessaire. Si le coût total dépasse celui d'un balayage de table, cette dernière solution est choisie. Par exemple, la requête suivante utilise un balayage de table si le coût total de tous les balayages d'index sur pub_id est supérieur à celui du balayage de table :

```
select title_id, price
from titles
where pub_id in ("P095", "P099", "P128", "P220",
" P411", "P445", "P580", "P988")
```

- Si la requête contient des arguments de recherche supplémentaires sur des colonnes indexées, la conversion des prédictats peut provoquer l'ajout d'arguments de recherche optimisables, offrant ainsi de nouvelles options d'optimisation. Les coûts d'utilisation de toutes les méthodes d'accès possibles sont comparés et la solution la plus avantageuse est adoptée. La requête suivante contient un argument de recherche sur type ainsi que des clauses liées avec or :

```
select title_id, type, price from titles
where type = "business"
and (pub_id = "P076" or pubdate > "12/1/93")
```

Avec un index distinct sur chaque argument de recherche, l'optimiseur utilise la méthode d'accès la moins onéreuse :

- L'index sur type
- La stratégie OR sur pub_id et pubdate

Utilisation des balayages de tables pour les requêtes or

Une requête contenant des clauses or ou une clause in (values_list) utilise un balayage de table si l'une des conditions suivantes se réalise :

- le coût de tous les accès à l'index est supérieur au coût d'un balayage de table, ou
- au moins une des colonnes n'est pas indexée, de sorte que la seule façon de résoudre la requête consiste à effectuer un balayage de table.

Index dynamique (stratégie OR)

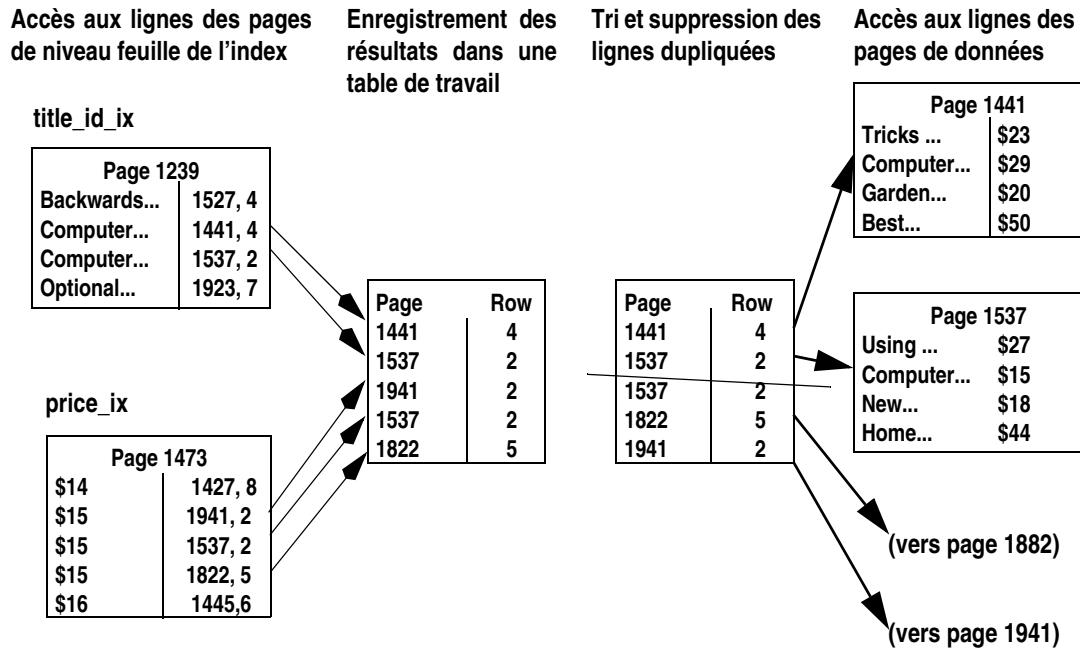
Si la requête utilise la stratégie OR car elle risque de renvoyer des lignes dupliquées, les index appropriés sont utilisés pour extraire les ID des lignes qui remplissent les conditions de chaque clause or. Les ID de ligne pour chaque clause or sont stockées dans une table de travail. La table de travail ne contenant que des ID de ligne, on parle d'« index dynamique ». Adaptive Server trie ensuite son contenu afin de supprimer les ID de lignes dupliqués. Les ID de ligne sont utilisés pour extraire les lignes des tables sous-jacentes. Le coût total de la requête englobe :

- le coût de tous les accès à l'index ; c'est-à-dire, pour chaque clause or, le coût d'utilisation de l'index afin d'accéder aux ID de ligne sur les pages feuille de l'index (ou sur les pages de données s'il s'agit d'un index clusterisé dans une table APL),
- le coût de lecture de la table de travail et du tri,
- le coût d'utilisation des ID de ligne pour accéder aux pages de données.

La [figure 5-7](#) illustre le processus de création et de tri d'un index dynamique pour une requête or sur deux colonnes différentes.

Figure 5-7 : Résolution de requêtes or avec la stratégie OR

```
select title_id, price
from titles
where price <= $15 or title like "Compute%"
```



Comme le montre la [figure 5-7](#), l'optimiseur peut utiliser un index différent pour chaque clause.

Lorsque la stratégie OR est employée, showplan affiche les messages « Using Dynamic Index » et « Positioning by Row Identifier (RID) ».

Reportez-vous à la section « [Message d'index dynamique \(stratégie OR\)](#) », [page 112](#) du manuel *Performances et optimisation : contrôle et analyse* pour de plus amples informations.

Les requêtes sur des curseurs ne peuvent pas utiliser la stratégie OR, mais elles doivent effectuer un balayage de table. Toutefois, elles peuvent utiliser la stratégie de balayages d'index multiples avec correspondance.

Le verrouillage durant les requêtes qui utilisent la stratégie OR dépend du plan de verrouillage de la table.

Balayages d'index multiples avec correspondance (stratégie OR spéciale)

Adaptive Server utilise des balayages d'index multiples avec correspondance lorsque les clauses or sont sur la même table et qu'elles ne peuvent en aucun cas renvoyer des lignes dupliquées. Par exemple, cette requête ne peut pas renvoyer de lignes dupliquées :

```
select title
      from titles
     where title_id in ("T6650", "T95065", "T11365")
```

Elle sera résolue par des balayages d'index multiples avec correspondance, à l'aide de l'index sur `title_id`. Le coût total de la requête est la somme de tous les accès à l'index effectués. Si l'index sur `title_id` a trois niveaux, chaque clause or nécessite 3 lectures de l'index, plus une pour la page de données, de sorte que le coût total pour chaque clause est de 4 E/S logiques et 4 E/S physiques, le coût total de la requête étant estimé à 12 E/S logiques et 12 E/S physiques.

L'optimiseur détermine l'index à utiliser pour chaque clause or ou la valeur de la clause `in (values_list)` en évaluant séparément leur coût respectif. Si chaque colonne spécifiée dans une clause est indexée, il est possible d'utiliser un index différent pour chaque clause ou valeur. Lorsque la stratégie OR spéciale est appliquée, `showplan` affiche le message « Using N Matching Index Scans ».

Reportez-vous à la section « [Message de balayages d'index avec correspondance](#) », page 111 du manuel *Performances et optimisation : contrôle et analyse*.

Optimisation des agrégats

Les agrégats sont traités en deux étapes :

- Premièrement, les lignes appropriées sont extraites à l'aide des index exploitables ou la table est balayée. Pour les agrégats vectoriels (ou groupés), les résultats sont placés dans une table de travail. Pour les agrégats scalaires, les résultats sont calculés dans une variable en mémoire.
- Deuxièmement, la table de travail est balayée pour récupérer les résultats (pour les agrégats vectoriels) ou les résultats sont renvoyés à partir de la variable interne.

Les agrégats vectoriels peuvent utiliser un index composé couvrant sur la colonne agrégée ainsi que sur la colonne de regroupement (le cas échéant), au lieu d'effectuer des balayages de tables. Par exemple, si la table titles dispose d'un index non clusterisé sur type, price, la requête suivante obtient ses résultats en balayant le niveau feuille de l'index non clusterisé :

```
select type, avg(price)
      from titles
        group by type
```

Les agrégats scalaires peuvent aussi utiliser la couverture par index pour réduire les E/S. Par exemple, la requête suivante peut utiliser l'index sur type, price :

```
select min(price)
      from titles
```

Le [tableau 5-1](#) illustre des méthodes d'accès que l'optimiseur peut choisir pour des requêtes contenant des agrégats quand il n'existe pas de clause where, having ou group by dans la requête.

Tableau 5-1 : Méthodes d'accès spéciales pour les agrégats

Agrégat	Description d'index	Méthode d'accès
min	L'agrégat scalaire est la colonne d'en-tête	Utilise en premier la valeur de la page racine de l'index.
max	Index clusterisé sur une table APL	Suit le dernier pointeur de la page racine et des pages intermédiaires vers la page de données, et renvoie la dernière valeur.
	Index clusterisé sur une table DOL Tout index non clusterisé	Suit le dernier pointeur de la page racine et des pages intermédiaires vers la page feuille, et renvoie la dernière valeur.
count(*)	Index non clusterisés ou index clusterisé sur une table DOL	Compte toutes les lignes du niveau feuille de l'index ayant le plus petit nombre de pages.
count(<i>nom_col</i>)	Index non clusterisés couvrants ou index clusterisé couvrant sur une table DOL	Compte toutes les valeurs non NULL de niveau feuille du plus petit index contenant le nom de la colonne.

Utilisation conjointe des agrégats **max** et **min**

Lorsqu'ils sont utilisés séparément sur des colonnes d'en-tête indexées, les agrégats **max** et **min** ont recours à un traitement spécial si la requête ne contient aucune clause **where** :

- Les agrégats **min** extraient la première valeur sur la page racine de l'index en effectuant une seule lecture pour trouver cette valeur.
- Les agrégats **max** suivent la dernière entrée sur la dernière page pour chaque niveau d'index jusqu'à atteindre le niveau feuille.

Lorsque **min** et **max** sont utilisés conjointement, cette optimisation n'est pas possible. Tout le niveau feuille d'un index est balayé pour localiser la première et la dernière valeur.

Les optimisations **min** et **max** ne s'appliquent pas si :

- l'expression à l'intérieur de la fonction **max** ou **min** n'est pas une colonne. Lorsque *col_numérique* a un index non clusterisé :
 - **max(col_numérique*2)** contient une opération sur une colonne, de sorte que la requête effectue une lecture au niveau feuille de l'index.
 - **max(col_numérique)*2** utilise l'optimisation **max** car la multiplication est effectuée sur le résultat de la fonction.
- Il existe un autre agrégat dans la requête.
- Il existe une clause **group by**.

Requêtes utilisant à la fois **min** et **max**

Si vous avez des agrégats **max** et **min** optimisables, vous obtiendrez de meilleures performances en les plaçant dans des requêtes séparées.

Par exemple, même s'il existe un index avec **price** comme clé principale, la requête suivante effectue un balayage de tout le niveau feuille de l'index :

```
select max(price), min(price)
      from titles
```

Lorsque vous les séparez, Adaptive Server utilise l'index une fois pour chacune des deux requêtes au lieu d'effectuer une lecture de toutes les pages feuille. Cet exemple présente deux requêtes :

```
select max(price)
      from titles
select min(price)
      from titles
```

Mise à jour des opérations

Adaptive Server traite les mises à jour de différentes façons, selon les modifications apportées aux données et les index utilisés pour localiser les lignes. Les deux principaux types sont les **mises à jour différées** et les **mise à jour directes**. Adaptive Server effectue des mises à jour directes chaque fois que cela est possible.

Mises à jour directes

Adaptive Server effectue des mises à jour directes en une seule opération :

- Il localise les index et les lignes de données concernés.
- Il écrit les enregistrements de journal relatifs aux modifications dans le journal de transactions.
- Il apporte les modifications aux pages de données et à toutes les pages d'index concernées.

Il existe trois méthodes pour effectuer des mises à jour directes :

- mises à jour en place ;
- mises à jour directes ;
- mises à jour directes longues.

Les mises à jour directes nécessitent un overhead moindre que les mises à jour différées et sont en général plus rapides car elles permettent de réduire le nombre de balayages de journaux, les journalisations, la consultation des arbres binaires d'index (en limitant les conflits de verrous) et les E/S, car Adaptive Server n'a pas besoin d'extraire de nouveau les pages pour effectuer les modifications basées sur des enregistrements de journal.

Mises à jour en place

Adaptive Server effectue des mises à jour en place chaque fois que cela est possible.

Lorsque Adaptive Server effectue une mise à jour en place, les lignes suivantes de la page ne sont pas déplacées ; les ID de ligne et les pointeurs dans la table des offsets de ligne ne sont pas modifiés.

Pour une mise à jour en place, les conditions suivantes doivent être remplies :

- La longueur de la ligne à modifier ne doit pas changer.
- La colonne à mettre à jour ne peut pas être la clé ou faire partie de la clé d'un index clusterisé sur une table APL. Puisque les lignes d'un index clusterisé sont stockées par ordre de clé, une modification de la clé entraîne la plupart du temps un changement d'emplacement de la ligne.
- Un ou plusieurs index doivent être uniques ou autoriser la duplication.
- L'instruction de mise à jour répond aux conditions évoquées dans la section « [Restrictions concernant les modes de mise à jour avec des jointures](#) », page 109.
- Les colonnes concernées ne sont pas utilisées pour l'intégrité référentielle.
- La colonne ne peut pas avoir de trigger.
- La table ne peut pas être répliquée (à l'aide de Replication Server).

La mise à jour en place est l'opération la plus rapide car elle effectue une seule modification dans la page de données. Elle change toutes les entrées d'index concernées en supprimant les anciennes lignes d'index et en insérant les nouvelles. Cela ne touche que les index dont les clés sont modifiées sans que l'emplacement des lignes et des pages change.

Mises à jour directes

Quand Adaptive Server ne peut pas effectuer de mise à jour en place, il tente une mise à jour directe courte, en modifiant une ligne et en l'écrivant de nouveau sur la page en gardant le même offset. Les lignes suivantes de la page se déplacent vers le haut ou vers le bas, de façon que les données restent côté à côté sur la même page sans que les ID de ligne aient été modifiés. Les pointeurs dans la table des offsets de ligne changent pour tenir compte des nouveaux emplacements.

Une mise à jour directe courte doit remplir les conditions suivantes :

- Soit la longueur des données de la ligne change (mais la ligne tient dans la même page de données), soit la longueur de la ligne ne change pas (mais la table contient un trigger ou elle est répliquée).
- La colonne à mettre à jour ne peut pas être la clé ou faire partie de la clé d'un index clusterisé. Puisqu'Adaptive Server stocke les lignes d'un index clusterisé par ordre de clé, une modification de la clé entraîne la plupart du temps un changement d'emplacement de la ligne.
- Un ou plusieurs index doivent être uniques ou autoriser la duplication.
- L'instruction de mise à jour répond aux conditions évoquées dans la section « [Restrictions concernant les modes de mise à jour avec des jointures](#) », page 109.
- Les colonnes concernées ne sont pas utilisées pour l'intégrité référentielle.

Les mises à jour directes courtes sont quasiment aussi rapides que les mises à jour en place. Elles nécessitent autant d'E/S mais un traitement légèrement plus important. Deux modifications sont apportées à la page de données (une pour la ligne et une autre pour la table des offsets). La mise à jour de toutes les clés d'index modifiées s'effectue par suppression des anciennes valeurs et insertion des nouvelles. Les mises à jour directes courtes n'ont d'incidence que sur les index dont les clés sont modifiées par la mise à jour, étant donné que l'ID de ligne et la page ne sont pas modifiés.

Mises à jour directes longues

Si les données ne tiennent pas sur la même page, Adaptive Server effectue une mise à jour directe longue, si cela est possible. Ce type de mise à jour supprime la ligne de données, y compris toutes les entrées d'index et insère ensuite la ligne et les entrées d'index modifiées.

Adaptive Server utilise un balayage de table ou une lecture d'index pour trouver la ligne sur son emplacement d'origine et la supprimer. Si la table dispose d'un index clusterisé, Adaptive Server détermine le nouvel emplacement de la ligne à l'aide de l'index ; sinon, Adaptive Server insère la nouvelle ligne à la fin de la table.

Une mise à jour directe longue doit remplir les conditions suivantes :

- Soit la longueur d'une ligne de données est modifiée de telle façon que la ligne ne tient plus sur la même page de données et elle est placée sur une autre page, soit la mise à jour porte sur des colonnes clés pour l'index clusterisé.
- L'index utilisé pour rechercher la ligne n'est pas modifié par la mise à jour.
- L'instruction de mise à jour répond aux conditions évoquées dans la section « [Restrictions concernant les modes de mise à jour avec des jointures](#) », page 109.
- Les colonnes concernées ne sont pas utilisées pour l'intégrité référentielle.

Une mise à jour directe longue est le plus lent des processus directs. La suppression et l'insertion sont effectuées sur des pages de données différentes. Toutes les entrées d'index doivent être mises à jour car l'emplacement des lignes change.

Mises à jour différées

Adaptive Server utilise des mises à jour différées lorsque les conditions relatives aux mises à jour directes ne sont pas remplies. Il s'agit du type de mise à jour le plus lent de tous.

Dans une mise à jour différée, Adaptive Server :

- Il localise les lignes de données concernées, avec écriture des enregistrements de journal pour la suppression et l'insertion différées sur les pages de données au fur et à mesure que les lignes sont localisées.
- Il lit les enregistrements de journal pour la transaction et supprime les données sur les pages de données ainsi que toutes les lignes d'index concernées.
- Il lit une nouvelle fois les enregistrements de journal et insère les données sur les pages de données ainsi que les lignes d'index concernées.

Quand les mises à jour différées sont-elles obligatoires ?

Les mises à jour différées sont obligatoires dans les cas suivants :

- mises à jour avec autojointures,
- mises à jour des colonnes utilisées pour l'intégrité référentielle,
- mises à jour d'une table référencée dans une sous-requête associée.

Les mises à jour différées sont également nécessaires dans d'autres cas, tels que :

- Déplacement d'une ligne sur une nouvelle page suite à la mise à jour, alors que l'accès à la table s'effectue à l'aide d'un balayage de table ou d'une lecture d'index clusterisé.
- La table n'accepte pas les lignes dupliquées et il n'y a pas d'index unique pour les retenir.
- Non unicité de l'index utilisé pour rechercher la ligne de données et déplacement de la ligne parce que la mise à jour modifie la clé de l'index clusterisé ou parce que la nouvelle ligne ne tient pas sur la page.

Les mises à jour différées entraînent un overhead supérieur à celui des mises à jour directes car le journal de transactions doit être relu par Adaptive Server pour que les modifications finales des données et des index soient effectuées. Des consultations supplémentaires des arbres d'index sont donc nécessaires.

Par exemple, s'il existe un index clusterisé sur `title`, la requête suivante effectue une mise à jour différée :

```
update titles set title = "Portable C Software" where  
title = "Designing Portable Software"
```

Insertion différée dans les index

Adaptive Server effectue des mises à jour différées dans les index lorsque la mise à jour concerne l'index utilisé pour accéder à la table ou lorsqu'elle concerne les colonnes d'un index unique. Dans ce type de mise à jour, Adaptive Server :

- supprime les entrées d'index en mode direct ;
- met à jour la page de données en mode direct, en écrivant les enregistrements différés des insertions dans l'index ;
- lit les enregistrements de journal pour la transaction et insère les nouvelles valeurs dans l'index en mode différé.

Le mode différé pour les insertions dans l'index doit être utilisé lorsque la mise à jour modifie l'index qui sert à rechercher la ligne ou qu'elle porte sur un index unique. Une requête ne doit mettre à jour qu'une seule ligne qualifiée à la fois : le mode de mise à jour différée de l'index garantit précisément cette unicité au cours du balayage d'index et évite que la requête ne viole prématurément une contrainte d'unicité.

La mise à jour de la [figure 5-8](#) ne modifie que le nom (last name), mais la ligne d'index passe à la page suivante. Pour effectuer la mise à jour, Adaptive Server réalise les opérations suivantes :

- 1 Il lit la page 1133 de l'index, supprime dans cette page la ligne d'index correspondant à « Green » et consigne dans le journal un enregistrement différé pour le balayage d'index.
- 2 Il remplace « Green » par « Hubbard » sur la page de données, en mode direct, et reprend le balayage d'index pour rechercher d'autres lignes à mettre à jour.
- 3 Il insère la nouvelle ligne d'index correspondant à « Hubbard » sur la page 1127.

La [figure 5-8](#) illustre les pages d'index et de données avant la mise à jour différée ainsi que la séquence dans laquelle la mise à jour différée modifie ces pages.

Figure 5-8 : Mise à jour différée de l'index

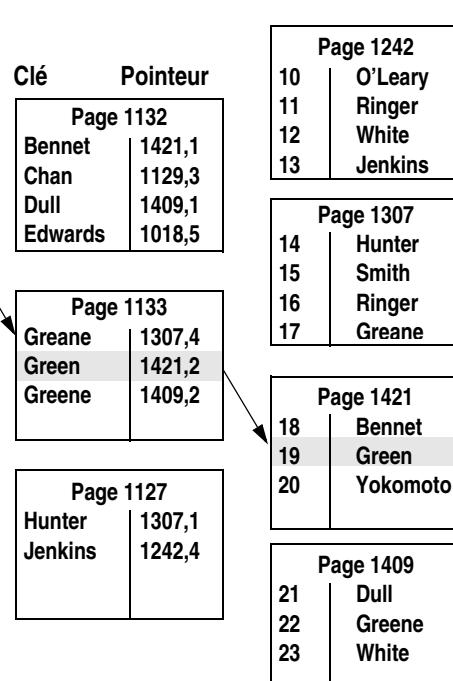
```
update employee
set lname = "Hubbard"
where lname = "Green"
```

Avant la mise à jour

Clé RowID Pointeur		
Page 1001		
Bennet	1421,1	1007
Karsen	1411,3	1009
Smith	1307,2	1062

Clé	RowID	Pointeur
Page 1007		
Bennet	1421,1	1132
Greane	1307,4	1133
Hunter	1307,1	1127

Clé	RowID	Pointeur
Page 1009		
Karsen	1411,3	1315



Page racine

Etapes de
mise à jour

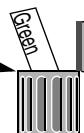
Intermédiaire

Etape 1 : Ecrire les
enregistrements de
journal et supprimer
des lignes d'index

Pages au niveau feuille

Page 1133		
Greane	1307,4	
Greene	1409,2	

Pages de données



Etape 2 : Modifier les
pages de données

Page 1421		
18	Bennet	
19	Hubbard	
20	Yokomoto	

Etape 3 : Lire le
journal, insérer des
lignes d'index

Page 1127		
Hubbard	1421,2	
Hunter	1307,1	
Jenkins	1242,4	

Supposons qu'une mise à jour similaire soit effectuée sur la table titles :

```
update titles
set title = "Computer Phobic's Manual",
    advance = advance * 2
where title like "Computer Phob%"
```

Cette requête risque d'entraîner des problèmes. Si le balayage de l'index non clusterisé sur la colonne title trouve « Computer Phobia Manual », modifie le titre et multiple l'avance par 2, puis trouve la nouvelle ligne d'index « Computer Phobic's Manual » et multiplie de nouveau l'avance par 2, le résultat de l'opération correspondra très peu à la réalité.

Une suppression différée dans un index peut être plus rapide ou beaucoup plus lente qu'une mise à jour directe longue, selon le nombre d'enregistrements de journal à balayer et selon que les pages de journal sont ou non en mémoire cache.

Au cours de la mise à jour d'une ligne de données, il peut y avoir un décalage entre le moment de la suppression et celui de l'insertion d'une ligne d'index. Pendant cet intervalle, aucune ligne d'index ne correspond à la ligne de données. Si une lecture d'index est effectuée au niveau d'isolement 0, aucune valeur de la ligne de données n'est renvoyée.

Restrictions concernant les modes de mise à jour avec des jointures

Les mises à jour et les suppressions qui font intervenir des jointures peuvent être réalisées en mode direct, différé_varcol ou différé_index lorsque la table modifiée est la table externe dans l'ordre de la jointure, ou lorsqu'elle est précédée dans l'ordre de jointure par des tables dont une seule ligne correspond à la recherche.

Jointures et sous-requêtes dans les instructions update et delete

L'utilisation de la clause from pour effectuer des jointures dans les instructions update et delete est une extension Transact-SQL de SQL ANSI. Les sous-requêtes au format SQL ANSI sont utilisables à la place de jointures, dans certaines mises à jour et suppressions.

Cet exemple utilise la syntaxe `from` pour réaliser une jointure :

```
update t1 set t1.c1 = t1.c1 + 50
  from t1, t2
 where t1.c1 = t2.c1
   and t2.c2 = 1
```

L'exemple suivant montre la mise à jour équivalente avec une sous-requête :

```
update t1 set c1 = c1 + 50
  where t1.c1 in (select t2.c1
                   from t2
                  where t2.c2 = 1)
```

Le mode de mise à jour utilisé pour la requête de la jointure varie selon que la table concernée représente la requête la moins imbriquée dans l'ordre de la requête ou non ; dans ce dernier cas, la mise à jour s'effectue en mode différé. La mise à jour avec une sous-requête se déroule toujours en mode direct, différé_varcol, ou différé_index.

Pour une requête qui utilise la syntaxe `from` et qui effectue une mise à jour différée en raison de l'ordre de la jointure, utilisez `showplan` et `statistics io` afin de déterminer si la réécriture de la requête avec une sous-requête peut améliorer les performances. Notez que les requêtes utilisant `from` ne peuvent pas toutes être réécrites afin d'inclure des sous-requêtes.

Suppression et mises à jour dans des triggers et intégrité référentielle

Les triggers qui joignent des tables utilisateur avec des tables supprimées ou insérées sont exécutés en mode différé. Si vous n'utilisez des triggers que pour mettre en oeuvre l'intégrité référentielle et non pour des mises à jour et des suppressions en cascade, vous pouvez éviter les mises à jour différées dans les triggers en utilisant plutôt une intégrité référentielle déclarative.

Optimisation des mises à jour

Les messages showplan fournissent des informations sur le mode de traitement d'une mise à jour (direct ou différé). Si une mise à jour directe est impossible, Adaptive Server met à jour les lignes de données en mode différé. Il arrive parfois que l'optimiseur ne puisse pas connaître le mode choisi ; dans ce cas, deux messages showplan sont fournis :

- Le message « deferred_varcol » indique que la mise à jour risque de modifier la longueur de la ligne parce qu'une colonne de longueur variable est modifiée. Si la ligne modifiée tient sur la page, la mise à jour est effectuée en mode direct ; dans le cas contraire, elle est effectuée en mode différé.
- Le message « deferred_index » indique que les modifications des pages de données et les suppressions dans les pages d'index sont effectuées en mode direct, alors que les insertions dans les pages d'index sont effectuées en mode différé.

Ces types de mises à jour directes dépendent d'informations qui sont uniquement disponibles au moment de l'exécution, puisqu'il convient d'abord d'extraire la page et de l'examiner pour savoir si la ligne tient ou non dans la page.

Conception pour les mises à jour directes

Lors de la phase de conception et d'écriture du code de vos applications, faites attention aux différences que peuvent entraîner des mises à jour différées. Suivez les instructions ci-dessous pour éviter les mises à jour différées :

- Créez au moins un index unique sur la table pour favoriser les mises à jour directes.
- Dans la mesure du possible lors de la mise à jour d'une clé, utilisez dans la clause `where` des colonnes qui ne sont pas des clés.
- Si vous n'utilisez pas de valeurs NULL dans vos colonnes, déclarez-les comme ayant la valeur `not null` dans l'instruction `create table`.

Influence des types de mise à jour et des index sur les modes de mise à jour

Le [tableau 5-2](#) montre de quelle façon les index affectent le mode de mise à jour, dans trois cas de figure spécifiques. Dans tous les cas, les lignes dupliquées ne sont pas autorisées. Pour les cas indexés, l'index porte sur `title_id`. Les trois types de mise à jour envisagés sont :

- Mise à jour d'une colonne clé de longueur variable :

```
update titles set title_id = valuer
    where title_id = "T1234"
```

- Mise à jour d'une colonne non-clé de longueur fixe :

```
update titles set pub_date = valuer
    where title_id = "T1234"
```

- Mise à jour d'une colonne non-clé de longueur variable :

```
update titles set notes = valuer
    where title_id = "T1234"
```

Le [tableau 5-2](#) montre comment un index unique permet un mode de mise à jour plus efficace qu'un index non unique sur la même clé. Remarquez les différences entre le mode direct et le mode différé dans les zones ombrées du tableau. Par exemple, avec un index clusterisé unique, toutes ces mises à jour peuvent se dérouler en mode direct, mais elles doivent être effectuées en mode différé si l'index est non unique.

Pour une table ayant un index clusterisé non unique, un index unique sur n'importe quelle autre colonne de la table améliore les performances de mise à jour. Dans certains cas, vous pouvez ajouter une colonne IDENTITY dans une table afin d'inclure la colonne comme clé dans un index qui, autrement, serait non unique.

Tableau 5-2 : Effets de l'indexation sur le mode de mise à jour

Mise à jour vers :			
Index	Clé de longueur variable	Colonne de longueur fixe	Colonne de longueur variable
Aucun index	non disponible	direct	différé_col_var
Clusterisé, unique	direct	direct	direct
Clusterisé, non unique	différé	différé	différé
Clusterisé, non unique, avec un index unique sur une autre colonne	différé	direct	différé_col_var
Non clusterisé, unique	différé_col_var	direct	direct
Non clusterisé, non unique	différé_col_var	direct	différé_col_var

Si la clé pour un index a une longueur fixe, la seule différence dans les modes de mise à jour par rapport à ceux indiqués dans le tableau se situe au niveau des index non clusterisés. Pour un index non clusterisé et non unique, le mode de mise à jour est différé_index pour les mises à jour sur la clé. Pour un index non clusterisé unique, le mode de mise à jour est direct pour les mises à jour sur la clé.

Si la longueur de varchar ou de varbinary est proche de la longueur maximale, utilisez plutôt char ou binary. Chaque colonne de longueur variable augmente l'overhead de ligne et la probabilité des mises à jour différées.

L'utilisation de max_rows_per_page pour réduire le nombre de lignes admises sur une page augmente la probabilité de mises à jour directes car une mise à jour augmentant la longueur d'une colonne de longueur variable peut souvent tenir sur la même page.

Pour plus d'informations sur l'utilisation de max_rows_per_page, reportez-vous à la section « [Utilisation de max_rows_per_page dans des tables APL](#) », page 219 du manuel *Performances et optimisation : concepts de base*.

Utilisation de sp_sysmon dans l'optimisation des mises à jour

Vous pouvez utiliser showplan pour déterminer si une mise à jour est différée ou directe, mais showplan ne donne pas d'informations détaillées sur le type de mise à jour. Les résultats de la procédure système sp_sysmon ou d'Adaptive Server Monitor fournissent des statistiques détaillées sur les types de mise à jour effectués dans un intervalle échantillon.

Exécutez sp_sysmon lorsque vous optimisez les mises à jour et recherchez les nombres indiquant la diminution des mises à jour différées, du verrouillage et des E/S.

Reportez-vous à la section « [Détail des transactions](#) », page 260 du manuel *Performances et optimisation : contrôle et analyse* pour de plus amples informations.

Méthodes d'accès et coût des requêtes pour des jointures et des sous-requêtes

Ce chapitre présente les méthodes utilisées par Adaptive Server pour accéder aux lignes de données lorsque la requête porte sur plusieurs tables et explique comment l'optimiseur calcule le coût de ces accès.

Sujet	Page
Estimation et optimisation des jointures	115
Jointures à boucles imbriquées	121
Méthodes d'accès et estimation du coût des jointures tri-fusion	126
Activation et désactivation des jointures par fusion	140
Stratégie de redéfinition d'index	141
Optimisation des sous-requêtes	142
Clauses or et unions dans les jointures	153

Lorsqu'il détermine le coût de requêtes portant sur plusieurs tables, Adaptive Server utilise les formules décrites au [Chapitre 5, « Méthodes d'accès et évaluation des coûts de requête sur une seule table »](#).

Estimation et optimisation des jointures

Les jointures extraient des informations de plusieurs tables. Dans une jointure de deux tables, une des tables est la table externe et l'autre, la table interne. Dans la table externe, Adaptive Server cherche les lignes qui répondent aux conditions spécifiées dans la requête. Pour chaque ligne qui correspond à la requête, Adaptive Server doit examiner dans la table interne chaque ligne contenant des colonnes de jointure correspondantes.

L'optimisation des requêtes avec jointure est extrêmement importante du point de vue des performances, car les bases de données relationnelles font très souvent appel aux jointures. En particulier, les requêtes qui effectuent des jointures sur plusieurs tables sont déterminantes pour les performances, comme l'expliquent les sections suivantes.

Dans les résultats de showplan, l'ordre des messages « FROM TABLE » indique l'ordre dans lequel Adaptive Server choisit de joindre les tables.

Reportez-vous à la section [« Message FROM TABLE », page 79](#) du guide *Performances et optimisation : contrôle et analyse* pour obtenir un exemple de jointure de trois tables. Certaines sous-requêtes sont également converties en jointures.

Reportez-vous à la section [« Mise à plat des sous-requêtes in, any et exists », page 142](#).

Traitement

Par défaut, Adaptive Server utilise des jointures à boucles imbriquées ainsi que des jointures par fusion si cette fonctionnalité est activée au niveau du serveur ou au niveau de la session.

Lorsque les jointures par fusion sont activées, Adaptive Server peut les utiliser au même titre que les jointures à boucles imbriquées afin de traiter les requêtes portant sur deux tables ou plus. Pour chaque jointure, l'optimiseur évalue le coût des deux méthodes. Pour les requêtes portant sur plus de deux tables, l'optimiseur analyse le coût de combinaisons de jointures par fusion et à boucles imbriquées et choisit la moins onéreuse.

Jointures et densité des index

Pour estimer le nombre de lignes dans une table jointe qui correspond à une valeur particulière, l'optimiseur utilise une statistique appelée **densité totale**.

Pour de plus amples informations, reportez-vous à la section [« Valeurs de densité et jointures », page 26](#).

L'optimiseur de requêtes utilise la densité totale pour évaluer le nombre de lignes qui seront renvoyées à chaque balayage de la table interne d'une jointure. Par exemple, s'il examine une jointure à boucles imbriquées avec une table de 250 000 lignes et que la table a une densité égale à 0,0001, il estime qu'une moyenne de 25 lignes de la table interne sont en correspondance avec les lignes qualifiées de la table externe.

`optdiag` affiche la densité totale de chacune des colonnes pour lesquelles des statistiques ont été créées. Vous pouvez également voir la densité totale utilisée pour les jointures dans les résultats de `dbcc traceon(302)`.

Densités multicolonne

Adaptive Server gère la densité totale pour chaque sous-ensemble ordonné de colonnes dans un index composé. Si deux tables sont jointes sur plusieurs colonnes d'en-tête d'un index composé, l'optimiseur utilise la densité appropriée lorsqu'il évalue le coût d'une jointure utilisant cet index. Dans une table de 10 000 lignes disposant d'un index sur sept colonnes, la clé totale des sept colonnes a une densité de 1/10 000, alors que la densité de la première colonne n'est que de 1/2, ce qui indique qu'elle peut renvoyer 5 000 lignes.

Jointures et incompatibilité des types de données

Lors de l'optimisation de jointures sur des tables disposant d'index, l'un des problèmes les plus courants est l'incompatibilité des types de données dans les colonnes de jointure. Si le cas se présente, un des types de données doit être converti et l'index ne peut être utilisé que sur un côté de la jointure.

Pour de plus amples informations, reportez-vous à la section « [Types de données incompatibles et optimisation des requêtes](#) », page 28.

Permutations de jointures

Lorsque vous effectuez la jointure de quatre tables maximum, Adaptive Server étudie toutes les permutations possibles pour les tables. Cependant, en raison de sa nature itérative, l'optimiseur d'Adaptive Server examine, pour les requêtes sur plus de quatre tables, les combinaisons des ordres de jointure par groupe de deux à quatre tables à la fois. Il procède à ce regroupement car le nombre des permutations des ordres de jointure augmente de façon exponentielle avec chaque table supplémentaire, ce qui entraîne des calculs très longs pour les jointures importantes. La méthode utilisée par l'optimiseur pour déterminer l'ordre de jointure donne d'excellents résultats pour la plupart des requêtes, avec une utilisation moindre du temps CPU que l'étude de toutes les permutations pour toutes les combinaisons.

Si, dans une jointure, le nombre de tables est supérieur à 25, Adaptive Server réduit automatiquement le nombre de tables examinées simultanément.

Le [tableau 6-1](#) répertorie les valeurs par défaut.

Tableau 6-1 : Tables examinées simultanément durant une jointure

Tables jointes	Tables examinées simultanément
4 – 25	4
26 – 37	3
38 – 50	2

L'optimiseur commence par étudier les deux à quatre premières tables et détermine l'ordre de jointure le mieux adapté. Il sélectionne la table externe du meilleur plan contenant les quatre tables examinées et l'élimine de l'ensemble des tables. Après quoi, il optimise le meilleur sous-groupe de tables parmi celles qui restent. Il continue ainsi jusqu'à ce qu'il ne reste que deux à quatre tables, qu'il optimise également.

Par exemple, supposons une instruction `select` avec la clause `from` suivante :

```
from T1, T2, T3, T4, T5, T6
```

L'optimiseur détermine tous les sous-groupes de quatre tables possibles à partir de ces six tables. Cela donne les 15 combinaisons suivantes :

T1, T2, T3, T4
T1, T2, T3, T5
T1, T2, T3, T6
T1, T2, T4, T5
T1, T2, T4, T6
T1, T2, T5, T6
T1, T3, T4, T5
T1, T3, T4, T6
T1, T3, T5, T6
T1, T4, T5, T6
T2, T3, T4, T5
T2, T3, T4, T6
T2, T3, T5, T6
T2, T4, T5, T6
T3, T4, T5, T6

Pour chacune de ces combinaisons, l'optimiseur étudie tous les ordres de jointure (ou permutations) possibles. Pour chaque sous-groupe de quatre tables, il existe 24 ordres de jointure possibles, pour un total de 360 ($24 * 15$) permutations. Par exemple, pour le jeu de tables T2, T3, T5 et T6, l'optimiseur étudie les 24 ordres suivants :

T2, T3, T5, T6
T2, T3, T6, T5
T2, T5, T3, T6
T2, T5, T6, T3
T2, T6, T3, T5
T2, T6, T5, T3
T3, T2, T5, T6
T3, T2, T6, T5
T3, T5, T2, T6
T3, T5, T6, T2
T3, T6, T2, T5
T3, T6, T5, T2
T5, T2, T3, T6
T5, T2, T6, T3
T5, T3, T2, T6
T5, T3, T6, T2
T5, T6, T2, T3
T5, T6, T3, T2
T6, T2, T3, T5
T6, T3, T2, T5
T6, T3, T5, T2
T6, T5, T2, T3
T6, T5, T3, T2

Supposons que le meilleur ordre de jointure soit :

T5, T3, T6, T2

A ce point de la procédure, T5 est la table la moins imbriquée de la requête.

La deuxième étape consiste à choisir la seconde table la moins imbriquée.

L'optimiseur élimine T5 du champ de sélection pour choisir la suite de l'ordre de jointure. Il doit alors déterminer la place de T1, T2, T3, T4 et T6 dans l'ordre de jointure. Il étudie toutes les combinaisons de quatre tables possibles à partir de ces cinq tables :

T1, T2, T3, T4
T1, T2, T3, T6
T1, T2, T4, T6
T1, T3, T4, T6
T2, T3, T4, T6

L'optimiseur étudie tous les ordres de jointure pour chacune de ces combinaisons, en sachant que la table la plus externe de la jointure est T5. Supposons que le meilleur ordre pour joindre les tables restantes à T5 soit :

T3, T6, T2, T4

Par conséquent, l'optimiseur choisit la table T3 comme venant après la table T5 dans l'ordre de jointure pour la requête. Pour choisir la suite de l'ordre de jointure, il élimine T3 du champ de sélection.

Il reste encore les tables suivantes :

T1, T2, T4, T6

Nous n'avons plus que 4 tables, pour lesquelles l'optimiseur examine les ordres de jointure. Imaginons que l'ordre de jointure optimal soit :

T6, T2, T4, T1

L'ordre de jointure pour la requête sera le suivant :

T5, T3, T6, T2, T4, T1

Jointures externes et permutations

Les jointures externes imposent certaines restrictions à l'ensemble des ordres de jointure possibles. Lorsque le membre interne d'une jointure externe est comparé à un membre externe, ce dernier doit précéder le membre interne dans l'ordre de jointure. Les seules permutations de jointures examinées pour les jointures externes sont celles qui respectent cette condition. Par exemple, les deux requêtes suivantes effectuent des jointures externes, la première avec la syntaxe SQL ANSI, la seconde avec la syntaxe Transact-SQL :

```
select T1.c1, T2.c1, T3.c2, T4.c2
  from T4 inner join T1 on T1.c1 = T4.c1
        left outer join T2 on T1.c1 = T2.c1
        left outer join T3 on T2.c2 = T3.c2
  select T1.c1, T2.c1, T3.c2, T4.c2
  from T1, T2, T3, T4
 where T1.c1 *= T2.c1
   and T2.c2 *= T3.c2
   and T1.c1 = T4.c1
```

Les seuls ordres de jointure pris en considération placent T1 à l'extérieur par rapport à T2 et T2 à l'extérieur par rapport à T3. Les ordres de jointure examinés par l'optimiseur sont :

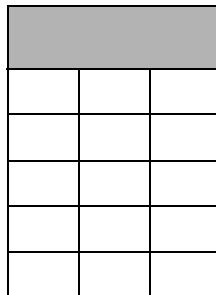
```
T1, T2, T3, T4
T1, T2, T4, T3
T1, T4, T2, T3
T4, T1, T2, T3
```

Jointures à boucles imbriquées

Les jointures à boucles imbriquées améliorent les procédures d'accès lorsque des tables sont indexées sur des colonnes jointes. Le processus de création d'un jeu de résultats pour une jointure par boucles imbriquées comprend l'imbrication des tables et le balayage successif des tables internes pour chaque ligne qualifiée de la table externe, comme l'illustre la [figure 6-1](#).

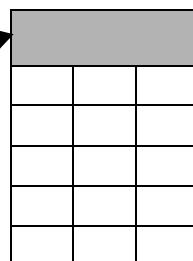
Figure 6-1 : Imbrication des tables dans une jointure à boucles imbriquées

Pour chaque ligne qualifiée de la TableA

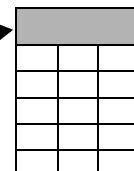


Balayage de la TableB interne

Pour chaque ligne qualifiée de la TableB



Balayage de la TableC la plus interne



Dans la [figure 6-1](#), l'accès aux tables à joindre est imbriqué :

- L'accès à TableA ne se fait qu'une seule fois. Si la table ne contient aucun index exploitable, un balayage de table est effectué. S'il existe un index permettant de réduire les E/S, il est utilisé pour localiser les lignes.
- L'accès à TableB a lieu une fois pour chaque ligne qualifiée de TableA. Par exemple, si 15 lignes de TableA satisfont les conditions de la requête, l'accès à TableB a lieu 15 fois. Si TableB contient un index exploitable sur la colonne jointe, 3 opérations d'E/S peuvent être nécessaires pour lire la page de données à chaque balayage, plus une E/S pour chaque page de données. Le coût d'accès à la TableB sera de 60 E/S logiques.
- L'accès à TableC a lieu une fois pour chaque ligne qualifiée de TableB *chaque fois* qu'il y a accès à TableB. Si 10 lignes de TableB sont retenues pour chaque ligne correspondante de TableA, TableC est balayée 150 fois. Si chaque accès à TableC requiert 3 E/S pour localiser la ligne de données, le coût d'accès à cette table est de 450 E/S logiques.

Si TableC est petite ou dispose d'un index exploitable, le nombre d'E/S sera relativement bas. Si TableC est grande et ne contient pas d'index pertinent sur les colonnes jointes, l'optimiseur peut choisir d'utiliser une jointure tri-fusion ou de redéfinir l'index afin d'éviter des E/S fastidieuses.

Formules de calcul

Pour une jointure à boucles imbriquées avec deux tables, la formule d'évaluation du coût est celle-ci :

Coût de la jointure = coût d'accès à A +
nombre de lignes qualifiées dans A * pages de B à analyser pour
chaque ligne qualifiée

Avec des tables supplémentaires, le coût d'une jointure à boucles imbriquée est :

Coût d'accès à la table externe
+ (nombre de lignes qualifiées dans la table externe) *
(coût d'accès à la table interne)
+ ...
+ (nombre de lignes qualifiées de la table précédente) *
(coût d'accès à la table la plus interne)

Définition des tables internes et externes

En général, la table externe est celle caractérisée par :

- le plus petit nombre de lignes qualifiées et/ou
- le plus grand nombre d'E/S requises pour localiser les lignes.

En général, la table interne est celle caractérisée par :

- le plus grand nombre de lignes qualifiées et/ou
- le plus petit nombre de lectures requises pour localiser les lignes.

Par exemple, si vous joignez une grande table non indexée à une table plus petite disposant d'index sur la clé de jointure, l'optimiseur choisit :

- la grande table comme table externe, de sorte qu'elle n'est balayée qu'une fois ;
- la table indexée comme table interne, de sorte que chaque fois qu'elle est consultée, quelques lectures suffisent pour retrouver les lignes qualifiées.

Autojointure

Cette jointure est utilisée pour comparer les valeurs d'une colonne d'une table. Dans la mesure où cette opération implique une jointure d'une table en son sein, vous devez attribuer à la table deux noms temporaires ou alias, qui sont ensuite utilisés pour qualifier les noms des colonnes dans le reste de la requête.

Identifiez les optimisations en fonction des conditions dans lesquelles le mode immédiat est possible lors de la mise à jour. Dans le cas du projet de mise à jour en place, les mises à jour impliquant une autojointure sont toujours effectuées en mode différé. Par exemple :

```
update t1 set t1.c1 = t1.c1 + 1  
FROM t1 a, t1 b  
where a.c1 = b.c2  
  
and  
delete t1 FROM t1 a, t1 b WHERE a.c1 = b.c2
```

est toujours traité en mode différé. Cette routine met en œuvre des contrôles pour les règles suivantes, qui doivent être respectées en vue d'exécuter la mise à jour en mode immédiat :

- 1 S'il n'existe aucune autojointure de ce type, mais que la requête référence plusieurs tables dans sa liste FROM, il sera possible d'effectuer la mise à jour en mode immédiat.

- 2 Dans le cas d'une requête de mise à jour, si cette requête référence plusieurs tables dans la liste FROM, vérifiez si la table de curseurs de lecture est la première dans l'ordre de jointure :
- Si elle occupe la première place, la mise à jour pourra être effectuée en mode immédiat, dans la mesure où toutes les tables en aval sont balayées dans une jointure de contrôle de présence.
 - Si elle n'est pas la première table de l'ordre de jointure, vérifiez si toutes les tables qui la précèdent sont des tables issues de sous-requêtes mises à plat auxquelles l'optimiseur a appliqué une des nombreuses techniques (jointure unique, filtrage de n-uplets et/ou redéfinition d'index unique) afin d'être certain qu'une seule ligne sera introduire dans le plan de jointure final.
 - Si toutes les tables qui précèdent la table de curseurs de lecture et qui proviennent du bloc de requêtes externes ne renvoient qu'une seule ligne, chaque ligne de la table de curseurs de lecture ne sera qualifiée qu'une seule fois.

Vous êtes ainsi certain que les lignes de la table cible ne se qualifieront qu'une fois.

- 3 Dans le cas d'une requête de suppression, il n'est pas nécessaire que la table cible soit la première table de l'ordre de jointure. En effet, même si une ligne de la table cible est qualifiée plusieurs fois en raison des conditions de jointure, grâce à la suppression directe, la ligne est supprimée lors de la « première » qualification et toutes les qualifications ultérieures reçoivent la condition « row not found ».

Lorsque ces règles sont utilisées, les requêtes suivantes peuvent être exécutées en mode de mise à jour immédiat.

Exemple 1 : Sous-requête non mise à plat créant 3 tables dans le nœud ROOT principal.

```
update t1 set t1.c1 = t1.c1 + 1
where t1.c2 NOT IN (select t2.c2 from t2)
```

Exemple 2 : Sous-requête mise à plat

```
update t1 set t1.c1 = t1.c1 + 1
FROM t1
where t1.c2 IN (select t2.c2 from t2)
```

Dans les deux cas, même si le nombre de tables du nœud ROOT principal est égal à 3, ces requêtes peuvent être exécutées en mode de mise à jour immédiat.

Exemple 3 :

```
update t1 set t1.c1 = t1.c1 + 1  
  FROM t1, t2  
 where t1.c1 = t2.c1
```

Résultat :

Data is:	t1	t2
	---	---
	1	1
	1	2

Si l'ordre de jointure est $t2 \rightarrow t1$ et que la mise à jour est effectuée en mode immédiat, les lignes de t1 se présenteront comme suit :

```
[ (1), (1) ] -> [ (2), (2) ]
```

Méthodes d'accès et estimation du coût des jointures tri-fusion

Il existe quatre méthodes d'exécution possibles pour les jointures par fusion :

- Jointure par fusion complète : les deux tables à joindre possèdent des index exploitables sur les colonnes de jointure. Il n'est pas nécessaire de trier les tables mais il est possible de les fusionner en utilisant les index.
- Jointure par fusion à gauche : trie la table interne dans l'ordre de jointure, puis la fusionne avec la table externe de gauche.
- Jointure par fusion à droite : trie la table externe dans l'ordre de jointure, puis la fusionne avec la table interne de droite.
- Jointure tri-fusion : trie les deux tables, puis les fusionne.

Les jointures par fusion opèrent toujours sur des tables stockées, à savoir des tables utilisateur ou des tables de travail créées pour la jointure par fusion.

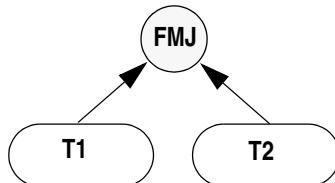
Lorsqu'une table de travail est nécessaire, elle est triée dans l'ordre adéquat sur la clé de jointure puis la fusion est effectuée. Le calcul du coût des jointures par fusion comportant un tri tient compte du coût estimé des E/S nécessaires pour créer et trier une table de travail. Pour les jointures par fusion complète, le seul coût encouru est celui du balayage des tables.

La [figure 6-2](#) représente les types de jointures par fusion.

Figure 6-2 : Types de jointure par fusion

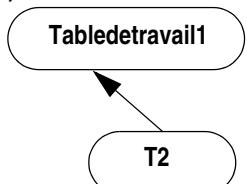
Jointure par fusion complète (FMJ)

Etape 1

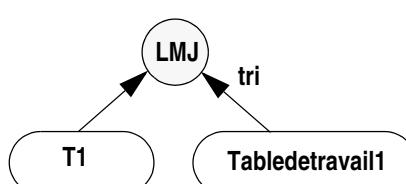


Jointure par fusion à gauche (LMJ)

Etape 1

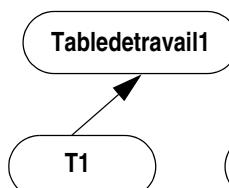


Etape 2

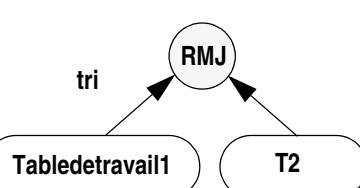


Jointure par fusion à droite (RMJ)

Etape 1

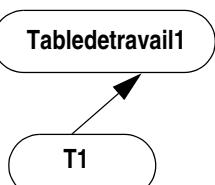


Etape 2

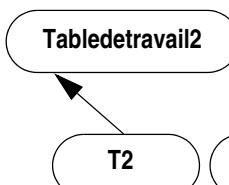


Jointure par tri-fusion (SMJ)

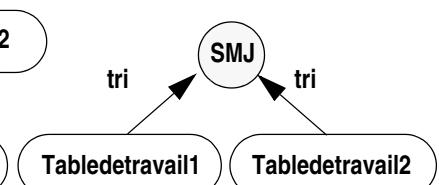
Etape 1



Etape 2



Etape 3



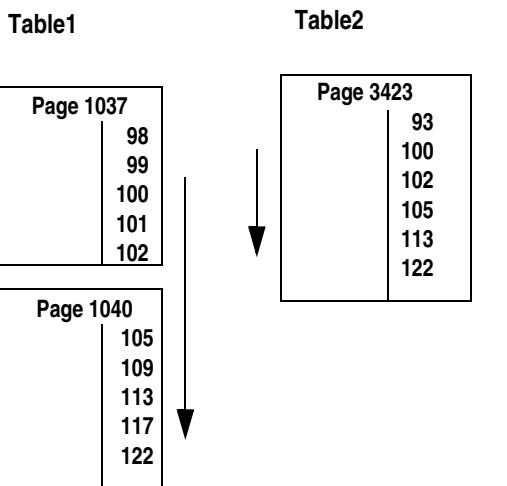
Déroulement d'une jointure par fusion complète

Si Table1 et Table2 possèdent toutes deux des index sur la clé de jointure, cette requête peut utiliser une jointure par fusion complète :

```
select *
  from Table1, Table2
 where Table1.c1 = Table2.c2
   and Table1.c1 between 100 and 120
```

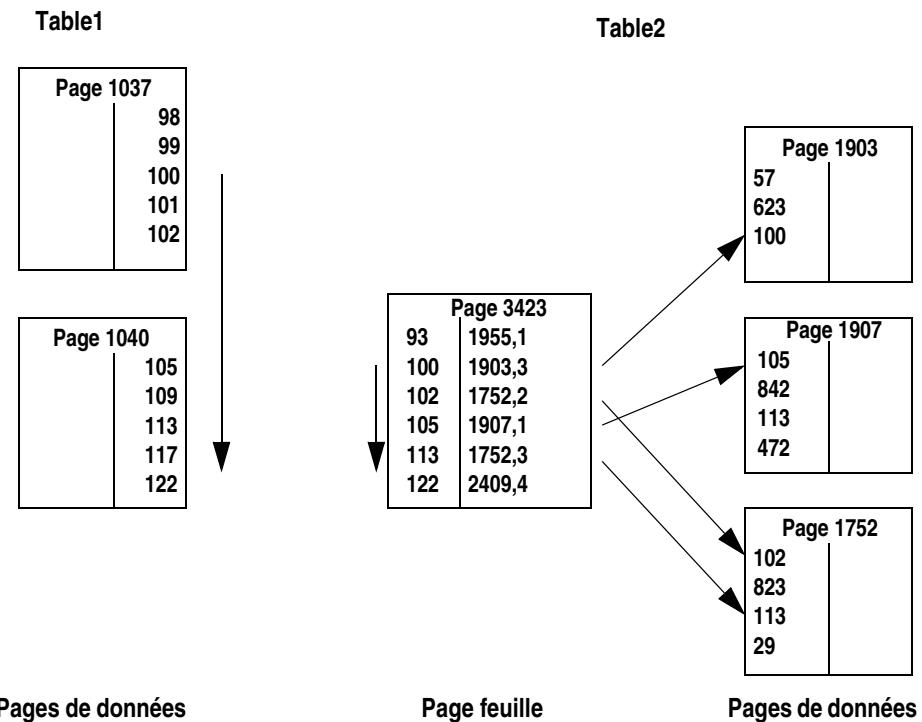
Si les deux tables sont verrouillées au niveau de toutes les pages à l'aide d'index clusterisés et que Table1 est choisie comme table externe, l'index sert à positionner la recherche sur la page de données au niveau de la ligne dont la valeur est égale à 100. L'index de Table2 est également utilisé pour positionner le balayage au niveau de la première ligne de Table2 dont la colonne de jointure est égale à 100. A partir de ce moment, les lignes des deux tables sont renvoyées à mesure que le balayage progresse.

Figure 6-3 : Balayage de fusion en mode série sur deux tables avec index clusterisés



Il est également possible d'effectuer des jointures par fusion en utilisant des index non clusterisés. L'index sert à positionner le balayage sur la première valeur correspondante de la page feuille de l'index. Pour chaque ligne qualifiée, les pointeurs d'index permettent d'accéder aux pages de données. La [figure 6-4](#) représente un balayage par fusion complète avec un index non clusterisé sur la table interne.

Figure 6-4 : Balayage de fusion complète avec un index non clusterisé sur la table interne



Déroulement d'une jointure par fusion à droite ou à gauche

Une jointure par fusion à droite ou à gauche opère toujours sur une table utilisateur et sur une table de travail créée pour la jointure par fusion.

Le processus se déroule en deux étapes :

- 1 Une table ou un groupe de tables est analysé et les résultats sont insérés dans une table de travail.
 - 2 La table de travail est triée puis fusionnée avec l'autre table de la jointure, à l'aide de l'index.

Déroulement d'une jointure par tri

Une jointure tri-fusion se déroule en trois étapes, car elle a en entrée des tables de travail triées :

- 1 Une table (ou un groupe de tables) est analysée et les résultats sont insérés dans une table de travail. Celle-ci constitue la table externe de la fusion.
- 2 Une autre table est analysée et les résultats sont insérés dans une seconde table de travail. Celle-ci constitue la table interne de la fusion.
- 3 Chaque table de travail est triée puis les deux ensembles de résultats sont fusionnés.

Exemple de jointures mixtes

Cette requête exécute une combinaison de jointures par fusion et de jointures par boucles imbriquées :

```
select pub_name, au_lname, price
  from titles t, authors a, titleauthor ta,
       publishers p
 where t.title_id = ta.title_id
   and a.au_id = ta.au_id
   and p.pub_id = t.pub_id
   and type = 'business'
   and price < $25
```

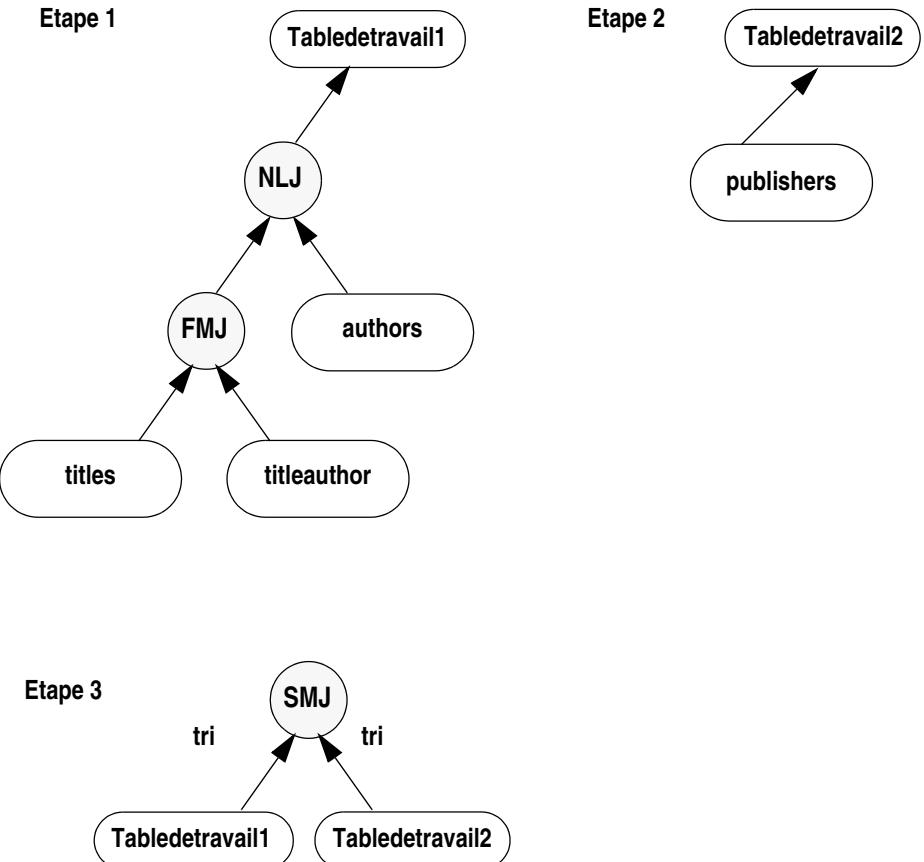
Adaptive Server exécute cette requête en trois étapes :

- Dans l'étape 1, trois processus de travail balayent `titles` en tant que table externe, en effectuant une jointure par fusion complète avec `titleauthor`, puis une jointure par boucles imbriquées avec `authors`. Aucun tri n'est nécessaire pour la jointure par fusion complète. `titles` possède un index clusterisé sur `title_id`. L'index sur `titleauthor`, `ta_ix`, contient `title_id` et `au_id`, de sorte qu'il couvre la requête. Les résultats sont stockés dans `Tabledetravail1`, en vue d'être utilisés dans la jointure tri-fusion qui sera réalisée à l'étape 3.
- Dans l'étape 2, la table `publishers` est balayée et les colonnes nécessaires (`pub_name` et `pub_id`) sont enregistrées dans `Tabledetravail2`.

- Dans l'étape 3 :
 - Tabledetravail1 est triée dans l'ordre des colonnes de la jointure, sur pub_id.
 - Tabledetravail2 est triée dans l'ordre adéquat sur pub_id.
 - Les résultats triés sont fusionnés.

La [figure 6-5](#) illustre ces différentes étapes.

Figure 6-5 : Étapes d'une jointure par fusion



Messages **showplan** pour les jointures tri-fusion

Les messages showplan pour chaque type de jointure par fusion apparaissent selon différentes combinaisons :

- Jointure par fusion complète : il n'y a pas de messages « FROM TABLE Worktable », seulement des messages « inner table » et « outer table » pour les tables sous-jacentes de la requête.
- Jointure par fusion à droite : la « table externe » est toujours une table de travail.
- Jointure par fusion à gauche : la « table interne » est toujours une table de travail.
- Jointure tri-fusion : les deux tables sont des tables de travail.

Pour de plus amples informations, reportez-vous à la section « [Messages décrivant les méthodes d'accès, la mise en mémoire cache et le coût des E/S](#) », page 98 du guide *Performances et optimisation : contrôle et analyse*.

Estimation du coût des jointures par fusion

Le coût total des jointures par fusion dépend des éléments suivants :

- Le type de jointure par fusion.
 - Les jointures par fusion complète ne nécessitent ni tri ni tables de travail.
 - Pour les jointures par fusion à gauche et à droite, un côté de la jointure est sélectionné et inséré dans une table de travail puis trié.
 - Pour les jointures tri-fusion, les deux côtés de la jointure sont copiés dans des tables de travail et chacune de ces tables est triée.
- Le type d'index utilisé pour accéder aux tables pendant la phase de fusion.
- Le plan de verrouillage de la table sous-jacente : les modèles de calcul des coûts pour la plupart des balayages diffèrent selon le verrouillage (DOL ou APL). Le coût d'accès des index clusterisés dans des tables DOL est plus comparable au coût d'accès des index non clusterisés.
- Le mode d'exécution de la requête, à savoir série ou parallèle.
- La présence dans la table externe de valeurs dupliquées pour la clé de jointure.

En général, lors de la comparaison des coûts entre une jointure à boucles imbriquées et une jointure par fusion pour les mêmes tables et avec les mêmes index, le coût associé à la table externe ne change pas. L'accès à la table interne est moins coûteux pour une jointure par fusion car le balayage reste positionné sur les pages de niveau feuille à mesure que les valeurs correspondantes sont renvoyées, ce qui réduit le coût des E/S logiques associées à chaque balayage de l'index à partir du niveau racine.

Estimation du coût d'une jointure par fusion complète avec des valeurs uniques

Si une jointure par fusion complète est réalisée en mode série et qu'aucun tri de tables n'est nécessaire, le coût d'une jointure par fusion sur T1 et T2 équivaut au coût total de balayage des deux tables, dès lors que les valeurs de la jointure sont uniques.

$$\begin{array}{lcl} \text{Coût de jointure} & & \text{coût du balayage de T1 +} \\ \text{par fusion} & = & \text{coût du balayage de T2} \end{array}$$

L'économie d'une jointure par fusion sur une jointure à boucles imbriquées équivaut à :

- Pour une jointure à boucles imbriquées, l'accès à la table interne de la jointure commence au niveau de la page racine de l'index, pour chaque ligne qualifiée de la table externe.
- Pour une jointure par fusion complète, les niveaux supérieurs de l'index sont utilisés pour le premier accès en vue de positionner le balayage :
 - sur la page de niveau feuille de l'index, pour les index non clusterisés et les index clusterisés de tables DOL,
 - sur la page de données, s'il existe un index clusterisé dans une table APL.

Il n'est pas nécessaire de lire les niveaux supérieurs de l'index pour chaque ligne qualifiée de la table externe.

Exemple : tables APL avec index clusterisés

Dans les tables APL qui utilisent des index clusterisés pour accéder aux données, les arguments de recherche sur l'index servent à positionner la recherche sur la première ligne qualifiée de chaque table. Le coût total de la requête équivaut au coût du balayage avant des pages de données de chaque table. Par exemple, avec des index clusterisés sur t1(c1) et t2(c1), la requête sur deux tables APL peut utiliser une jointure par fusion complète :

```
select t1.c2, t2.c2
  from t1, t2
 where t1.c1 = t2.c1
   and t1.c1 >= 1000 and t1.c1 < 1100
```

Si 100 lignes de la table t1 et 100 lignes de la table t2 sont qualifiées et que chaque table contient 10 lignes par page et un index de trois niveaux, les coûts seront les suivants :

- trois pages d'index pour positionner le balayage sur la première ligne qualifiée de t1,
- balayage de 10 pages de t1,
- trois pages d'index pour positionner le balayage sur la première ligne qualifiée de t2,
- balayage de 10 pages de t2.

Estimation du coût d'une jointure par fusion complète avec des valeurs en double

Si, dans une jointure par fusion, la table externe contient des valeurs dupliquées, un accès à la table interne est nécessaire depuis la page racine de l'index pour chaque valeur dupliquée. Prenons la même requête que celle de l'exemple précédent :

```
select t1.c2, t2.c2
  from t1, t2
 where t1.c1 = t2.c1
   and t1.c1 >= 1000 and t1.c1 < 1100
```

Si t1 est la table externe et que certaines de ses lignes contiennent des valeurs dupliquées, de telle sorte qu'il y a 120 lignes entre les valeurs 1 000 et 1 100, avec 20 valeurs dupliquées, chaque fois qu'une valeur dupliquée est atteinte, le balayage de t2 redémarre au niveau de la page racine de l'index. Si une ligne de t2 correspond à chaque valeur de t1, les coûts d'E/S pour la requête sont les suivants :

- trois pages d'index pour positionner le balayage sur la première ligne qualifiée de t1 ;
- balayage de 12 pages de t1 ;
- trois pages d'index pour positionner le balayage sur la première ligne qualifiée de t2, plus une E/S pour lire la page de données ;
- Pour les lignes restantes :
 - Si la valeur extraite de t1 est dupliquée, le balayage de t2 redémarre à partir de la page racine de l'index.
 - Pour toutes les valeurs de t1 qui ne sont pas dupliquées, le balayage reste positionné au niveau feuille de t2. Dans la table interne, le balayage reste positionné au niveau feuille pendant que les lignes sont renvoyées, jusqu'à ce que la valeur dupliquée suivante de la table externe impose au balayage de redémarrer à la page racine.

La formule ci-après permet de connaître le coût du balayage de la table interne pour une jointure par fusion :

$$\begin{aligned} \text{Coût de balayage de interne} = & \quad \text{nbre valeurs dupliquées *} \\ & (\text{niveau d'index} + \text{taille du balayage}) \\ & + \text{nbre valeurs uniques * taille du balayage} \end{aligned}$$

La *taille du balayage* correspond au nombre de pages de la table interne qu'il convient de lire pour chaque valeur de la table externe. Pour les tables contenant plusieurs lignes internes correspondantes, la taille du balayage équivaut au nombre moyen de pages à lire pour chaque ligne externe.

Evaluation du coût des tris

Le coût des opérations de tri lors de jointures tri-fusion dépend de :

- la taille des tables de travail, dépendante elle-même du nombre de colonnes et de lignes sélectionnées,
- la valeur du paramètre de configuration `number of sort buffers`, qui détermine le nombre de pages de cache qu'il est possible d'utiliser.

Ces variables influent sur le nombre des fusions requises pour trier la table de travail.

Taille de la table de travail pour les jointures tri-fusion

Lorsqu'une table de travail est créée pour une jointure par fusion exigeant un tri, seules les colonnes nécessaires pour le jeu de résultats et pour les jointures ultérieures dans la requête sont insérées dans la table de travail. Lorsque la table de travail pour `titles` est créée pour la jointure montrée à la [figure 6-5, page 131](#) :

- `Tabledetravail1` contient les colonnes `price` et `authors.state`, car elles font partie du jeu de résultats, ainsi que la colonne `pub_id`, nécessaire pour une jointure ultérieure.
- `Tabledetravail2` contient la colonne `publishers.state`, car elle fait partie du jeu de résultats, et la colonne `pub_id`, nécessaire pour la phase de fusion.

La colonne `type` est utilisée comme argument de recherche pendant la sélection des lignes de `titles`, mais comme elle n'est plus utilisée ensuite, ni dans la requête ni dans le jeu de résultats, elle n'est pas insérée dans la table de travail.

Chaque tri réalisé pour une jointure par fusion peut utiliser un nombre maximal de buffers (défini par `number of sort buffers`) pour les phases de tri intermédiaires. Les buffers nécessaires aux tris des tables de travail sont alloués depuis le cache utilisé par `tempdb`. Si le nombre de pages à trier est inférieur à la valeur de `number of sort buffers`, le nombre de buffers réservés au tri équivaut au nombre de pages de la table de travail.

Impossibilité d'utiliser des jointures par fusion

Les jointures par fusion ne peuvent pas être appliquées dans les cas suivants :

- Jointures utilisant <, >, <=, >=, ou != sur les colonnes de la jointure.
- Jointures externes, c'est-à-dire des requêtes utilisant *= ou =* ainsi que left join et right join.
- Requêtes qui contiennent une colonne de type text ou image ou des objets Java dans la liste de sélection ou dans une clause where.
- Sous-requêtes qui ne sont pas mises à plat ou matérialisées dans des requêtes parallèles.
- Mises à jour et suppressions de plusieurs tables, telles que :

```
update R set a = 5
           from R, S, T
           where ...
```

- Jointures qui effectuent des contrôles d'intégrité référentielle pour les commandes insert, update et delete. Ces jointures sont générées en interne afin de vérifier l'existence des valeurs de colonne. Il s'agit généralement de jointures qui renvoient une seule valeur de la table référencée. Souvent, ces jointures utilisent des index. L'utilisation d'une jointure par fusion pour des vérifications de contrainte ne présenterait en effet aucun avantage.
- Lorsque le nombre d'octets d'une ligne dans une table de travail excède la limite de la taille de page (soit 1 960 octets de données utilisateur) ou la limite du nombre de colonnes (1 024). Si les colonnes de la liste de sélection et de la jointure requise génèrent une table de travail dépassant ces limites, l'optimiseur n'envisage pas la réalisation d'une jointure par fusion à ce stade du plan d'exécution de requête.
- Lorsque l'usage des tables de travail pour une jointure par fusion requiert plus que le nombre autorisé de tables de travail dans une requête (14).

Il existe certaines limites à l'utilisation des jointures par fusion dans l'ordre de jointure :

- Les jointures par fusion ne peuvent avoir lieu qu'avant une jointure d'existence, jamais après. Certaines requêtes distinct sont converties en jointures d'existence et les jointures par fusion ne sont pas utilisables dans celles-ci.
- Les jointures de fusion complète et les jointures par fusion à gauche ne sont possibles que sur les tables externes dans l'ordre de jointure.

Utilisation des processus de travail

Lorsque le traitement parallèle est activé, les jointures par fusion peuvent utiliser plusieurs processus de travail pour réaliser :

- le balayage qui sélectionne les lignes en vue de leur insertion dans les tables de travail,
- les opérations de tri des tables de travail,
- la jointure par fusion et les jointures ultérieures dans la même étape.

Pour de plus amples informations, reportez-vous à la section « [Balayages en parallèle reposant sur des intervalles](#) », page 202.

Recommandations pour améliorer les performances des fusions

Voici quelques suggestions pour améliorer les performances des jointures tri-fusion :

- Ne sélectionnez que les colonnes nécessaires pour les tables utilisées dans les jointures par fusion, afin de réduire la taille des tables de travail. Evitez d'utiliser `select *` à moins que toutes les colonnes ne soient indispensables. Ceci réduit la charge de travail au niveau de tempdb ainsi que le coût lié au tri des tables de résultat.
- Pour connaître les répercussions éventuelles des jointures par fusion sur les performances ou les problèmes d'espace susceptibles de se poser dans tempdb, reportez-vous au [Chapitre 14, « Présentation des plans abstraits »](#), qui explique comment les plans abstraits d'exécution des requêtes permettent de déterminer les requêtes du système qui utilisent des jointures par fusion.

- Examinez les possibilités de couvertures par index. Celles-ci sont possibles par exemple pour des jointures qui ont la forme suivante :

```
select t1.c3, t3.c4
from t1, t2, t3
where t1.c1 = t2.c1 and t2.c2 = t3.c2
and ...
```

et les colonnes de la table t_2 ne sont pas dans la liste de sélection, ou seules les colonnes de jointure s'y trouvent. Un index sur les colonnes de jointure $t_2(c_1, c_2)$ couvre la requête, ce qui permet d'éviter l'accès aux pages de données de t_2 .

- Les jointures par fusion peuvent utiliser des index créés dans l'ordre croissant ou décroissant lorsque deux tables sont jointes sur plusieurs colonnes, comme ci-dessous :

$A.c1 = B.c1 \text{ and } A.c2 = B.c2 \text{ and } A.c3 = B.c3$

L'ordre des colonnes spécifié pour les index doit être la reproduction exacte, ou être exactement l'inverse, de celui des colonnes utilisées comme prédictats de jointure lors du calcul du coût et de l'accès aux données. S'il y a incompatibilité au niveau de l'ordre de la deuxième colonne ou des colonnes suivantes, seules les colonnes compatibles sont utilisées, les autres servant à filtrer les résultats après extraction de la ligne. Le tableau ci-dessous fournit quelques exemples illustrant la requête précédente.

Ordre de création de l'index	Clauses utilisées comme prédictats de jointure
$A(c1 \text{ asc}, c2 \text{ asc}, c3 \text{ asc})$ $B(c1 \text{ asc}, c2 \text{ asc}, c3 \text{ asc})$	Les trois clauses.
$A(c1 \text{ asc}, c2 \text{ asc}, c3 \text{ asc})$ $B(c1 \text{ desc}, c2 \text{ desc}, c3 \text{ desc})$	Les trois clauses.
$A(c1 \text{ asc}, c2 \text{ asc}, c3 \text{ asc})$ $B(c1 \text{ desc}, c2 \text{ desc}, c3 \text{ asc})$	Les deux premières clauses sont utilisées comme prédictats de jointure et la troisième sert à filtrer les résultats.
$A1(c1 \text{ asc}, c2 \text{ desc}, c3 \text{ desc})$ $B1(c1 \text{ desc}, c2 \text{ desc}, c3 \text{ asc})$	Seule la première clause est utilisée comme prédictat de jointure. Les deux autres servent à filtrer les résultats.

L'ordre des clés d'index est généralement choisi dans le but d'éliminer le coût des opérations de tri des requêtes de type `order by`. L'utilisation d'un ordre compatible pour les tables souvent jointes peut également réduire les coûts de jointure.

Activation et désactivation des jointures par fusion

Vous pouvez activer et désactiver les jointures par fusion au niveau du serveur et de la session, en utilisant la commande `set sort_merge`, ou au niveau du serveur avec le paramètre de configuration `enable sort-merge joins and JTC`. Ce paramètre active et désactive également la fermeture transitive au niveau des clauses de jointures.

Au niveau du serveur

Pour activer les jointures par fusion au niveau du serveur, définissez `enable sort-merge joins and JTC` à 1. La valeur par défaut est 0, ce qui indique que les jointures par fusion ne sont pas prises en considération. Lorsque ce paramètre a la valeur 1, les jointures par fusion et la fermeture transitive de jointure sont prises en compte pour les équijoинtures. Si les jointures par fusion sont désactivées au niveau du serveur, il est possible de les activer le temps d'une session avec `set sort_merge`.

La fermeture transitive de jointure peut aussi être activée de manière indépendante au niveau de la session, avec `set jtc on`.

Reportez-vous à la section « [Activation et désactivation de la fermeture transitive de jointures](#) », page 58.

Le paramètre de configuration est dynamique et peut être réinitialisé sans redémarrage du serveur.

Au niveau session

Pour activer les jointures par fusion, utilisez :

```
set sort_merge on
```

Pour désactiver les jointures par fusion, utilisez :

```
set sort_merge off
```

Le paramètre défini au niveau session l'emporte sur le paramètre défini au niveau du serveur. Vous pouvez utiliser des jointures par fusion dans une session ou une procédure stockée même si elles sont désactivées au niveau du serveur.

Stratégie de redéfinition d'index

Lorsqu'une table est volumineuse et ne contient pas d'index exploitable pour une jointure, l'optimiseur étudie la possibilité de réaliser une jointure tri-fusion, mais aussi de créer et trier une table de travail, ou d'utiliser une jointure à boucles imbriquées.

Le processus de génération d'une table de travail avec un index clusterisé et la réalisation d'une jointure par boucles imbriquées est connu sous le nom de **redéfinition d'index**.

Comme une jointure tri-fusion, cette redéfinition d'index balaie les tables et copie les lignes qualifiées dans une table de travail. Mais au lieu de procéder au tri et à la fusion utilisés dans une jointure par fusion, Adaptive Server crée un index clusterisé temporaire sur la colonne de jointure pour la table interne. Dans certains cas, la création et l'utilisation de l'index clusterisé sont moins onéreuses qu'une jointure tri-fusion.

Les étapes de la stratégie de redéfinition d'index sont les suivantes :

- création d'une table de travail,
- insertion des colonnes nécessaires à partir des lignes qualifiées,
- création d'un index clusterisé sur les colonnes de jointure de la table de travail,
- utilisation de l'index clusterisé dans la jointure pour extraire les lignes qualifiées de chaque table.

Le coût le plus important dans une stratégie de redéfinition d'index est lié au temps et aux E/S nécessaires à la création de la table de travail ainsi qu'à la construction de l'index clusterisé sur cette table. Adaptive Server n'utilise la redéfinition d'index que lorsqu'elle est plus avantageuse que la jointure par fusion ou des balayages de tables répétés.

Un message de showplan indique si Adaptive Server utilise la stratégie de redéfinition d'index et inclut d'autres messages illustrant les étapes utilisées pour construire les tables de travail.

Reportez-vous à la section « [Message de reformatage](#) », page 114 du guide *Performances et optimisation : contrôle et analyse*.

Optimisation des sous-requêtes

Les sous-requêtes utilisent des méthodes spéciales d'optimisation pour améliorer les performances, notamment :

- Mise à plat : conversion de la sous-requête en jointure
- Matérialisation : stockage des résultats d'une sous-requête dans une table de travail
- Court-circuitage : placement d'une sous-requête en dernière position dans l'ordre d'exécution
- Mise en mémoire cache des résultats d'une sous-requête : enregistrement des résultats des exécutions

Vous trouverez dans les sections suivantes la description de ces stratégies.

Reportez-vous à la section [« Messages showplan pour les sous-requêtes », page 124](#) du guide *Performances et optimisation : contrôle et analyse* pour obtenir des explications sur les messages showplan pour le traitement des sous-requêtes.

Mise à plat des sous-requêtes *in*, *any* et *exists*

Adaptive Server peut transformer certaines sous-requêtes de prédicat quantifié en jointure. Les sous-requêtes introduites par *in*, *any* ou *exists* sont de type prédicat quantifié. Chaque ligne de résultat de la requête externe est renvoyée une seule fois, si la condition de la sous-requête est vraie (TRUE).

Opportunité d'une mise à plat

- Pour tous les niveaux d'imbrication de sous-requêtes, par exemple :

```
select au_lname, au_fname
  from authors
 where au_id in
       (select au_id
        from titleauthor
       where title_id in
             (select title_id
              from titles
             where type = "popular_comp") )
```

- Pour des sous-requêtes multiples dans la requête externe, par exemple :

```
select title, type
from titles
where title in
  (select title
   from titles, titleauthor, authors
   where titles.title_id = titleauthor.title_id
   and titleauthor.au_id = authors.au_id
   and authors.state = "CA")
and title in
  (select title
   from titles, publishers
   where titles.pub_id = publishers.pub_id
   and publishers.state = "CA")
```

Exceptions de mise à plat

Une sous-requête introduite par `in`, `any` ou `exists` ne peut pas être transformée si l'une des conditions suivantes est remplie :

- La sous-requête est en corrélation avec la requête principale et contient un ou plusieurs agrégats.
- La sous-requête figure dans la liste de sélection ou dans la clause `set` d'une instruction `update`.
- La sous-requête est liée à la requête externe avec `or`.
- La sous-requête fait partie d'un prédictat `isnull`.
- La sous-requête est la plus externe dans une expression `case`.

Si la sous-requête calcule un agrégat scalaire, une matérialisation est effectuée, de préférence à la mise à plat.

Reportez-vous à la section « [Matérialisation des résultats de sous-requête](#) », [page 148](#).

Méthodes de mise à plat

Adaptive Server utilise l'une des méthodes de mise à plat suivantes pour résoudre une sous-requête de prédictat quantifié utilisant une jointure :

- Jointure standard : si les conditions d'unicité dans la sous-requête permettent le renvoi d'un jeu unique de valeurs, la sous-requête peut être mise à plat pour utiliser une jointure standard.
- Jointure de contrôle de présence (ou semi-jointure) : au lieu de balayer une table afin de renvoyer toutes les valeurs correspondantes, cette jointure renvoie TRUE lorsqu'elle détecte la première valeur correspondante, puis arrête le traitement. Si aucune valeur correspondante n'est trouvée, elle renvoie FALSE.
- Redéfinition d'index unique : le jeu de résultats de la sous-requête est inséré dans une table de travail, soumis à un tri pour éliminer les valeurs dupliquées, puis un index clusterisé est créé sur la table de travail. L'index clusterisé est utilisé pour réaliser une jointure standard.
- Optimisation du tri pour éliminer les valeurs dupliquées : la sous-requête est transformée en une jointure standard qui insère les résultats dans une table de travail, qui est ensuite triée afin de supprimer les lignes dupliquées.

Ordre de jointure et méthodes de mise à plat

Un facteur important dans le choix d'une méthode de mise à plat est le coût des ordres de jointure possibles. Par exemple, dans une jointure de t1, t2 et t3 :

```
select * from t1, t2  
where t1.c1 = t2.c1  
and t2.c2 in (select c3 from t3)
```

Si l'ordre de jointure le plus avantageux est t1, t2, t3 ou t2, t1, t3, une jointure standard ou une jointure de contrôle de présence est utilisée. Cependant, s'il est plus intéressant de réaliser la jointure avec t3 comme table externe, par exemple t3, t1, t2, une redéfinition d'index unique ou un tri pour élimination des valeurs dupliquées est utilisé.

La jointure mise à plat obtenue peut inclure des jointures par boucles imbriquées ou des jointures par fusion. En cas d'utilisation d'une jointure d'existence, les jointures par fusion ne sont réalisables qu'avant celle-ci, jamais après.

Sous-requêtes mises à plat exécutées comme jointures d'existence

Les sous-requêtes de prédicat quantifié peuvent être exécutées comme jointures normales lorsque leur jeu de résultats est un ensemble de valeurs uniques. Par exemple, s'il existe un index unique sur publishers.pub_id, cette requête portant sur une seule table renvoie à coup sûr un ensemble de valeurs uniques :

```
select title
from titles
where pub_id in (select pub_id
                  from publishers
                  where state = "TX")
```

Avec un index non unique sur publishers.city, cette requête peut aussi être exécutée en utilisant une jointure standard :

```
select au_lname
from authors a
where exists (select city
               from publishers p where p.city = a.city)
```

Bien que l'index sur publishers.city ne soit pas unique, la jointure peut toujours être mise à plat si l'index sert à filtrer les lignes dupliquées et à les supprimer de la requête.

Lorsqu'une sous-requête est mise à plat, les résultats de showplan affichent une jointure normale. En cas de filtrage, les résultats de showplan ne changent pas ; le seul message de diagnostic figure dans les résultats de dbcc traceon(310), où la *méthode* concernant la table indique « NESTED ITERATION with Tuple Filtering ».

Sous-requêtes mises à plat exécutées comme jointures d'existence

Toutes les requêtes in, any et exists vérifient l'existence de valeurs pertinentes et renvoient TRUE dès qu'une ligne correspondante est trouvée.

L'optimiseur convertit la sous-requête suivante en jointure d'existence :

```
select title
      from titles
     where title_id in
           (select title_id
              from titleauthor)
        and title like "A Tutorial%"
```

La requête de la jointure d'existence ressemble à la jointure normale ci-dessous, à la différence près que les résultats renvoyés ne sont pas les mêmes :

```
select title
      from titles T, titleauthor TA
     where T.title_id = TA.title_id
       and title like "A Tutorial%"
```

La base de données pubtune contient deux livres correspondant à la chaîne de recherche pour title. Chaque livre dispose d'entrées multiples dans titleauthor, car il a plusieurs auteurs. Une jointure normale renverrait cinq lignes, alors que la sous-requête ne renvoie que deux lignes, une par title_id, puisqu'elle arrête l'exécution de la jointure après la première ligne correspondante.

Lorsqu'il y a mise à plat de sous-requêtes pour utiliser des jointures de contrôle de présence, showplan affiche les résultats pour une jointure, avec le message « EXISTS TABLE: nested iteration » comme type de jointure pour la table de la sous-requête.

Sous-requêtes mises à plat exécutées avec une seule redéfinition d'index

Pour procéder à une seule redéfinition d'index, Adaptive Server :

- insère des lignes dans une table de travail puis trie cette dernière, en supprimant les lignes dupliquées et en créant un index clusterisé sur la clé de jointure ;
- joint la table de travail avec la table suivante dans l'ordre de jointure. S'il existe un index non unique sur publishers.pub_id, cette requête peut appliquer la stratégie de redéfinition d'index unique :

```
select title_id
      from titles
     where pub_id in
          (select pub_id from publishers where state = "TX")
```

Cette requête s'exécute ainsi :

```
select pub_id
      into #publishers
      from publishers
     where state = "TX"
```

Puis, après le tri, les valeurs dupliquées sont supprimées et l'index clusterisé est créé :

```
select title_id  
from titles, #publishers  
where titles.pub_id = #publishers.pub_id
```

Les messages de showplan pour une redéfinition d'index unique affichent « Worktable created for REFORMATTING » à l'étape 1 et « Using Clustered Index » sur la table de travail à l'étape 2.

dbcc traceon(310) affiche « REFORMATTING with Unique Reformatting » pour la méthode relative à la table publishers.

Sous-requêtes mises à plat avec élimination des valeurs dupliquées

Lorsqu'il est plus avantageux de placer les tables de la sous-requête à l'extérieur dans l'ordre de jointure, la requête est exécutée comme suit :

- réalisation d'une jointure standard avec la sous-requête transformée en requête externe, les résultats étant insérés dans une table de travail ;
- élimination des doubles par tri de la table de travail.

Par exemple, salesdetail contient des valeurs dupliquées pour title_id et est utilisé dans la sous-requête suivante :

```
select title_id, au_id, au_ord  
from titleauthor ta  
where title_id in (select ta.title_id  
                   from titles t, salesdetail sd  
                   where t.title_id = sd.title_id  
                   and ta.title_id = t.title_id  
                   and type = 'travel' and qty > 10)
```

Si l'ordre de jointure le plus approprié pour cette requête est salesdetail, titles, titleauthor, l'ordre de jointure optimal permet :

- la sélection de tous les résultats de requêtes et leur insertion dans une table de travail,
- la suppression des valeurs dupliquées de la table de travail et renvoi des résultats à l'utilisateur.

Messages de showplan pour des sous-requêtes mises à plat effectuant des tris.

Les résultats de showplan incluent deux phases pour les sous-requêtes qui utilisent des jointures normales plus un tri. La première étape affiche « Worktable1 created for DISTINCT » et la jointure mise à plat. La seconde phase affiche le tri et la sélection à partir de la table de travail.

dbcc traceon(310) imprime un message pour chaque permutation de jointure lorsqu'une ou plusieurs tables d'une sous-requête de prédictat quantifié figurent en premier dans l'ordre de jointure. Voici les messages obtenus lorsque l'ordre de jointure utilisé pour la requête ci-dessus est pris en considération :

2 - 0 - 1 -

This join order created while converting an exists join to a regular join, which can happen for subqueries, referential integrity, and select distinct.

Mise à plat des sous-requêtes d'expression

Les sous-requêtes d'expression sont des sous-requêtes qui sont comprises dans la liste de sélection d'une requête ou introduites par >, >=, <, <=, = ou !=.

Adaptive Server convertit ou met à plat des sous-requêtes en **équijointures** si :

- la sous-requête effectue des jointures sur des colonnes uniques ou renvoie des colonnes uniques, et
- il existe un index unique sur les colonnes.

Matérialisation des résultats de sous-requête

Dans certains cas, la sous-requête est traitée en deux étapes : les résultats de la requête interne sont *mérialisés* ou stockés dans une table de travail temporaire ou une variable interne avant l'exécution de la requête externe. Dans une première étape, la sous-requête est exécutée et les résultats de l'exécution sont enregistrés ; puis dans une deuxième étape ils sont utilisés. Adaptive Server matérialise les types de sous-requête suivants :

- sous-requêtes d'expression non corrélées
- sous-requêtes de prédictat quantifié contenant des agrégats avec la condition de corrélation dans la clause having.

Sous-requêtes d'expression non corrélées

Les sous-requêtes d'expression non corrélées doivent renvoyer une seule valeur. Lorsqu'une sous-requête n'est pas corrélée, elle renvoie la même valeur, quelle que soit la ligne en cours de traitement dans la requête externe. La requête s'exécute ainsi :

- exécution de la sous-requête, puis stockage des résultats dans une variable interne,
- remplacement de la valeur du résultat pour la sous-requête dans la requête externe.

La requête suivante contient une sous-requête d'expression non corrélée :

```
select title_id
  from titles
 where total_sales = (select max(total_sales)
                        from ts_temp)
```

Adaptive Server transforme la requête en :

```
select <internal_variable> = max(total_sales)
      from ts_temp
select title_id
  from titles
 where total_sales = <internal_variable>
```

Dans la seconde étape de cette transformation, la clause de recherche peut être optimisée. S'il existe un index sur `total_sales`, la requête peut l'utiliser. Le coût total d'une sous-requête d'expression matérialisée correspond à la somme des coûts des deux requêtes séparées.

Sous-requêtes de prédicat quantifié contenant des agrégats

Certaines sous-requêtes contenant des agrégats vectoriels (ou groupés) peuvent être matérialisées. C'est notamment le cas pour :

- sous-requêtes de prédicat quantifié non corrélées,
- sous-requêtes de prédicat quantifié, corrélées uniquement dans la clause `having`.

La matérialisation de la sous-requête comprend les deux étapes suivantes :

- Adaptive Server exécute d'abord la sous-requête, puis stocke les résultats dans une table de travail.
- Adaptive Server joint la table externe à la table de travail sous forme de jointure d'existence. Dans la plupart des cas, cette jointure ne peut pas être optimisée, car les statistiques pour la table de travail ne sont pas disponibles.

La matérialisation permet d'éviter le coût entraîné par l'évaluation des agrégats pour chaque ligne de la table. Par exemple, la requête :

```
select title_id  
      from titles  
     where total_sales in (select max(total_sales)  
                           from titles  
                           group by type)
```

est exécutée avec les étapes suivantes :

```
select maxsales = max(total_sales)  
      into #work  
      from titles  
      group by type  
select title_id  
      from titles, #work  
     where total_sales = maxsales
```

Le coût total d'exécution de sous-requêtes de prédicat quantifié correspond à la somme des coûts de requête pour les deux étapes.

Si des clauses `where` accompagnent une sous-requête, Adaptive Server exécute la sous-requête en dernier, pour éviter des exécutions superflues de cette dernière. Il arrive souvent que l'exécution de la sous-requête ne soit pas nécessaire parce que d'autres clauses moins longues ont déjà permis de déterminer si la ligne doit être ou non renvoyée :

- si l'une des clauses `and` est évaluée à la valeur FALSE, la ligne ne doit pas être renvoyée,
- si l'une des clauses `or` est évaluée à la valeur TRUE, la ligne doit être renvoyée.

Dans les deux cas, dès que l'évaluation d'une clause a permis de déterminer l'état de la ligne, il n'est plus nécessaire d'appliquer d'autres clauses à cette ligne. Ceci offre une amélioration des performances car les sous-requêtes longues sont exécutées moins souvent.

Sous-requête introduite par une clause **and**

Lorsque la clause **and** joint les clauses, l'évaluation est arrêtée dès qu'une des clauses a la valeur FALSE. La ligne est ignorée.

La requête suivante contient deux clauses **and** en plus de la sous-requête corrélée :

```
select au_fname, au_lname, title, royaltyper
      from titles t, authors a, titleauthor ta
     where t.title_id = ta.title_id
       and a.au_id = ta.au_id
       and advance >= (select avg(advance)
                           from titles t2
                          where t2.type = t.type)
       and price > $100
       and au_ord = 1
```

Adaptive Server ordonne les étapes de l'exécution afin d'évaluer la sous-requête en dernier, puis il évalue les conditions sur **price** et **au_ord**. Si une ligne ne remplit pas une condition **and**, Adaptive Server l'élimine sans vérifier les autres conditions **and** et passe à l'évaluation de la ligne suivante ; par conséquent, la sous-requête n'est pas traitée tant qu'une ligne ne remplit pas toutes les conditions **and**. Le nombre maximal de conditions AND dans une expression de requête est de 1 024.

Sous-requête introduite par une clause **or**

Si les conditions **where** d'une requête sont jointes par la clause **or**, l'évaluation s'interrompt lorsqu'une clause est évaluée à TRUE et la ligne est renvoyée.

La requête suivante contient deux clauses **or** en plus de la sous-requête :

```
select au_fname, au_lname, title
      from titles t, authors a, titleauthor ta
     where t.title_id = ta.title_id
       and a.au_id = ta.au_id
       and (advance > (select avg(advance)
                           from titles t2
                          where t2.type = t.type)
            or title = "Best laid plans"
            or price > $100)
```

Adaptive Server ordonne les conditions dans le plan de requête de façon à évaluer la sous-requête en dernier. Si une ligne remplit la condition de la clause or, Adaptive Server renvoie la ligne sans exécuter la sous-requête et passe à l'évaluation de la ligne suivante. Le nombre maximal de conditions OR dans une expression de requête est de 1 024.

Mise en mémoire cache des résultats de la sous-requête

Lorsque la mise à plat ou la matérialisation d'une sous-requête est impossible, Adaptive Server stocke les résultats de chaque évaluation de la sous-requête dans une mémoire cache interne. Pendant l'exécution de la requête, il analyse le nombre de fois où les résultats d'une sous-requête sont trouvés dans le cache. Il est alors question de **taux de présence dans le cache**. Un taux élevé de présence dans le cache signifie que la présence de celui-ci réduit le nombre de fois où la sous-requête est exécutée. Un taux faible de présence dans le cache signifie que le cache n'est pas pertinent et sa taille est réduite à mesure que la requête s'exécute.

La mise en mémoire cache des résultats de la sous-requête améliore les performances lorsque les colonnes de jointure ou les colonnes de corrélation contiennent des valeurs en double. Elle est encore plus efficace lorsque les valeurs sont ordonnées, comme dans une requête utilisant un index. Lorsque aucune valeur de corrélation n'est en double, la mise en mémoire cache ne permet pas de gain de performances.

Affichage des informations sur le cache de sous-requête

La commande `set statistics subquerycache on` affiche pour chaque sous-requête le nombre de présences (hits) et de non-présences (misses) dans le cache, ainsi que le nombre de lignes. L'exemple suivant illustre cette commande :

```
set statistics subquerycache on

select type, title_id
from titles
where price > all
  (select price
   from titles
   where advance < 15000)
Statement: 1 Subquery: 1 cache size: 75 hits: 4925
misses: 75
```

Si l'instruction comporte des sous-requêtes de part et d'autre d'une union, celles-ci sont numérotées en séquence.

Optimisation des sous-requêtes

Lorsque des requêtes contenant des sous-requêtes ne sont ni mises à plat ni matérialisées :

- la requête externe et chaque sous-requête non mise à plat sont optimisées individuellement,
- les sous-requêtes les plus internes (les plus imbriquées) sont optimisées en premier,
- l'estimation de l'utilisation du cache de buffer pour chaque sous-requête est diffusée à l'extérieur, pour faciliter l'évaluation du coût d'E/S et de la stratégie des requêtes externes.

Dans un grand nombre de requêtes contenant des sous-requêtes, une sous-requête est « rattachée » à l'une des tables externes balayées selon un processus en deux étapes. Premièrement, l'optimiseur recherche le point, dans l'ordre de jointure, où toutes les colonnes de corrélation sont disponibles. Ensuite, il cherche à partir de ce point l'accès à la table contenant le plus petit nombre de lignes qualifiées et « lie » la sous-requête à cette table. La sous-requête est finalement exécutée pour chaque ligne qualifiée de la table avec laquelle elle est imbriquée.

Clauses or et unions dans les jointures

Adaptive Server ne peut pas optimiser les clauses de jointure reliées par or et peut exécuter des produits cartésiens pour traiter la requête.

Remarque Adaptive Server effectue l'optimisation des arguments de recherche reliés par or. Cette description concerne uniquement les clauses de jointure.

Par exemple, lorsque Adaptive Server traite la requête suivante, il doit consulter toutes les lignes de l'une des tables pour chaque ligne de l'autre table :

```
select *
  from tab1, tab2
 where tab1.a = tab2.b
   or tab1.x = tab2.y
```

Si vous utilisez union, chaque partie de l'union est optimisée séparément :

```
select *
  from tab1, tab2
 where tab1.a = tab2.b
union all
select *
  from tab1, tab2
 where tab1.x = tab2.y
```

Vous pouvez utiliser union à la place de union all pour éliminer les lignes dupliquées, mais dans ce cas ces lignes seront toutes supprimées. Il peut s'avérer impossible d'obtenir exactement le même nombre de lignes dupliquées à partir de la requête réécrite.

Adaptive Server peut optimiser des sélections comportant des jointures effectuées à l'aide de l'opérateur union. L'utilisation de l'opérateur or donne des résultats semblables à ceux d'union, sauf en ce qui concerne le traitement des lignes dupliquées et des tables vides :

- union supprime toutes les lignes dupliquées (dans une phase de tri) ; union all ne supprime aucune entrée dupliquée. Une requête équivalente utilisant or peut renvoyer des entrées dupliquées ;
- une jointure comportant une table vide ne renvoie aucune ligne.

Traitement des requêtes parallèles

Le présent chapitre présente les concepts fondamentaux et la terminologie concernant l'optimisation des requêtes parallèles, le tri parallèle et aborde d'autres aspects des requêtes parallèles. Vous y trouverez également une présentation des commandes permettant de traiter ces requêtes.

Sujet	Page
Types de requête pouvant bénéficier du traitement parallèle	156
Modèle de processus de travail d'Adaptive Server	158
Types d'accès parallèle aux données	162
Contrôle du degré de parallélisation	167
Commandes associées aux tables partitionnées	175
Equilibre ressources/performances	177
Recommandations pour la configuration des requêtes parallèles	179
Incidence au niveau du système	185
Cas dans lesquels les résultats des requêtes parallèles peuvent être différents	186

D'autres chapitres abordent certains aspects du traitement parallèle de façon plus approfondie :

- Pour plus de précisions sur la façon dont l'optimiseur d'Adaptive Server détermine l'opportunité et le coût d'une exécution en parallèle, reportez-vous au [Chapitre 8, « Optimisation des requêtes parallèles »](#).
- Pour davantage de précisions sur les concepts liés au tri parallèle, reportez-vous au [Chapitre 9, « Tri parallèle »](#).
- Pour plus d'informations sur le placement d'objets pour les performances parallèles, reportez-vous à la section [« Partitionnement des tables pour les performances », page 107](#) du manuel *Performances et optimisation : concepts de base*.
- Pour plus d'informations sur le verrouillage au cours du traitement des requêtes parallèles, reportez-vous au *Guide d'administration système*.

- Pour plus d'informations sur les messages showplan, reportez-vous à la section « [Messages showplan pour les requêtes parallèles](#) », page 119 du guide *Performances et optimisation : contrôle et analyse*.
- Pour obtenir des explications sur la manière dont Adaptive Server utilise plusieurs moteurs, reportez-vous au [Chapitre 4, « Utilisation des moteurs et des CPU »](#) du manuel *Performances et optimisation : contrôle et analyse*.

Types de requête pouvant bénéficier du traitement parallèle

Lorsque Adaptive Server est configuré pour le traitement parallèle des requêtes, l'optimiseur évalue chaque requête pour déterminer si elle est adaptée à une exécution en parallèle. Le cas échéant, et si l'optimiseur détermine qu'un plan d'exécution en parallèle permet d'obtenir des résultats plus rapides qu'un plan en série, la requête est divisée en composants traités simultanément.

Les résultats sont combinés et livrés au client en un temps inférieur à celui qui serait nécessaire pour traiter la requête en série, sous la forme d'un composant unique.

Le traitement parallèle des requêtes peut améliorer les performances des types de requête ci-après :

- select permettant d'analyser un grand nombre de pages mais retournant un nombre de lignes limité, notamment :
 - balayage de tables ou d'index clusterisés avec des agrégats groupés ou non groupés,
 - balayage de tables ou d'index clusterisés permettant d'analyser un grand nombre de pages, mais contenant des clauses where qui retournent un faible pourcentage de lignes.

- instructions `select` comportant une clause `union`, `order by` ou `distinct`, car ces requêtes permettent de remplir des tables de travail en parallèle et d'utiliser le tri parallèle,
- instructions `select` qui utilisent des jointures par fusion, en vue d'analyser les tables et d'effectuer les opérations de tri et de fusion,
- instructions `select` dans lesquelles l'optimiseur choisit la stratégie de redéfinition d'index, car elles peuvent remplir des tables de travail en parallèle et utiliser le tri parallèle,
- instructions `create index` et clauses `alter table...add constraint` qui créent des index unique et primary key,
- commande `dbcc checkstorage`.

Les requêtes de jointure peuvent faire appel au traitement en parallèle sur une ou plusieurs tables.

Les commandes retournant de vastes jeux de résultats non triés ont peu de chances de tirer profit du traitement parallèle en raison des contraintes du réseau. Dans la plupart des cas, les résultats sont retournés par la base de données plus rapidement qu'ils ne peuvent être fusionnés et retournés au client via le réseau.

Les commandes modifiant les données (`insert`, `update` et `delete`) et les curseurs ne s'exécutent pas en parallèle. Les blocs internes et imbriqués de requêtes contenant des sous-requêtes ne sont jamais exécutés en parallèle mais le bloc externe peut l'être.

Les requêtes des systèmes d'aide à la décision (DSS) permettant d'accéder à des tables volumineuses et retournant des informations de synthèse sont celles qui tirent le meilleur parti du traitement parallèle. En effet, l'overhead lié à l'allocation et à la gestion des requêtes parallèles rend l'exécution parallèle moins efficace pour les requêtes transactionnelles (OLTP), qui accèdent généralement à moins de lignes et joignent moins de tables. Lorsqu'un serveur est configuré pour le traitement parallèle, seules les requêtes qui doivent accéder à 20 pages au moins sont prises en considération ; par conséquent, la plupart des requêtes transactionnelles OLTP s'exécutent en série.

Modèle de processus de travail d'Adaptive Server

Adaptive Server utilise un **processus de coordination** et plusieurs **processus de travail** pour exécuter les requêtes en parallèle. Une requête qui s'exécute en parallèle avec huit processus de travail est comparable à huit requêtes en série accédant chacune à un huitième de la table, tandis que le processus de coordination supervise l'interaction et gère le retour des résultats au client.

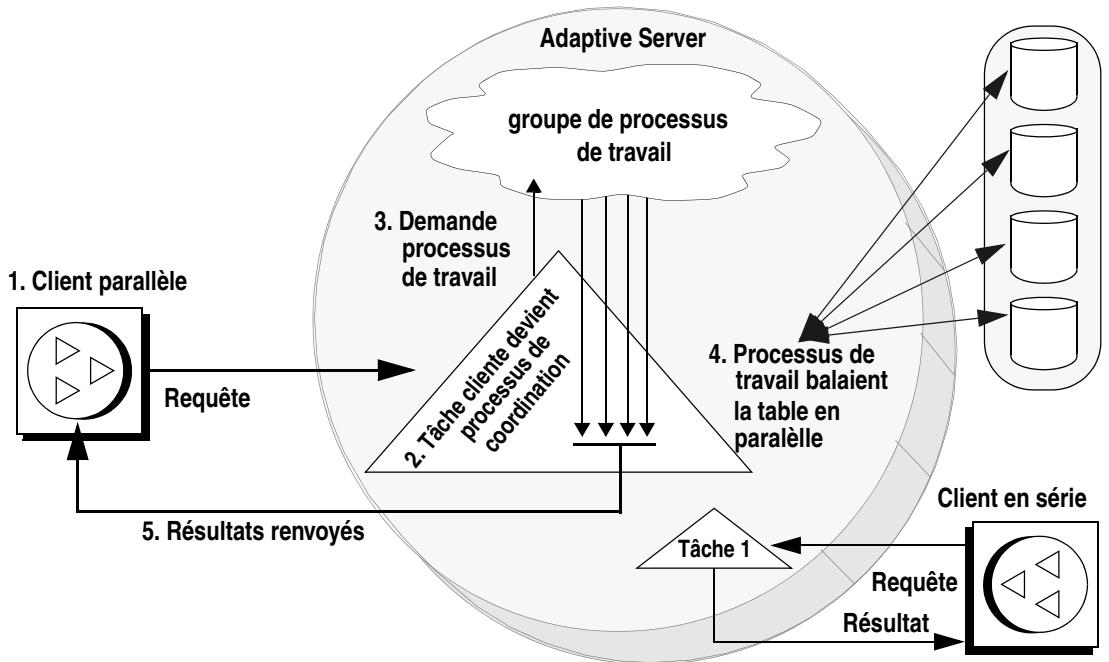
Chaque processus de travail utilise à peu près le même espace mémoire qu'une connexion utilisateur. Le processus de travail s'exécute comme une tâche programmée sur un moteur : il balaie les pages de données, place les E/S disque dans des files d'attente et se déroule comme n'importe quelle autre tâche sur le serveur. L'une des principales différences tient à ce que, au cours de la dernière phase du traitement des requêtes, le processus de coordination gère la fusion des résultats et leur retour au client, après coordination avec les processus de travail.

La [figure 7-1](#) montre les événements intervenant dans le traitement des requêtes parallèles :

- 1 Le client soumet une requête.
- 2 La tâche du client affectée à l'exécution de la requête devient le processus de coordination pour l'exécution parallèle des requêtes.
- 3 Le processus de coordination demande quatre processus de travail au groupe des processus de travail. Le processus de coordination et les processus de travail forment, ensemble, une **famille**.
- 4 Les processus de travail exécutent la requête en parallèle.
- 5 Le processus de coordination renvoie au client les résultats générés par tous les processus de travail.

Le client en série représenté dans l'angle inférieur droit de la [figure 7-1](#) soumet une requête qui est traitée en série.

Figure 7-1 : Modèle de processus de travail



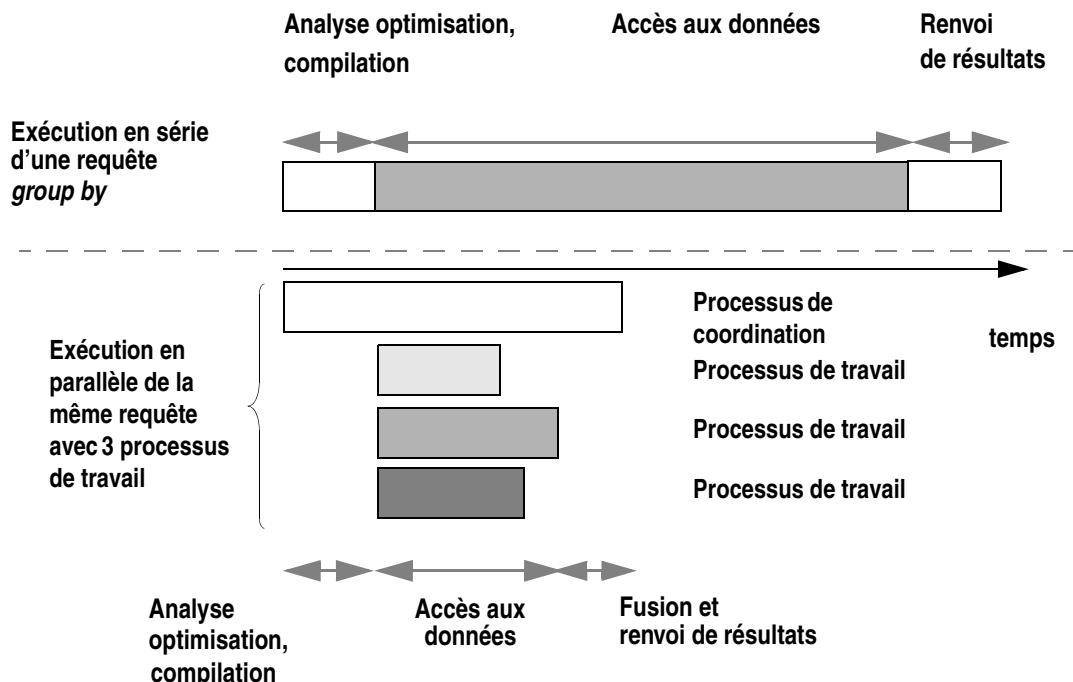
Au cours du traitement de la requête, les tâches sont repérées, dans les tables système, par un ID de famille (fid). Tous les processus de travail d'une même famille possèdent le même identifiant de famille mais chacun porte un ID de processus serveur qui lui est propre (spid). Les procédures système telles que `sp_who` et `sp_lock` affichent à la fois le fid et le spid pour les requêtes parallèles, ce qui vous permet d'observer le comportement de tous les processus d'une famille.

Exécution de requêtes parallèles

La [figure 7-2](#) montre comment le traitement d'une requête parallèle réduit le temps de réponse par rapport à la même requête exécutée en série.

Dans l'exécution parallèle, trois processus de travail balaiennent les pages de données. Le délai requis par chaque processus peut varier en fonction de la quantité de données auxquelles l'accès est nécessaire. Le balayage peut être temporairement bloqué du fait de verrous sur les pages de données détenus par d'autres utilisateurs. Lorsque toutes les données ont été lues, les résultats des différents processus de travail sont fusionnés pour former un jeu de résultats unique, établi par le processus de coordination et renvoyé au client.

Figure 7-2 : Délais relatifs d'exécution de requêtes en série et en parallèle



Le volume total des tâches effectuées par la requête exécutée en parallèle est supérieur à celui des tâches effectuées en série mais le temps de réponse est plus court.

Retour des résultats des requêtes parallèles

Les résultats des requêtes parallèles sont renvoyés au moyen de l'une des trois stratégies de fusion ou dans l'étape finale d'un tri. Les requêtes parallèles qui ne comportent pas d'étape finale de tri font appel à l'un des types de fusion ci-après :

- Les requêtes qui contiennent un agrégat vectoriel (groupé) utilisent des tables de travail pour le stockage temporaire des résultats ; le processus de coordination fusionne ces résultats dans une table et les renvoie au client.
- Les requêtes contenant un agrégat scalaire (non groupé) font appel à des variables internes et le processus de coordination effectue les calculs finals pour retourner les résultats au client.
- Les requêtes qui ne contiennent pas d'agrégat et qui n'utilisent pas de clauses ne nécessitant pas un tri final peuvent renvoyer les résultats au client à mesure du balayage des tables. Chaque processus de travail stocke les résultats dans un buffer de résultats et utilise des verrous d'adresse pour coordonner le transfert des résultats aux buffers du réseau pour la tâche considérée.

Plusieurs types de fusion peuvent être utilisés lorsque les requêtes nécessitent plusieurs étapes ou tables de travail.

Reportez-vous à la section « [Messages showplan pour les requêtes parallèles](#) », [page 119](#) du manuel *Performances et optimisation : contrôle et analyse* pour plus d'informations sur les messages de fusion.

Dans le cas de requêtes parallèles comportant une clause `order by`, `distinct` ou `union`, les résultats sont stockés dans une table de travail dans `tempdb`, puis triés. Si le tri peut être effectué en parallèle, ce type de tri est utilisé et les résultats sont renvoyés au client au cours de l'étape finale de fusion effectuée par le tri.

Pour plus d'informations sur le déroulement du tri parallèle, reportez-vous au Chapitre 9, « [Tri parallèle](#) ».

Remarque Comme les requêtes parallèles utilisent plusieurs processus pour balayer les pages de données, celles qui ne font pas appel aux agrégats et qui n'incluent pas de tri final peuvent renvoyer les résultats dans un ordre différent des requêtes en série. Les résultats renvoyés peuvent être différents lorsque les requêtes contiennent l'option `set rowcount activée` ou lorsqu'elles opèrent des sélections dans une variable locale.

Pour plus de détails et des solutions, reportez-vous à la section « [Cas dans lesquels les résultats des requêtes parallèles peuvent être différents](#) », [page 186](#).

Types d'accès parallèle aux données

Adaptive Server accède aux données en parallèle de différentes manières, selon les paramètres de configuration, le partitionnement des tables et la disponibilité des index. L'optimiseur peut choisir un mélange de méthodes en parallèle et en série dans les requêtes faisant appel à plusieurs tables et comportant plusieurs étapes. Les méthodes en parallèle sont les suivantes :

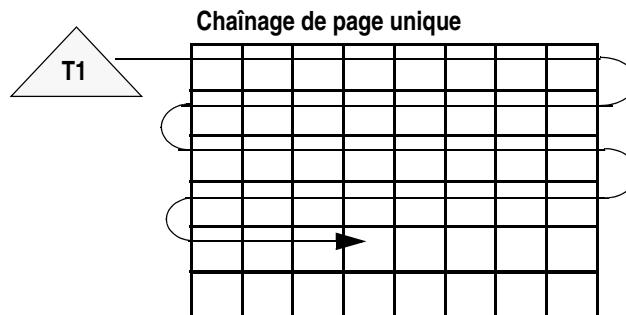
- balayage de table reposant sur un hachage ;
- balayages d'index non clusterisé, reposant sur le hachage ;
- balayage reposant sur les partitions, soit un balayage de table complète ; soit un balayage positionné avec un index clusterisé ;
- balayage basé sur des intervalles durant des jointures par fusion.

Les sections suivantes décrivent quelques unes de ces méthodes.

Pour obtenir d'autres exemples, reportez-vous au [Chapitre 8, « Optimisation des requêtes parallèles »](#).

La [figure 7-3](#) représente un balayage portant sur une table verrouillée au niveau de toutes les pages (table APL), exécuté en série par une même tâche. La tâche suit le chaînage des pages de la table pour lire chaque page, en s'arrêtant pour effectuer des opérations d'E/S physiques lorsque les pages nécessaires ne se trouvent pas dans le cache.

Figure 7-3 : Une tâche en série balaie les pages de données

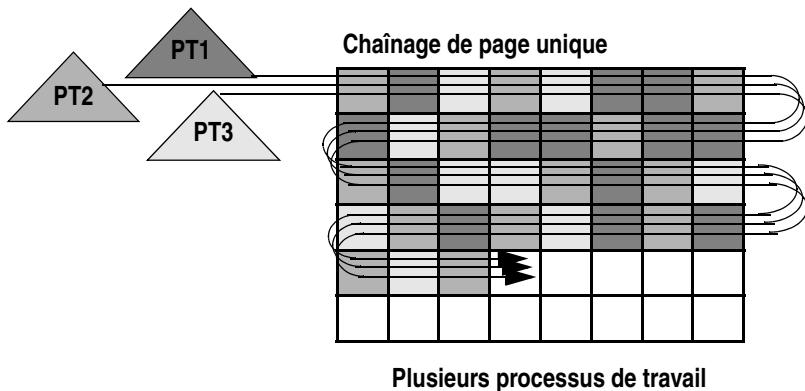


Balayage de table reposant sur un hachage

La [figure 7-4](#) montre comment trois processus divisent le travail d'accès aux pages de données d'une table APL au cours d'un balayage de table reposant sur le hachage. Dans ce type de balayage, chaque processus de travail effectue une E/S logique sur chaque page mais examine les lignes sur un tiers des pages seulement, comme l'indique l'ombrage différent selon les pages. Les balayages de tables basés sur le hachage sont utilisés uniquement pour la requête externe dans une jointure.

Avec un seul moteur, la requête continue de bénéficier de l'accès parallèle puisqu'un processus de travail peut s'exécuter tandis que les autres attendent des E/S. S'il y a plusieurs moteurs, certains processus de travail peuvent être exécutés simultanément.

Figure 7-4 : Les processus de travail balaiennent une table non partitionnée

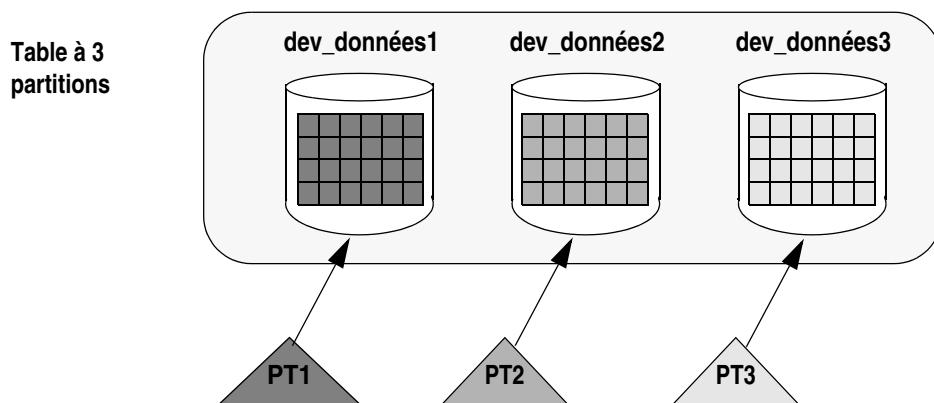


Les balayages de tables reposant sur le hachage augmentent le nombre d'E/S logiques nécessaires car chaque processus de travail doit accéder à chaque page pour appliquer un hachage à l'ID de page. Pour les tables verrouillées au niveau des pages de données seulement (tables DOL), ce type de balayage procède à un hachage au niveau de l'ID d'extent ou de l'ID de page d'allocation, de sorte qu'un seul processus de travail balaie une page et le nombre des E/S logiques n'augmente pas.

Balayage reposant sur les partitions

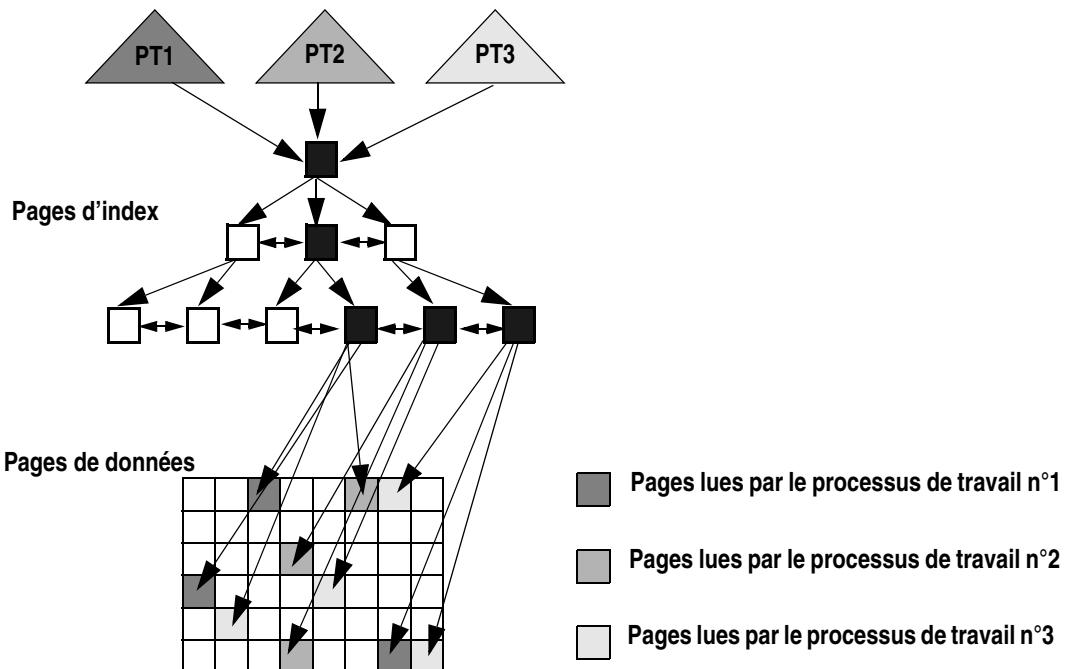
La [figure 7-5](#) montre comment une requête balaie une table disposant de trois partitions sur trois disques physiques. Lorsqu'un seul moteur est utilisé, cette requête bénéficie du traitement parallèle parce qu'un processus de travail s'exécute pendant que les autres sont mis en veille en attendant des E/S ou la libération de verrous détenus par d'autres processus. Si plusieurs moteurs sont disponibles, les processus de travail peuvent être exécutés simultanément. Cette configuration permet d'atteindre des performances élevées grâce à la parallélisation des E/S.

Figure 7-5 : Plusieurs processus de travail accèdent à plusieurs partitions



Balayage d'index reposant sur un hachage

La [figure 7-6](#) représente un balayage d'index reposant sur le hachage. Les balayages d'index basés sur le hachage peuvent être effectués avec des index non clusterisés ou des index clusterisés sur des tables DOL. Chaque processus de travail parcourt les niveaux supérieurs de l'index et lit les pages de niveau feuille de l'index. Chacun procède alors au hachage, soit en fonction de l'ID de la page de données, soit en fonction de la valeur clé, pour déterminer quelles sont les pages de données ou les lignes de données à traiter. La lecture de chaque page feuille entraîne un overhead négligeable.

Figure 7-6 : Balayage d'index non clusterisé, reposant sur le hachage

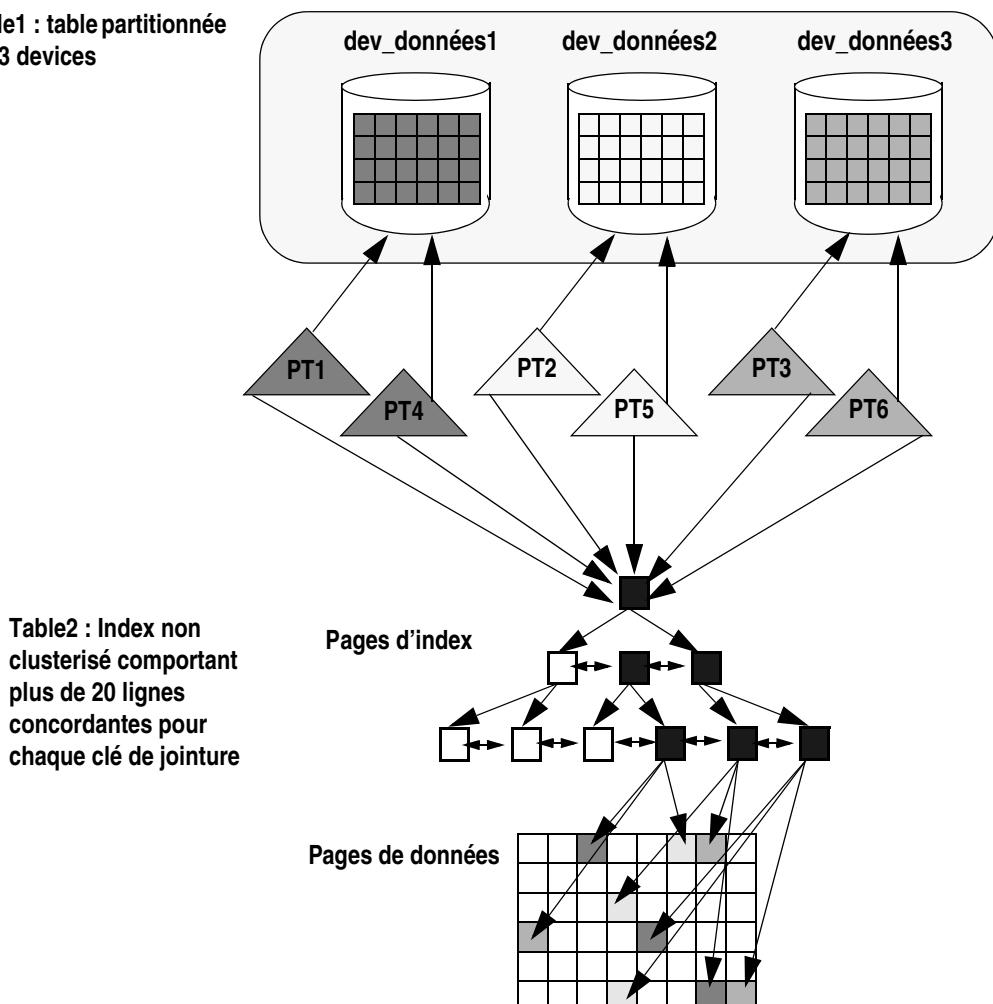
Traitement parallèle de deux tables dans le cadre d'une jointure

La figure 7-7 représente une requête de jointure par boucles imbriquées effectuant un balayage reposant sur les partitions dans une table comportant trois partitions et un balayage d'index reposant sur le hachage, avec deux processus de travail sur la seconde table. Lorsque des méthodes d'accès parallèle sont utilisées sur plusieurs tables au cours d'une jointure à boucle imbriquée, le nombre total de processus de travail requis est le produit du processus de travail de chaque balayage. Dans ce cas, six processus de travail effectuent la requête, chaque processus balayant les deux tables. Deux processus de travail balaiennent chaque partition de la première table et les six processus naviguent dans l'arbre de l'index pour trouver la seconde table dont ils balaiennent les pages de niveau feuille. Chaque processus de travail accède aux pages de données correspondant à sa valeur de hachage.

L'optimiseur choisit un plan d'exécution parallèle pour une table uniquement lorsqu'un balayage renvoie 20 pages ou plus. Ces types de requêtes de jointure nécessitent 20 correspondances ou plus sur la clé de jointure de la table interne pour que le balayage interne puisse être optimisé en parallèle.

Figure 7-7 : Requête de jointure utilisant différentes méthodes d'accès parallèle sur chaque table

Table1 : table partitionnée sur 3 devices



Messages *showplan*

showplan affiche le degré de parallélisation chaque fois qu'une table fait l'objet d'un accès parallèle. L'exemple ci-après montre les messages correspondant à chacune des tables de la jointure dans la [figure 7-7](#) :

```
Executed in parallel with a 2-way hash scan.  
Executed in parallel with a 3-way partition scan.
```

showplan affiche également un message indiquant le nombre total de processus de travail utilisés. Pour la requête représentée dans la [figure 7-7](#), il signale :

```
Executed in parallel by coordinating process and 6  
worker processes.
```

Pour obtenir des exemples supplémentaires, reportez-vous au [Chapitre 8, « Optimisation des requêtes parallèles »](#).

Reportez-vous à la section [« Messages *showplan* pour les requêtes parallèles »](#), [page 119](#) du manuel *Performances et optimisation : contrôle et analyse* pour de plus amples informations.

Contrôle du degré de parallélisation

Le **degré de parallélisation** d'une requête parallèle est le nombre de processus de travail utilisés pour l'exécuter. Ce nombre dépend de plusieurs facteurs, notamment :

- les valeurs des paramètres de configuration parallèle ou les limites au niveau de la session,
(reportez-vous au [tableau 7-1](#) et au [tableau 7-2](#))
- le nombre de partitions sur une table (pour les balayages reposant sur les partitions),
- le niveau de parallélisation proposé par l'optimiseur,
- le nombre de processus de travail disponibles au moment de l'exécution de la requête.

Il est possible de fixer des limites en ce qui concerne le degré de parallélisation :

- A l'échelle du serveur : en utilisant la procédure système `sp_configure` avec les paramètres présentés dans le [tableau 7-1](#). Seul l'administrateur système peut utiliser `sp_configure`.
- Pour une session : en utilisant la commande `set` avec les paramètres présentés dans le [tableau 7-2](#). Tous les utilisateurs peuvent exécuter `set` ; il est également possible de l'inclure dans des procédures stockées.
- Dans une requête `select` : en utilisant la clause `parallel`, comme indiqué dans la section « [Contrôle de la parallélisation pour une requête](#) », [page 173](#).

Paramètres de configuration pour le contrôle de la parallélisation

Les paramètres de configuration permettant de contrôler le degré de parallélisation à l'échelle du serveur sont repris dans le [tableau 7-1](#).

Tableau 7-1 : Configuration des paramètres pour une exécution en parallèle

Paramètre	Explication	Commentaire
number of worker processes	Nombre maximal de processus de travail disponibles pour toutes les requêtes parallèles. Chaque processus de travail requiert approximativement autant de mémoire qu'une connexion utilisateur.	Redémarrage du serveur requis
max parallel degree	Nombre de processus de travail susceptibles d'être utilisés par une seule et même requête. Ce nombre doit être inférieur ou égal à <code>number of worker processes</code> et supérieur ou égal à <code>max scan parallel degree</code> .	Dynamique, sans redémarrage
max scan parallel degree	Nombre maximal de processus de travail susceptibles d'être utilisés pour un balayage reposant sur le hachage. Il doit être inférieur ou égal à <code>number of worker processes</code> et à <code>max parallel degree</code> .	Dynamique, sans redémarrage

La configuration de `number of worker processes` influe sur la taille du cache de données et du cache de procédures, de sorte qu'il peut être nécessaire de modifier également la valeur de `total memory`.

Pour de plus amples informations, reportez-vous au *Guide d'administration système*.

Lorsque vous modifiez max parallel degree ou max scan parallel degree, tous les plans d'exécution de requête se trouvant dans le cache sont invalidés. Lors de l'exécution suivante d'une procédure stockée ou d'un trigger, le plan est recompilé et les nouvelles valeurs utilisées.

Application des limites aux plans d'exécution de requête

Lorsque des requêtes sont optimisées, les paramètres de configuration influent sur les plans d'exécution.

- max parallel degree limite :
 - le nombre de processus de travail pour un balayage reposant sur les partitions,
 - le nombre total des processus de travail applicables aux requêtes de jointure à boucle imbriquée dans lesquelles des méthodes d'accès parallèle sont utilisées sur plusieurs tables,
 - le nombre de processus de travail utilisés pour les opérations de tri et de fusion dans les jointures par fusion,
 - le nombre de processus de travail susceptibles d'être utilisés par les opérations de tri parallèle.
- max scan parallel degree limite le nombre de processus de travail pour les balayages de tables basés sur le hachage et les balayages d'index.

Fonctionnement combiné des limites

Vous pouvez affecter à number of worker processes la valeur 50 pour permettre l'exécution de plusieurs requêtes parallèles en même temps. Si la table possédant le plus grand nombre de partitions possède 10 partitions, vous pouvez fixer à 10 le paramètre max parallel degree, limitant ainsi toutes les requêtes select à un maximum de 10 processus de travail. Comme le balayage reposant sur le hachage fonctionne mieux avec 2 ou 3 processus de travail, max scan parallel degree peut être défini sur 3.

Dans le cas d'une requête liée à une table unique ou d'une jointure impliquant un accès série à d'autres tables, ces valeurs autorisent un certain nombre de possibilités de traitement en parallèle :

- balayages de partitions parallèles dans toute table comportant entre 2 et 10 partitions,
- balayages de tables reposant sur le hachage avec un maximum de 3 processus de travail,
- balayages d'index non clusterisés reposant sur le hachage dans des tables comportant des index non clusterisés avec un maximum de 3 processus de travail.

Pour les jointures à boucle imbriquée où des méthodes d'accès parallèle sont utilisées sur plusieurs tables, certains choix de traitement parallèle sont possibles :

- Jointures utilisant un balayage basé sur le hachage sur une table et balayages basés sur les partitions dans des tables comportant 2 ou 3 partitions.
- Jointures utilisant des balayages basés sur les partitions dans les deux tables. Par exemple :
 - un degré de parallélisation de 3 pour une table partitionnée multiplié par une valeur max scan parallel degree de 3 pour un balayage reposant sur le hachage nécessite 9 processus de travail ;
 - une table comportant 2 partitions et une table en comportant 5 nécessitent 10 processus de travail pour des balayages basés sur les partitions dans les deux tables ;
 - Les tables comportant entre 4 et 10 partitions peuvent être impliquées dans une jointure, l'accès à une ou plusieurs tables s'effectuant en série.

Pour les jointures par fusion :

- Pour une jointure par fusion complète, 10 processus de travail balaiennent les tables (à moins que celles-ci ne contiennent moins de 10 valeurs distinctes sur les clés de jointure) ; le nombre de partitions sur les tables n'est pas pris en compte.
- Pour une jointure par fusion qui balaie une table et sélectionne des lignes pour les copier dans une table de travail :
 - La lecture effectuée avant la jointure par fusion peut être en série ou en parallèle. Pour ce type de requête, le degré de parallélisation est déterminé normalement.
 - Pour la fusion, 10 processus de travail sont utilisés, sauf s'il y a moins de valeurs distinctes dans la clé de jointure.
 - Pour le tri, un maximum de 10 processus de travail peuvent être utilisés.

Pour des performances plus élevées, lors de la création d'un index clusterisé dans une table comportant 10 partitions, la valeur 50 affectée au paramètre `number of worker processes` vous permet de fixer `max parallel degree` à 20 pour la commande `create index`.

Pour plus d'informations sur la configuration des processus de travail pour le tri, reportez-vous à la section « [Processus de travail requis pour le tri parallèle](#) », page 240.

Exemples de paramètres de configuration parallèle

La commande ci-après définit le paramètre `number of worker processes` :

```
sp_configure "number of worker processes", 50
```

Après le redémarrage du serveur, ces commandes définissent les autres paramètres de configuration :

```
sp_configure "max parallel degree", 10  
sp_configure "max scan parallel degree", 3
```

Pour afficher les valeurs en cours de ces paramètres, utilisez la commande :

```
sp_configure "Parallel Query"
```

Utilisation des options set pour contrôler la parallélisation d'une session

Deux options **set** vous permettent de restreindre le degré de parallélisation dans le cadre d'une session ou dans des procédures stockées ou des triggers. Ces options sont utiles pour tester des requêtes parallèles et pour limiter les requêtes non essentielles à une exécution en série, afin que les processus de travail restent disponibles pour d'autres tâches. Les options **set** sont récapitulées dans le [tableau 7-2](#).

Tableau 7-2 : Définition d'options pour optimiser l'exécution en parallèle

Paramètre	Fonction
parallel_degree	Définit le nombre maximal de processus de travail pour une requête dans une session, une procédure stockée ou un trigger. Ce nombre remplace le paramètre de configuration max parallel degree mais il doit être inférieur ou égal à la valeur de max parallel degree.
scan_parallel_degree	Définit le nombre maximal de processus de travail pour un balayage reposant sur le hachage pendant une session, dans une procédure stockée ou dans un trigger spécifiques. Ce nombre remplace le paramètre de configuration max scan parallel degree mais il doit être inférieur ou égal à la valeur de max scan parallel degree.

Si vous spécifiez une valeur trop grande pour l'une des options **set**, la valeur du paramètre de configuration correspondant est utilisée et un message indique la valeur effective. Aussi longtemps que **set parallel_degree** ou **set scan_parallel_degree** est actif au cours d'une session, les plans d'exécution associés à toute procédure stockée que vous exécutez ne sont pas placés dans le cache des procédures. Les procédures exécutées alors que ces options sont actives peuvent produire des plans d'exécution moins efficaces.

Exemples de commandes **set**

L'exemple ci-après restreint toutes les requêtes lancées dans la session en cours à 5 processus de travail :

```
set parallel_degree 5
```

Aussi longtemps que cette commande est effective, aucune requête portant sur une table ayant plus de 5 partitions ne peut utiliser un balayage basé sur des partitions.

Pour annuler la limite fixée pour la session, utilisez :

```
set parallel_degree 0
or
set scan_parallel_degree 0
```

Pour exécuter les requêtes suivantes en mode série, utilisez :

```
set parallel_degree 1
or
set scan_parallel_degree 1
```

Contrôle de la parallélisation pour une requête

L'extension parallel associée à la clause from d'une commande select permet aux utilisateurs de suggérer le nombre de processus de travail utilisés dans une instruction select. Le degré de parallélisation que vous spécifiez ne peut pas être supérieur à la valeur définie à l'aide de sp_configure ou à la limite de la session contrôlée par une commande set. Si vous indiquez une valeur supérieure, cette spécification est ignorée et l'optimiseur utilise la limite set ou sp_configure.

La syntaxe de l'instruction select est la suivante :

```
select ...
from nom_table [( [index nom_index]
    [parallel [degré_parallélisation | 1 ]]
    [prefetch taille] [lru|mru] ) ]
nom_table [( [index nom_index]
    [parallel [degré_parallélisation | 1 ]]
    [prefetch taille] [lru|mru])] ...
```

Exemples de clauses *parallel* au niveau requête

Pour spécifier le degré de parallélisation pour une seule requête, insérez parallel après le nom de la table. Dans cet exemple, la requête s'exécute en mode série :

```
select * from huge_table (parallel 1)
```

L'exemple ci-après spécifie l'index à utiliser dans la requête et fixe le degré de parallélisation à 2 :

```
select * from huge_table (index ncix parallel 2)
```

Pour de plus amples informations, reportez-vous à la section « [Suggestion d'un degré de parallélisation pour une requête](#) », page 60.

Disponibilité des processus de travail et exécution des requêtes

Au moment de l'exécution, si le nombre de processus de travail défini dans le plan d'exécution de requête n'est pas disponible, Adaptive Server ajuste le plan pour que la requête soit exécutée en utilisant un nombre inférieur de processus de travail. Ce type de modification est appelé **ajustement lors de l'exécution** et peut entraîner une exécution en série de la requête.

Un ajustement lors de l'exécution survenant de temps à autre indique probablement un goulet d'étranglement occasionnel et momentané.

En revanche, des ajustements trop fréquents lors de l'exécution peuvent indiquer que le nombre de processus de travail configuré est insuffisant pour la charge.

Pour de plus amples informations, reportez-vous à la section « [Ajustements lors de l'exécution des processus de travail](#) », page 216.

Vous pouvez également utiliser l'option `set process_limit_action` pour vérifier si une requête ou une procédure stockée doit utiliser automatiquement un plan ajusté, si elle doit en avertir l'utilisateur ou si la commande doit échouer lorsqu'elle ne peut pas utiliser le nombre optimal de processus de travail.

Pour de plus amples informations, reportez-vous à la section « [Utilisation de set process_limit_action](#) », page 226.

Les ajustements lors de l'exécution sont transparents pour l'utilisateur final sauf dans les cas suivants :

- une requête s'exécutant habituellement en parallèle peut être considérablement ralenti lorsqu'elle est exécutée en série ;
- si `set process_limit_action` est actif, un avertissement peut être émis ou la requête peut se trouver interrompue, en fonction de la configuration de l'option.

Autres paramètres de configuration pour le traitement en parallèle

Deux autres paramètres de configuration sont utilisables pour le traitement des requêtes en parallèle :

- `number of sort buffers` configure le nombre maximal de buffers que les opérations de tri parallèle peuvent utiliser à partir du cache de données.

Reportez-vous à la section « [Caches, buffers de tri et tris parallèles](#) », page 245.

- `memory per worker process` établit une zone mémoire que tous les processus de travail utilisent pour les messages au cours du traitement de la requête. La valeur par défaut, 1024 octets par processus de travail, constitue un espace suffisant dans la plupart des cas, de sorte qu'il est généralement inutile de redéfinir cette valeur.

Reportez-vous à la section « [Gestion des processus de travail](#) », page 231 du manuel *Performances et optimisation : contrôle et analyse* pour plus d'informations sur le contrôle et l'optimisation de cette valeur.

Commandes associées aux tables partitionnées

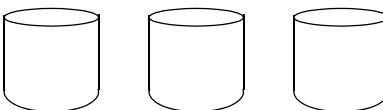
Vous pouvez trouver la procédure détaillée permettant de partitionner des tables, de les placer sur des devices spécifiques et de charger des données avec une copie parallèle « bulkcopy » au [Chapitre 6, « Gestion de l'emplacement physique des données »](#) du manuel *Performances et optimisation : concepts de base*. Les commandes et les tâches utilisées pour créer, gérer et conserver des tables partitionnées sont les suivantes :

- `alter database` : permet à la base de données d'accéder aux devices.
- `sp_addsegment` : crée un segment sur un device ; `sp_extendsegment` étend le segment à d'autres devices et `sp_dropsegment` supprime le journal et les segments système des devices des données.
- `create table...on nom_segment` crée une table sur un segment.
- `alter table...partition` et `alter table...unpartition` ajoutent ou suppriment un partitionnement de table.
- `create clustered index` répartit les données de façon uniforme entre les différentes partitions de la table.
- `bcp (bulkcopy)` copie les données dans des partitions spécifiques de la table (identifiées par leur numéro à la suite du nom de la table).
- `sp_helppartition` affiche le nombre de partitions et la répartition des données entre les différentes partitions, tandis que `sp_helpsegment` vérifie l'espace utilisé sur chacun des devices d'un segment et dans l'ensemble du segment.

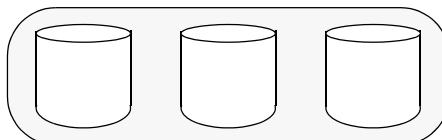
La [figure 7-8](#) présente un schéma possible pour la création d'une table partitionnée.

Figure 7-8 : Etapes de la création et du chargement d'une nouvelle table partitionnée

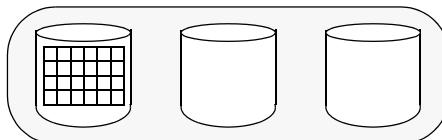
`alter database` permet à la base de données d'accéder aux devices.



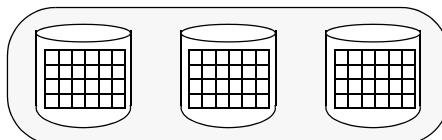
`sp_addsegment` crée un segment sur un device, `sp_extendsegment` étend le segment à d'autres devices et `sp_dropsegment` supprime le journal et les segments système des devices des données.



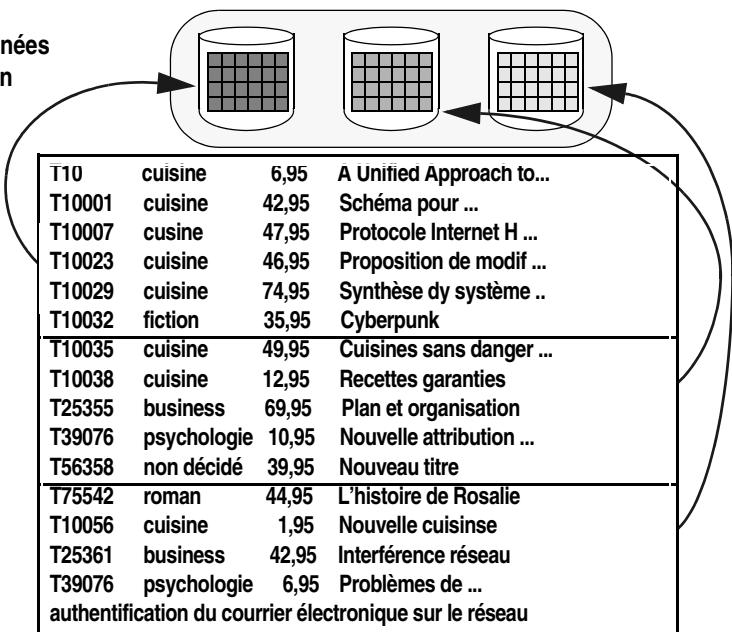
`create table...on nom_segment` crée la table sur le segment.



`alter table...partition` crée une partition sur chaque device.



Bulkcopy parallèle charge les données dans chaque partition à partir d'un fichier d'entrée..



Equilibre ressources/performances

Des performances optimales en mode parallèle exigent de nombreuses CPU et devices E/S pour permettre la parallélisation des E/S. Comme souvent lorsqu'il s'agit de configurer les performances, les systèmes parallèles atteignent un point au-delà duquel l'ajout de ressources n'améliore plus les performances.

Vous devez déterminer si les requêtes font un usage intensif des opérations CPU ou des E/S et repérer les moments de saturation de la CPU ou d'étranglement des E/S qui freinent les performances. Si l'utilisation de la CPU est faible, la répartition d'une table sur plusieurs devices et le recours à plusieurs processus de travail peut y remédier et améliorer les temps de réponse. En revanche, si l'utilisation de la CPU est extrêmement forte mais que le système d'E/S n'est pas saturé, l'accroissement du nombre d'unités centrales peut améliorer les performances.

Ressources CPU

Si le nombre de moteurs (ressources CPU) est insuffisant, les tâches et les processus de travail doivent attendre avant d'accéder aux moteurs Adaptive Server et le temps de réponse peut être relativement long. De multiples facteurs déterminent le nombre de moteurs requis par le système. En particulier, cherchez à savoir si la requête fait un usage intensif d'opérations CPU ou d'E/S ou, à des moments différents, des deux à la fois, et tenez-compte d'éléments tels que ceux-ci :

- Les processus de travail passent dans l'ensemble beaucoup de temps à attendre les E/S disque et autres ressources système tandis que d'autres tâches sont actives sur la CPU.
- Les requêtes comportant des tris et agrégats ont tendance à beaucoup solliciter la CPU.
- Les classes d'exécution et les liaisons avec des moteurs spécialisés dans des requêtes parallèles qui utilisent intensivement la CPU peuvent avoir des conséquences complexes sur le système. Si le nombre d'unités centrales est insuffisant, les performances des requêtes, aussi bien en série qu'en parallèle, risquent de diminuer.

Reportez-vous au [Chapitre 5, « Répartition des ressources des moteurs »](#) du manuel *Performances et optimisation : concepts de base* pour de plus amples informations.

Ressources disque et E/S

Dans la plupart des cas, la configuration de la structure physique des tables et des index sur les devices détermine les performances du traitement en parallèle. La répartition des partitions entre différents disques et contrôleurs peut améliorer les performances au cours d'un balayage reposant sur les partitions, à condition que toutes les conditions suivantes soient remplies :

- les données sont réparties sur différents disques ;
- ces disques sont répartis sur plusieurs contrôleurs différents ;
- le nombre de processus de travail disponibles est suffisant au moment de l'exécution pour allouer un processus à chaque partition.

Exemple d'optimisation : saturation de la CPU et des E/S

Une expérience portant sur une requête fortement consommatrice de CPU a permis de mettre en évidence une évolution presque linéaire des performances en cas d'ajout de CPU jusqu'à saturation du sous-système d'E/S. En effet, une fois ce stade atteint, l'ajout de ressources CPU ne permet pas d'améliorer les performances. La requête procède à un balayage d'une table de 800 Mo comportant 30 partitions, en utilisant des E/S de 16 Ko. Le [tableau 7-3](#) illustre l'évolution des CPU.

Tableau 7-3 : Evolution des moteurs et processus de travail

Moteurs	Temps écoulé (en secondes)	Utilisation CPU	Saturation E/S	Débit par device, par seconde
1	207	100%	Non saturé	0,13 Mo
2	100	98.7%	Non saturé	0,27 Mo
4	50	98%	Non saturé	0,53 Mo
8	27	93%	Saturé à 100%	0,99 Mo

Recommandations pour la configuration des requêtes parallèles

Le traitement en parallèle impose aux ressources système des exigences très différentes de celles de l'exécution des mêmes requêtes en série. Deux éléments sont essentiels à la planification du traitement en parallèle :

- une bonne compréhension des capacités du matériel sous-jacent (en particulier des unités de disque et des contrôleurs) utilisé sur votre système ;
- un ensemble d'objectifs de performances pour les requêtes que vous envisagez d'exécuter en parallèle.

Recommandations relatives au matériel

Vous trouverez ci-après une liste de recommandations concernant la configuration matérielle et la vitesse des E/S disque :

- Chaque moteur Adaptive Server peut prendre en charge environ cinq processus de travail avant saturation, dans le cas où la CPU est utilisée pour des requêtes intensives. Si la CPU n'est pas saturée à ce stade et que vous souhaitez améliorer les performances des requêtes parallèles, vous devez accroître le nombre de processus de travail par moteur jusqu'à ce que la largeur de bande E/S devienne un goulet d'étranglement.
- Dans le cadre de balayages séquentiels, tels que des balayages de tables faisant appel à des E/S de 16 Ko, il est possible d'atteindre 1,6 Mo par seconde et par device, c'est-à-dire 100 E/S de 16 Ko ou 800 pages par seconde et par device.
- Dans le cas de requêtes procédant à un accès aléatoire, notamment l'accès à un index non clusterisé, le chiffre est d'environ 50 E/S de 2 Ko ou 50 pages par seconde et par device.
- Un contrôleur d'E/S peut soutenir un taux de transfert pouvant atteindre 10 à 18 Mo par seconde. Cela signifie qu'un contrôleur d'E/S SCSI peut prendre en charge jusqu'à 6 à 10 devices effectuant des balayages séquentiels. Certains contrôleurs de disque haut de gamme peuvent prendre en charge des débits supérieurs. Vérifiez vos spécifications matérielles et utilisez, pour vos calculs, des cadences soutenues plutôt que des débits maximum.

- Les disk arrays RAID peuvent présenter des caractéristiques de performances très variées, en fonction du niveau RAID, du nombre de devices dans le « stripe set » et de caractéristiques spécifiques, notamment la mise en cache. Les devices RAID peuvent fournir, pour la parallélisation, un débit supérieur ou inférieur à celui qui serait obtenu avec le même nombre de disques physiques sans « striping ». Dans la plupart des cas, vous devez tenter d'optimiser les requêtes parallèles en définissant le nombre de partitions pour les tables des devices en fonction du nombre de disques du disk array.

Objectifs de performances et recommandations matérielles

Les exemples ci-après se réfèrent aux recommandations matérielles et aux données du [tableau 7-3](#) pour montrer comment remplir les objectifs de performances en utilisant la parallélisation :

- Le nombre de partitions de chaque table doit être inférieur ou égal au nombre de devices. Dans le cadre de l'expérience faisant apparaître l'évolution des moteurs et des processus de travail présentée dans le [tableau 7-3](#), 30 devices étaient disponibles. C'est pourquoi 30 partitions ont été utilisées. En effet, les performances sont optimales lorsque chaque partition se trouve sur un device physique distinct.
- Déterminez le nombre de partitions en fonction du débit d'E/S à atteindre. Si vos disques et contrôleurs peuvent gérer 1 Mo par seconde et par device et si vous voulez que le balayage d'une table de 800 Mo s'effectue en 30 secondes, vous devez atteindre un débit total d'environ 27 Mo par seconde ; vous avez donc besoin d'au moins 27 devices avec une partition par device et d'au moins 27 processus de travail, un pour chaque partition. Ces chiffres sont très proches des débits d'E/S indiqués dans le [tableau 7-3](#).
- Evaluatez le nombre de CPU en fonction du nombre de partitions, puis déterminez le nombre optimal en analysant à la fois l'utilisation de CPU et la saturation des E/S. Dans l'exemple présenté dans le [tableau 7-3](#), 30 partitions étaient disponibles. Si l'on se réfère aux recommandations relatives au matériel (1 CPU pour 5 devices), il faut utiliser 6 moteurs pour les requêtes consommatrices de temps CPU. A ce niveau, les E/S mentionnées dans l'exemple n'étaient pas saturées, de sorte que l'ajout de moteurs a permis d'améliorer les temps de réponse.

Exemples d'optimisation des requêtes parallèles

Les exemples ci-après utilisent les capacités d'E/S décrites à la section « [Recommandations relatives au matériel](#) », page 179.

Amélioration des performances d'un balayage de table

Cet exemple indique comment une table peut être partitionnée pour répondre aux objectifs de performances. Les requêtes qui procèdent à un balayage de tables entières et renvoient un nombre de lignes limité sont idéales sur le plan des performances parallèles. La requête ci-après contenant une clause group by en est un bon exemple :

```
select type, avg(price)
      from titles
     group by type
```

Vous trouverez ci-après les statistiques relatives aux performances et les objectifs d'optimisation :

Taille de la table	48 000 pages
Méthode d'accès	Balayage de table, E/S de 16 Ko
Temps de réponse série	60 secondes
Performances ciblées	6 secondes

Les étapes de configuration d'un fonctionnement parallèle sont les suivantes :

- Créez 10 partitions pour la table et répartissez les données de façon égale entre ces partitions.
- Affectez aux paramètres de configuration number of worker processes et max parallel degree une valeur supérieure ou égale à 10.
- Vérifiez que la table utilise un cache configuré pour des E/S de 16 Ko.

Dans le cas d'une exécution en série, 48 000 pages peuvent être balayées en 60 secondes en utilisant des E/S de 16 Ko. Dans le cas d'une exécution en parallèle, chaque processus balaye 1 partition, soit environ 4 800 pages, en 6 secondes approximativement, toujours avec des E/S de 16 Ko.

Amélioration des performances pour un balayage d'index non clusterisé

L'exemple ci-après montre comment il est possible d'améliorer les performances d'une requête utilisant un balayage d'index non clusterisé, en configurant celle-ci pour un balayage basé sur le hachage. Les statistiques relatives aux performances et les objectifs d'optimisation sont indiqués ci-après :

Pages de données	1500
Méthode d'accès	Index non clusterisé, E/S de 2 Ko
Temps de réponse série	30 secondes
Performances ciblées	6 secondes

Les étapes de configuration d'un fonctionnement parallèle sont les suivantes :

- Affectez au paramètre `max scan parallel degree` la valeur 5 afin d'utiliser 5 processus de travail dans le balayage reposant sur le hachage.
- Affectez aux paramètres `number of worker processes` et `max parallel degree` une valeur supérieure ou égale à 5.

Dans l'exécution en parallèle, chaque processus de travail balaie 300 pages en 6 secondes.

Recommandations pour le partitionnement et le degré de parallélisation

Vous trouverez ci-après quelques recommandations supplémentaires à prendre en compte pour passer du mode d'exécution des requêtes en série au mode parallèle ou si vous envisagez un nouveau partitionnement ou de nouveaux processus de travail pour un système qui exécute déjà des requêtes parallèles :

- Si le taux de présence dans le cache associé à une table est supérieur à 90 %, le partitionnement de cette table ne permettra pas une forte amélioration des performances. En effet, comme la plupart des pages nécessaires se trouvent dans le cache, la parallélisation des E/S physiques ne présente aucun intérêt.

- Si l'utilisation de la CPU est supérieure à 80 % et qu'un pourcentage important de requêtes de votre système fait appel aux requêtes parallèles, l'accroissement du degré de parallélisation risque d'entraîner une saturation de la CPU. Cette recommandation s'applique également dans le cas où l'utilisateur passe du traitement en série au traitement en parallèle, quand un grand nombre de requêtes doit faire appel à la parallélisation. Dans ce cas, vous pouvez utiliser davantage de moteurs ou commencer par un faible degré de parallélisation.
- Si l'utilisation de la CPU est importante et qu'un petit nombre d'utilisateurs seulement exécutent des requêtes DSS volumineuses, tandis que la plupart exécutent des requêtes OLTP qui ne fonctionnent pas en parallèle, l'activation ou l'augmentation de la parallélisation peut améliorer le temps de réponse des requêtes DSS. Toutefois, si le temps de réponse des requêtes OLTP est critique, commencez par un faible degré de parallélisation ou apportez des changements mineurs au degré de parallélisation existant.
- Si l'utilisation de la CPU est faible, orientez-vous progressivement vers un degré de parallélisation supérieur. Sur un système équipé de deux CPU où l'utilisation moyenne de la CPU est de 60 %, le doublement du nombre de processus de travail risquerait de saturer les CPU.
- Si les E/S sur les devices sont bien inférieures au seuil de saturation, vous pouvez améliorer les performances de certaines requêtes en ne respectant pas le principe d'une « partition par device ». Sauf pour les devices RAID, utilisez toujours un multiple du nombre de devices logiques dans un segment de partitionnement : ainsi, pour une table ou un segment avec quatre devices, utilisez huit partitions. Si vous utilisez deux fois plus de partitions par device, les mouvements des têtes de lecture risquent d'augmenter tandis que la parallélisation E/S diminue. Dans ce cas-là, la création d'un index sur une table disposant plus de partitions que de devices provoque l'affichage d'un message que vous pouvez ignorer.

Expériences avec des sous-ensembles de données

Le traitement de requêtes parallèles peut donner lieu à des gains de performances considérables sur les tables les plus importantes et avec la plupart des requêtes fortement consommatrices d'E/S. Cependant, l'expérimentation de différentes structures physiques sur des tables volumineuses prend beaucoup de temps. Vous trouverez ci-après quelques suggestions pour travailler avec des sous-ensembles de données réduits :

- Pour une exploration initiale visant à déterminer les types de plan d'exécution des requêtes que choisirait l'optimiseur, choisissez un sous-ensemble proportionnellement représentatif de vos données. Par exemple, si vous disposez d'une table de 50 millions de lignes jointe à une table de 5 millions de ligne, choisissez de travailler avec un dixième seulement des données, en utilisant respectivement 5 millions et 500 000 lignes. Sélectionnez des sous-ensembles de tables constituant des jointures valables. Faites attention à la sélectivité de la jointure : si la jointure de la table s'exécute en parallèle (elle doit pour cela renvoyer au moins 20 lignes par balayage), vérifiez que le sous-ensemble défini reflète cette sélectivité de jointure.
- L'optimiseur ne prend pas en compte les devices physiques sous-jacents mais seulement le partitionnement des tables. Durant la phase d'exploration préalable à l'optimisation, la répartition des données sur des devices physiques distincts permet d'obtenir des prévisions plus précises quant aux caractéristiques probables de votre système de production avec les tables complètes. Vous pouvez partitionner des tables qui résident sur un seul et même device et ignorer les éventuels messages d'avertissement qui s'affichent au cours des premières étapes de votre travail de planification (telles que le test des paramètres de configuration, le partitionnement des tables et la vérification de l'optimisation de vos requêtes). Naturellement, cette démarche ne permet pas d'obtenir des statistiques précises sur les E/S.

Le travail portant sur des sous-ensembles de données peut vous aider à déterminer les plans d'exécution des requêtes parallèles et le degré de parallélisation pour les tables. Une différence toutefois est que, sur des tables de petite taille, les tris sont effectués en série alors que sur des tables plus volumineuses, ils s'exécuteraient en parallèle.

Incidence au niveau du système

Outre les effets de la parallélisation déjà évoqués, vous trouverez ci-après plusieurs aspects à considérer pour introduire un mode d'exécution parallèle dans des environnements mixtes DSS et OLTP. Votre objectif doit être d'améliorer les performances des DSS grâce à la parallélisation, sans qu'il y ait d'incidence sur les performances des applications OLTP.

Problèmes de verrouillage

Déetectez les éventuels conflits de verrous :

- Les requêtes parallèles sont plus lentes que les requêtes évaluées en l'absence de tout conflit. Si les balayages détectent de nombreuses pages avec des verrous exclusifs dus à des mises à jour, les performances peuvent se trouver modifiées.
- Si les requêtes parallèles renvoient un grand nombre de lignes utilisant des fusions de buffers en réseau, il est probable que d'importants conflits se manifestent au niveau de ces derniers. Au cours des balayages, les requêtes posent des verrous partagés sur les pages de données. Ainsi, pour modifier des données, il peut être nécessaire d'attendre la libération de ces verrous. Dans ce cas, vous pouvez peut-être envisager de réserver ces types de requêtes pour une exécution en série.
- Si vos applications font l'objet d'interblocages lorsque les requêtes DSS s'exécutent en série, ces interblocages risquent d'augmenter en cas de traitement parallèle de ces requêtes. La transaction annulée au cours des interblocages est le plus souvent une requête OLTP car la décision d'annulation des interblocages s'appuie sur le temps CPU cumulé des processus concernés.

Reportez-vous à la section « [Interblocages et concurrence d'accès](#) », [page 88](#) du manuel *Performances et optimisation : verrouillage* pour plus d'informations sur les interblocages.

Problèmes de devices

La configuration de plusieurs devices pour tempdb devrait améliorer les performances des requêtes parallèles faisant appel à des tables de travail, en particulier celles comportant des tris et agrégats et celles utilisant la stratégie de redéfinition d'index.

Conséquences du cache des procédures

Les plans d'exécution des requêtes parallèles sont légèrement plus volumineux que ceux des requêtes série parce qu'ils contiennent des instructions supplémentaires sur les partitions ou les pages auxquelles les processus de travail doivent pouvoir accéder.

Dans le cas de requêtes ad hoc, chaque processus de travail doit disposer d'une copie du plan d'exécution de la requête. L'espace provenant du cache des procédures est utilisé pour conserver ces plans en mémoire et il est de nouveau disponible pour le cache des procédures lorsque l'exécution de la requête ad hoc est terminée.

Les procédures stockées dans le cache sont invalidées chaque fois que vous modifiez les paramètres de configuration de la parallélisation, `max parallel degree` et `max scan parallel degree`. Lors de la prochaine exécution d'une requête, cette dernière est lue à partir du disque, puis recompilée.

Cas dans lesquels les résultats des requêtes parallèles peuvent être différents

Lorsqu'une requête ne comprend ni agrégat vectoriel, ni agrégat scalaire et qu'elle ne nécessite pas de tri final, son exécution en parallèle peut renvoyer des résultats dont l'ordre diffère de celui de la même requête exécutée en série et les exécutions successives de la même requête en parallèle peuvent donner des résultats placés chaque fois dans un ordre différent.

Les résultats de requêtes parallèles et série comportant des agrégats vectoriels ou agrégats scalaires ou nécessitant un tri final sont retournés après que tous les résultats des tables de travail ont été fusionnés ou triés lors de l'étape finale du traitement de la requête. En l'absence de clause demandant cette étape finale, les requêtes parallèles envoient des résultats au client au moyen d'une fusion de buffers réseau, c'est-à-dire que chaque processus de travail envoie des résultats au buffer réseau à mesure qu'il extrait les données satisfaisant les requêtes.

La vitesse relative des différents processus de travail entraîne des différences dans l'ordre des résultats. Chaque balayage parallèle se comporte différemment du fait des pages qui se trouvent déjà dans le cache, des conflits de verrous et ainsi de suite. Les requêtes parallèles retournent toujours le même *jeu* de résultats, mais ces derniers ne sont pas toujours placés dans le même *ordre*. Pour obtenir un ordre fiable des résultats, vous devez utiliser `order by` ou exécuter la requête en mode série.

En outre, compte tenu de la régulation des processus de travail multiples procédant à la lecture des pages de données, deux types de requêtes accédant aux mêmes données peuvent retourner des résultats différents lorsque aucun agrégat ou tri final n'est effectué :

- requêtes utilisant `set rowcount` ;
- requêtes sélectionnant une colonne dans une variable locale, sans clauses de requête suffisamment restrictives.

Requêtes utilisant `set rowcount`

L'option `set rowcount` interrompt le traitement une fois qu'un certain nombre de lignes ont été retournées au client. Avec le traitement en série, les résultats obtenus sont identiques lors des exécutions successives. En mode série, les mêmes lignes sont retournées dans le même ordre pour une valeur `rowcount` donnée, car un même processus lit systématiquement les pages de données dans le même ordre.

Dans le cas de requêtes parallèles, l'ordre des résultats et le jeu de lignes peuvent varier, parce que les processus de travail peuvent accéder aux pages avant ou après d'autres processus. Lorsque `set rowcount` est actif, chaque ligne est écrite dans le buffer réseau dès son extraction et le buffer est ensuite envoyé au client lorsqu'il est saturé, jusqu'à ce que le nombre de lignes requis ait été retourné. Pour obtenir des résultats homogènes, vous devez soit utiliser une clause procédant à un tri final, soit exécuter la requête en mode série.

Requêtes définissant des variables locales

Cette requête définit la valeur d'une variable locale dans une instruction select :

```
select @tid = title_id from titles  
      where type = "business"
```

La clause where correspond à différentes lignes de la table titles. Ainsi, la variable locale prend toujours la valeur de la dernière ligne correspondante retournée par la requête. Cette valeur est toujours la même dans le traitement en série mais, dans le traitement de requêtes parallèles, les résultats dépendent du dernier processus de travail réalisé. Pour obtenir des résultats homogènes, veillez à utiliser une clause procédant à un tri final, exécutez la requête en mode série ou ajoutez des clauses pour que les arguments de la requête sélectionnent des lignes uniques.

Homogénéisation des résultats

Pour obtenir des résultats homogènes pour tous les types de requête présentés dans cette section, vous pouvez soit ajouter une clause pour provoquer un tri final, soit exécuter les requêtes en mode série. Les clauses permettant un tri final sont :

- order by
- distinct, à l'exception de l'utilisation de cette dernière au sein d'un agrégat, tel que avg(distinct price)
- union, mais pas union all

Pour exécuter des requêtes en mode série, vous pouvez :

- utiliser set parallel_degree 1 pour limiter la session au traitement en série ;
- inclure la clause (parallel 1) après chacune des tables énumérées dans la clause from de la requête.

Optimisation des requêtes parallèles

Ce chapitre décrit les principales techniques employées par Adaptive Server pour effectuer des requêtes parallèles et explique comment l'optimiseur applique ces techniques à des requêtes différentes.

L'optimisation des requêtes parallèles est un processus automatique et les plans d'exécution de requêtes optimisés, élaborés par Adaptive Server, proposent généralement la solution la plus adaptée à chaque requête.

Il est cependant utile de connaître les mécanismes internes d'une requête parallèle pour mieux comprendre pourquoi les requêtes sont parfois exécutées en série ou avec des processus de travail beaucoup moins nombreux que vous ne le pensiez. C'est en maîtrisant ces éléments que vous pourrez modifier votre système pour que certaines requêtes soient traitées en parallèle avec le nombre de processus désiré.

Sujet	Page
Concept d'optimisation des requêtes parallèles	190
Conditions d'exécution de l'optimisation	191
Coûts d'overhead	191
Méthodes d'accès parallèles	192
Récapitulatif des méthodes d'accès parallèles	205
Degré de parallélisation des requêtes parallèles	207
Exemples de requête parallèle	216
Ajustements lors de l'exécution des processus de travail	224
Identification des problèmes liés aux performances des requêtes parallèles	229
Limites d'utilisation des ressources dans les requêtes parallèles	231

Concept d'optimisation des requêtes parallèles

L'optimisation des requêtes parallèles est un processus qui consiste à analyser une requête et à choisir la meilleure combinaison de méthodes d'accès en parallèle et en série pour obtenir un temps de réponse optimal. Elle est une extension des stratégies d'optimisation des requêtes en série traitée dans les précédents chapitres. Outre l'évaluation des coûts visant à optimiser les requêtes en série, le processus d'optimisation parallèle analyse le coût des méthodes d'accès en parallèle pour chaque combinaison d'ordres de jointure, de types de jointures et d'index. L'optimiseur peut choisir n'importe quelle combinaison de méthodes d'accès en série et en parallèle pour créer le plan d'exécution de requête le plus rapide.

Optimisation du temps de réponse par rapport à la charge de travail globale

L'optimisation des requêtes en série sélectionne le plan d'exécution le plus avantageux. La requête ne peut être exécutée que par un seul processus ; c'est pourquoi le plan le plus rentable est celui qui garantit un temps de réponse optimal *et* qui représente une charge de travail minimale pour le serveur.

L'objectif de l'exécution en parallèle est d'obtenir le temps de réponse le plus rapide, même si cela représente une charge supplémentaire pour le serveur. Lors de l'optimisation des requêtes parallèles, l'optimiseur applique des techniques de comparaison des coûts semblables à celles qui sont utilisées lors de l'optimisation en série pour définir le plan d'exécution de requête définitif.

Cependant, étant donné que les requêtes parallèles sont traitées par plusieurs processus de travail, le plan d'exécution exige des ressources Adaptive Server plus importantes. Les différents processus de travail, les moteurs et les partitions permettant d'accélérer une requête requièrent des ressources supplémentaires en termes d'overhead, d'utilisation de la CPU et d'accès sur le disque. Autrement dit, l'optimisation des requêtes en série permet d'améliorer les performances en limitant l'utilisation des ressources du serveur, tandis que l'optimisation des requêtes parallèles permet d'améliorer les performances de requêtes individuelles en utilisant la totalité des ressources disponibles pour obtenir un temps de réponse optimal.

Conditions d'exécution de l'optimisation

L'optimiseur envisage des plans d'exécution de requête en parallèle uniquement lorsque Adaptive Server et la session en cours sont configurés pour la parallélisation, comme indiqué à la section « [Contrôle du degré de parallélisation](#) », page 167.

Si Adaptive Server et la session active sont configurés pour un traitement en parallèle, toutes les requêtes de la session peuvent faire l'objet d'une optimisation en parallèle. Les requêtes individuelles peuvent également être optimisées avec les options `parallel N` (optimisation en parallèle) ou `parallel 1` (optimisation en série) de l'optimiseur.

Si Adaptive Server ou la session active ne sont pas configurés pour le traitement des requêtes parallèles ou si une requête utilise les options de l'optimiseur pour appliquer un traitement en série, l'optimiseur peut recourir aux méthodes d'accès en série ; les méthodes d'accès en parallèle décrites dans ce chapitre ne sont pas prises en compte.

Adaptive Server n'exécute pas de requête parallèle sur les tables système.

Coûts d'overhead

Les requêtes parallèles requièrent des coûts d'overhead importants pour effectuer les opérations internes suivantes :

- allocation et initialisation des processus de travail ;
- coordination des processus de travail lorsqu'ils exécutent un plan d'exécution de requête ;
- libération d'un processus de travail lorsque la requête est terminée.

Pour éviter d'appliquer ces coûts d'overhead aux requêtes OLTP, l'optimiseur « interdit » à certaines tables d'utiliser des méthodes d'accès en parallèle lorsqu'un balayage accède à moins de 20 pages de données. Cette restriction est applicable, qu'un index soit utilisé ou non pour accéder aux données de la table. Lorsque Adaptive Server doit balayer moins de 20 pages dans une table, l'optimiseur tient compte uniquement des balayages en série des tables et des index et laisse de côté les méthodes d'optimisation en parallèle.

Facteurs non pris en compte

Lorsqu'il calcule les ressources requises par une méthode d'accès en parallèle, l'optimiseur ne tient *pas* compte de certains facteurs, tels que le nombre de moteurs disponibles, le rapport moteurs/CPU, ainsi que la présence ou non des partitions d'une table sur des devices physiques et des contrôleurs dédiés. Chacun de ces facteurs peut avoir une influence déterminante sur les performances d'une requête. L'administrateur système doit donc s'assurer que ces ressources sont configurées de manière optimale sur la totalité du système Adaptive Server.

Pour plus d'informations sur la configuration d'Adaptive Server, reportez-vous à la section « [Paramètres de configuration pour le contrôle de la parallélisation](#) », page 168.

Reportez-vous à la section « [Commandes servant au partitionnement des tables](#) », page 114 du guide *Performances et optimisation : concepts de base* pour plus d'informations sur le partitionnement des données en vue de faciliter les requêtes parallèles.

Méthodes d'accès parallèles

Les sections suivantes décrivent les méthodes d'accès en parallèle, ainsi que d'autres stratégies examinées par l'optimiseur lors de l'optimisation des requêtes parallèles. Les méthodes d'accès en parallèle entrent dans les catégories générales suivantes :

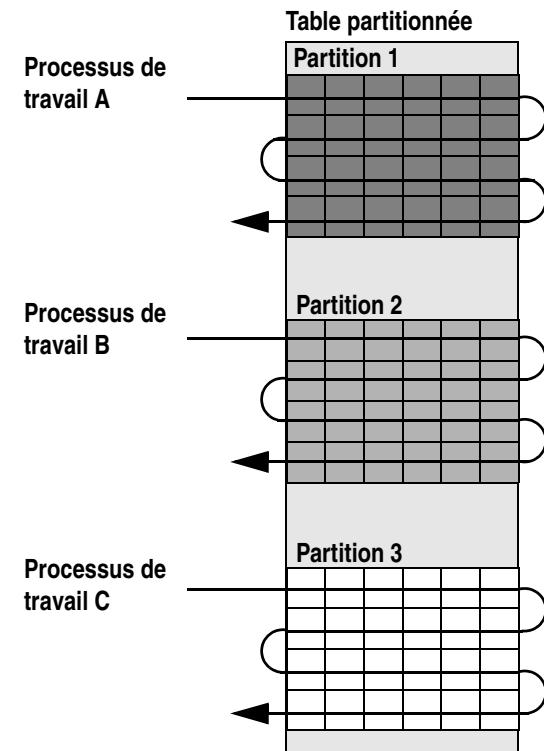
- Les **méthodes fondées sur des partitions** utilisent deux processus de travail ou plus pour accéder à des partitions séparées d'une table. Ces méthodes permettent d'obtenir les meilleurs temps de réponse car elles répartissent la charge de travail entre les CPU et les disques physiques lors de l'accès à une table. Au niveau de la CPU, les processus de travail peuvent être placés dans des files d'attente affectées à des moteurs différents pour accroître les performances du traitement. Au niveau du disque physique, les processus de travail peuvent traiter des E/S indépendamment les uns des autres lorsque les partitions de la table sont réparties sur des devices physiques et des contrôleurs distincts.

- Les **méthodes fondées sur le hachage** permettent l'accès en parallèle à des tables partitionnées, par balayage de table ou d'index, selon le cas. Ces méthodes font appel à de nombreux processus de travail pour effectuer des opérations sur une chaîne de pages de données ou sur un ensemble de pages d'index. Les E/S ne sont pas réparties entre des devices physiques et des contrôleurs ; en revanche, il est possible de placer les processus de travail dans des files d'attente affectées à de nombreux moteurs différents, de manière à répartir la charge et améliorer les temps de réponse.
- Les **méthodes fondées sur des intervalles** permettent un accès parallèle lors des jointures par fusion sur des tables partitionnées et non partitionnées, y compris des tables de travail créées pour le tri et la fusion, ainsi qu'un accès par le biais d'index. Le partitionnement de ces tables n'est pas pris en compte lors du choix du degré de parallélisation, de sorte qu'elles ne sont pas réparties sur des devices physiques ou des contrôleurs. Les processus de travail peuvent être placés dans des files d'attente affectées à de nombreux moteurs différents, de manière à répartir la charge et à améliorer les temps de réponse.

Balayage d'une partition en parallèle

Au cours d'un balayage de partition en parallèle, de nombreux processus de travail analysent complètement chaque partition d'une table. Un processus de travail est affecté à chaque partition et lit toutes les pages qu'elle contient. La [figure 8-1](#) représente une balayage de partition en parallèle.

Figure 8-1 : Balayage d'une partition en parallèle



Le balayage des partitions en parallèle est plus rapide qu'un balayage de table en série. La charge de travail est répartie entre plusieurs processus de travail qui s'exécutent simultanément sur des moteurs différents. Certains processus peuvent être exécutés pendant que d'autres attendent des E/S ou d'autres ressources système. Si les partitions de la table se trouvent sur des devices physiques distincts, la parallélisation des E/S est également possible.

Conditions à prendre en compte

L'optimiseur n'effectue des balayages de partitions en parallèle que lorsque la requête porte sur des tables partitionnées. Les données de la table doivent être correctement réparties par rapport au nombre de partitions, faute de quoi l'optimiseur ne prend pas en compte les méthodes d'accès fondées sur les partitions. On considère que les données d'une table sont mal réparties lorsque la taille de la partition la plus grande est au moins deux fois supérieure à la taille de partition moyenne.

Enfin, la requête doit au moins accéder à 20 pages de données pour que l'optimiseur sélectionne une méthode d'accès en parallèle.

Modèle de calcul des coûts

L'optimiseur d'Adaptive Server calcule le coût d'un balayage en parallèle des partitions d'une table en tenant compte du plus grand nombre d'E/S physiques et logiques effectuées par un processus de travail lors du balayage. En d'autres termes, le coût total de cette méthode d'accès est égal au nombre d'E/S requises pour lire toutes les pages figurant dans la plus grande partition de la table.

Par exemple, si une table dotée de 3 partitions comporte 200 pages dans la première partition, 300 dans la seconde et 500 dans la troisième, le coût lié au balayage des partitions est égal à 500 E/S logiques et 500 E/S physiques (à raison d'E/S de 2 Ko). A titre de comparaison, le coût d'un balayage en série de cette même table serait de 1000 E/S logiques et physiques.

Balayage de partitions d'index clusterisés en parallèle (table APL)

Le balayage de partitions d'index clusterisés fait intervenir plusieurs processus de travail pour balayer les pages de données dans une table partitionnée lorsque la clé d'index clusterisé correspond à un argument de recherche (SARG). Cette méthode n'est utilisable que sur des tables APL.

Un seul processus de travail est affecté à chaque partition de la table. Chaque processus accède à des pages de données dans la partition en utilisant l'une des deux méthodes, selon la plage de valeurs clés auxquelles le processus peut accéder. Lorsqu'une table partitionnée possède un index clusterisé, des lignes sont affectées aux partitions sur la base de la clé d'index clusterisé.

La [figure 8-2](#) décrit le balayage d'une partition d'index clusterisé qui s'étend sur trois partitions. Les processus de travail A, B et C sont affectés à chacune des trois partitions de la table. Le balayage repose sur deux méthodes :

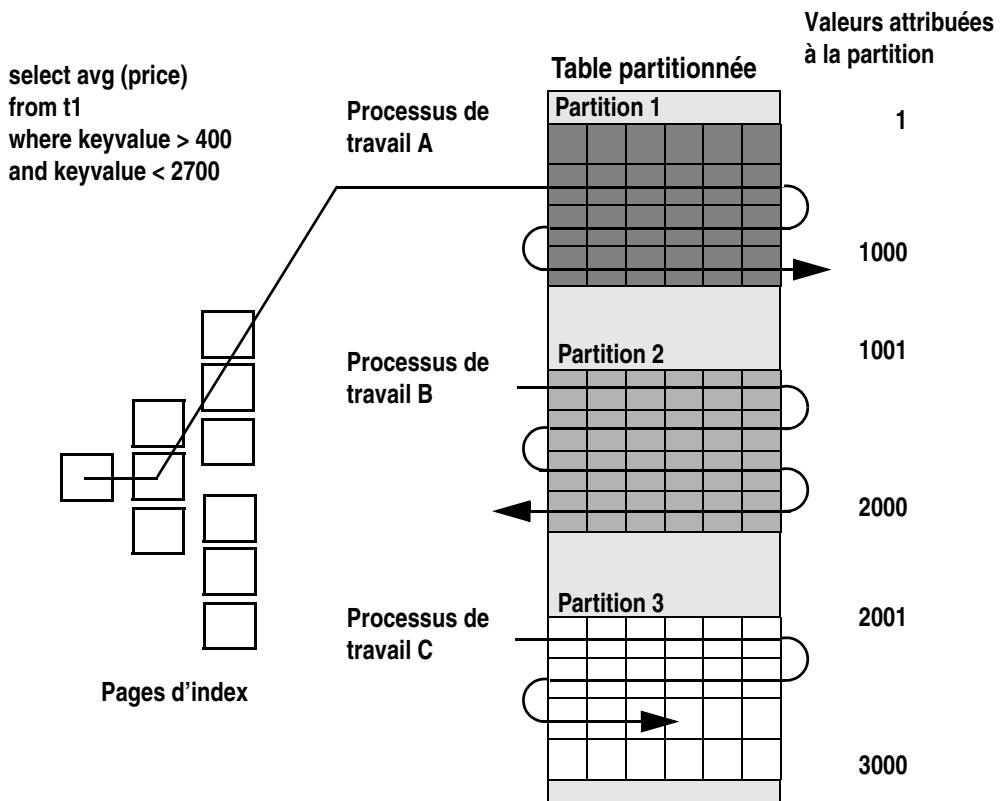
- Méthode 1

Le processus de travail A parcourt l'index clusterisé pour rechercher la première page répondant aux critères de l'argument de recherche, environ à mi-chemin dans la partition 1. Ensuite, il commence le balayage des pages de données jusqu'à la fin de la partition 1.

- Méthode 2

Les processus de travail B et C n'utilisent pas l'index clusterisé ; en revanche, ils balayaient les pages de données depuis le début de leurs partitions respectives. Le processus de travail B achève le balayage lorsqu'il parvient à la fin de la partition 2. Le processus de travail C achève le balayage environ à mi-chemin dans la partition 3, là où les lignes de données ne satisfont plus aux critères de l'argument de recherche.

Figure 8-2 : Balayage de partitions d'index clusterisés en parallèle



Conditions à prendre en compte

L'optimiseur peut recourir à un balayage de partition d'index clusterisé uniquement dans les cas suivants :

- la requête accède à 20 pages de données au minimum dans la table ;
- la table est partitionnée et verrouillée au niveau de toutes les pages ;
- les données de la table sont correctement réparties par rapport au nombre de partitions. On considère que les données d'une table sont mal réparties lorsque la taille de la partition la plus grande est au moins deux fois supérieure à la taille de partition moyenne.

Modèle de calcul des coûts

L'optimiseur d'Adaptive Server évalue différemment le coût lié au balayage d'une partition d'index clusterisé, selon le nombre total de pages à analyser :

- Si le nombre de pages à analyser est inférieur ou égal au double de fois la taille moyenne d'une partition, l'optimiseur évalue le coût du balayage en divisant par deux le nombre de pages à analyser.
- Si le nombre de pages à analyser est supérieur au double de la taille moyenne d'une partition, l'optimiseur évalue le coût du balayage comme égal au nombre moyen de pages dans une partition.

Le coût du balayage peut être supérieur lorsque :

- le nombre de pages à analyser est inférieur à la taille d'une partition et
- la totalité des données à analyser figure dans une partition.

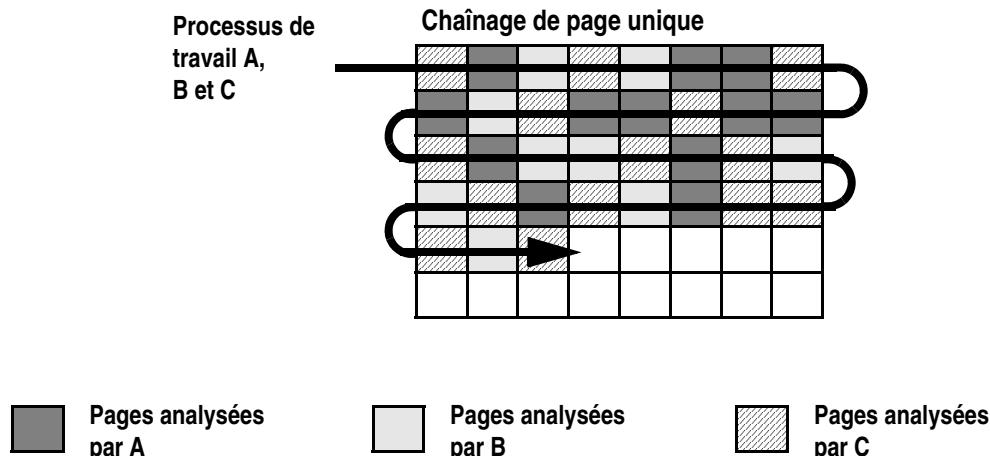
Si ces deux conditions sont réunies, le coût total du balayage est identique à celui qui s'appliquerait s'il s'agissait d'un balayage en série.

Balayage de table en parallèle avec hachage

Les balayages de tables en parallèle avec hachage s'exécutent un peu différemment, selon le plan de verrouillage de la table.

Balayages de tables APL avec hachage

Dans un balayage de tables APL avec hachage, plusieurs processus de travail analysent une chaîne unique de pages de données simultanément. Chacun traverse le chaînage de pages et applique une fonction de hachage interne à chaque ID de page. La fonction de hachage désigne le processus de travail qui lit les lignes dans la page. Avec cette méthode, un seul processus balaie les lignes d'une page donnée. La [figure 8-3](#) illustre ce type de balayage.

Figure 8-3 : Balayage d'une table APL en parallèle avec hachage

Le balayage avec hachage permet de répartir le traitement d'une chaîne de pages de données entre plusieurs moteurs. L'optimiseur utilise cette méthode pour accéder à une table externe dans une requête de jointure et traiter une jointure en parallèle.

Balayages de tables DOL avec hachage

Ce type de balayage effectue un hachage sur le nombre d'extents ou le nombre de pages d'allocation figurant dans la table et non plus sur le nombre de pages. Le choix entre un hachage sur le nombre de pages d'allocation ou sur le nombre d'extents dépend de l'optimiseur, en fonction du coût encouru. Les deux méthodes peuvent réduire le coût des requêtes en parallèle sur des tables non partitionnées. Les requêtes qui, dans une table APL, procèdent à un balayage en série peuvent utiliser l'une des nouvelles méthodes fondées sur le hachage si la table est convertie pour un verrouillage des pages de données seulement.

Conditions à prendre en compte

L'optimiseur n'effectue un balayage de table avec hachage que s'il s'agit d'une table sans index ou d'une table externe dans une requête de jointure. Il n'applique pas cette méthode sur des tables dotées d'index clusterisés ou des requêtes portant sur une seule table. Les balayages fondés sur un hachage s'appliquent indifféremment aux tables non partitionnées ou partitionnées. La requête doit accéder à 20 pages de données au minimum pour que l'optimiseur sélectionne une méthode d'accès en parallèle.

Modèle de calcul des coûts

L'optimiseur calcule les ressources requises par un balayage de table avec hachage en tenant compte du nombre d'E/S logiques et physiques nécessaires.

Pour une table APL, le coût des E/S physiques est à peu près le même que celui d'un balayage de table en série. Pour connaître le coût logique, il suffit de multiplier le nombre de pages à lire par le nombre de processus de travail. Le coût par processus de travail équivaut à une E/S logique pour chaque page de la table et approximativement $1/N$ E/S physiques, N étant le nombre de processus de travail.

Pour une table DOL, le coût d'un balayage en série est pratiquement le même, les E/S physiques et logiques se répartissant également entre les processus de travail.

Balayage d'index en parallèle avec hachage

Les balayages d'index basés sur le hachage peuvent être effectués avec des index non clusterisés ou des index clusterisés sur des tables DOL.

Pour effectuer le balayage :

- tous les processus de travail traversent les niveaux supérieurs de l'index ;
- tous les processus de travail analysent les pages d'index de niveau feuille.

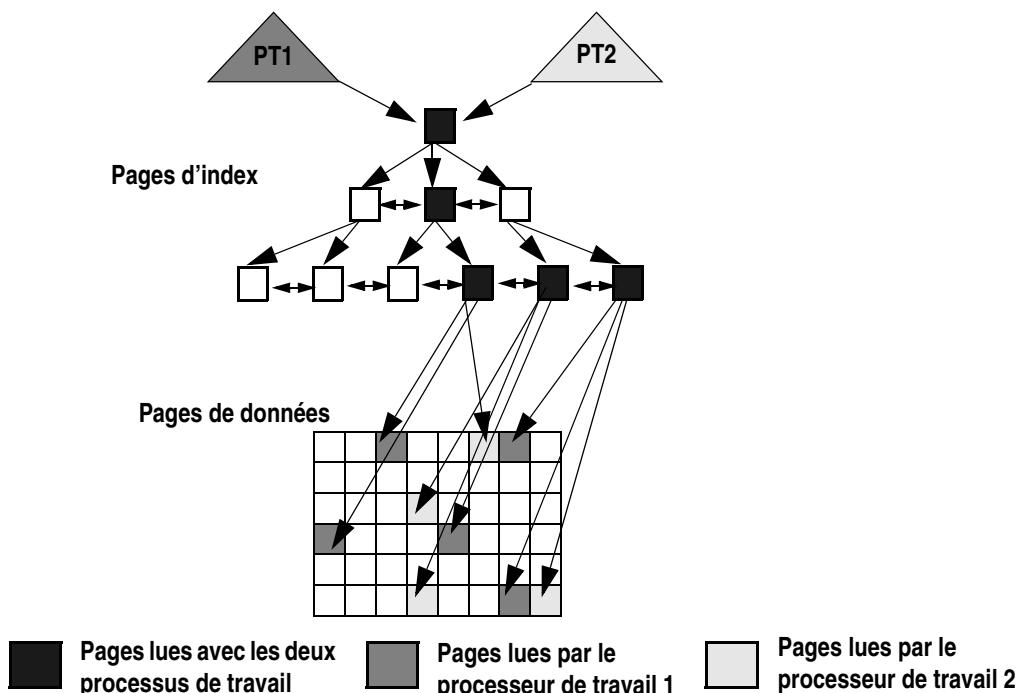
Dans les tables DOL, les processus de travail qui balaiennent le niveau feuille appliquent un hachage à chaque ID de page pour chaque ligne et analysent les pages de données correspondantes.

Dans les tables APL, le balayage avec hachage d'un index est effectué de deux manières différentes, selon qu'il s'agit d'une table sans index ou d'une table dotée d'un index clusterisé. La différence principale est le mécanisme de hachage :

- S'il s'agit d'une table dotée d'un index clusterisé, le hachage se trouve sur les valeurs clés.
- Si la table est sans index, le balayage effectue un hachage sur l'ID de page.

La [figure 8-4](#) illustre un balayage d'index non clusterisé avec hachage sur une table sans index, avec deux processus de travail.

Figure 8-4 : Balayage d'index non clusterisé avec hachage



Modèle de calcul des coûts et conditions requises

L'évaluation du coût de balayage d'un index non clusterisé repose sur la formule suivante :

$$\begin{aligned} \text{Coût de lecture} = & \text{ nombre de niveaux d'index} \\ & + \text{nombre de pages au niveau feuille / pages par E/S} \\ & + (\text{nombre de pages de données / pages par E/S}) / \text{nombre de processus de travail} \end{aligned}$$

L'optimiseur recourt au balayage d'un index avec hachage pour les tables d'une requête qui disposent d'index non clusterisés exploitables et pour les tables DOL avec index clusterisés. La requête doit accéder à 20 pages de données au minimum dans une table.

Remarque Si un index non clusterisé couvre les résultats d'une requête, l'optimiseur n'effectue pas de lecture par hachage de l'index non clusterisé.

Reportez-vous à la section « [Couverture par index](#) », page 317 du guide *Performances et optimisation : concepts de base* pour plus d'informations sur la couverture par index.

Balayages en parallèle reposant sur des intervalles

Les balayages en parallèle reposant sur des intervalles sont utilisés dans les jointures par fusion, précisément pour la phase de fusion.

Lorsque deux tables sont fusionnées en parallèle, chaque processus de travail a un intervalle de valeurs à fusionner. Cet intervalle est déterminé au moyen des statistiques d'histogramme ou par échantillonnage.

Lorsqu'il existe un histogramme pour au moins une des colonnes de jointure, il est utilisé pour partitionner les intervalles afin que chaque processus opère sur environ le même nombre de lignes. Si aucune colonne de jointure ne possède d'histogramme, un échantillonnage semblable à celui réalisé pour les opérations de tri parallèle permet de déterminer les intervalles de valeurs que chaque processus de travail devra fusionner.

La [figure 8-5](#) illustre une jointure parallèle avec fusion à droite.

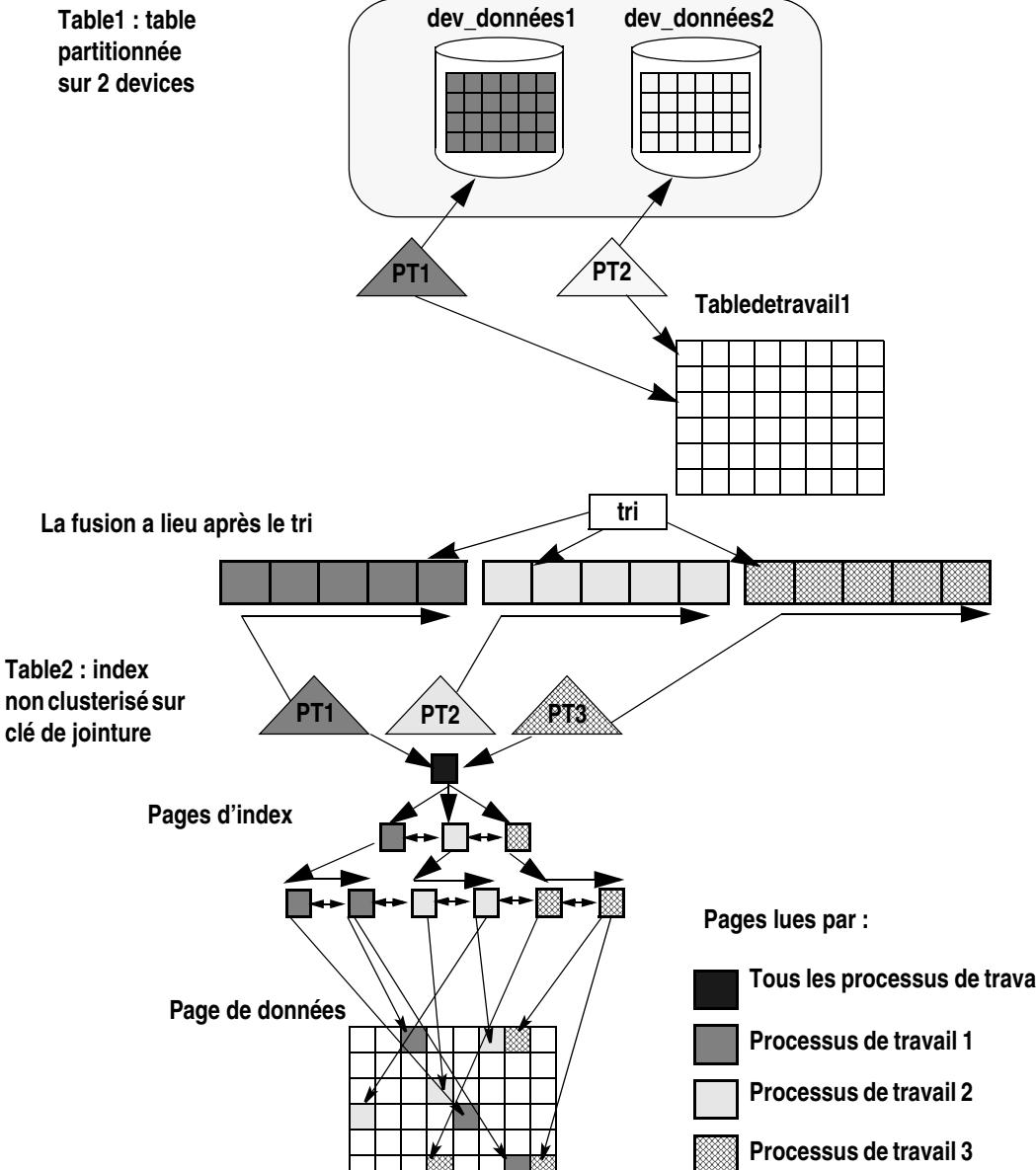
Dans ce cas :

- Une jointure par fusion à droite est effectuée. La Table1, externe, est balayée et convertie en table de travail, triée puis fusionnée avec la table interne. Ces processus de travail sont libérés à la fin de cette étape.
- La table externe a deux partitions, de sorte que deux processus de travail effectuent un balayage de partition en parallèle.
- La table interne a un index non clusterisé sur la clé de jointure. Le paramètre max parallel degree étant défini à 3, trois processus de travail sont utilisés.

Conditions à prendre en compte

L'optimiseur recourt à des jointures par fusion en parallèle lorsque le paramètre de configuration enable merge joins est défini à 1 et que le nombre des accès aux pages dans la table externe de la jointure est supérieur à 20.

Figure 8-5 : Jointure parallèle avec fusion à droite



Méthodes d'accès en parallèle supplémentaires

Adaptive Server peut recourir à d'autres stratégies pour exécuter les requêtes en parallèle. Ces méthodes reposent sur l'utilisation des tables de travail partitionnées et des tris parallèles.

Tables de travail partitionnées

Lorsque les requêtes requièrent une table de travail, Adaptive Server crée une, partitionnée, qu'il remplit en faisant intervenir de nombreux processus de travail. Le partitionnement d'une table de travail améliore les performances lorsque Adaptive Server la remplit et permet donc d'optimiser le temps de réponse de la requête.

Pour obtenir des exemples de requêtes susceptibles de tirer profit de l'utilisation de tables de travail partitionnées, reportez-vous à la section « [Exemples de requête parallèle](#) », page 216.

Tri parallèle

Le tri parallèle fait appel à plusieurs processus de travail pour trier des données en parallèle, comme dans le cas d'une requête exécutée en parallèle. `create index` et toutes les requêtes qui requièrent un tri peuvent faire l'objet d'un tri parallèle.

L'optimiseur de requêtes n'agit pas directement sur le tri parallèle.

Pour obtenir des exemples de requêtes qui peuvent bénéficier d'un tri parallèle, reportez-vous à la section « [Exemples de requête parallèle](#) », page 216.

Reportez-vous également à la section « [Présentation de la méthode de tri parallèle](#) », page 236 qui explique en détail comment Adaptive Server exécute un tri parallèle.

Récapitulatif des méthodes d'accès parallèles

Le [tableau 8-1](#) répertorie les différents cas d'emploi des méthodes d'accès parallèles lors du traitement d'une requête d'Adaptive Server. Dans tous les cas, la requête doit impérativement accéder à un minimum de 20 pages de données dans la table pour que l'optimiseur utilise l'une de ces méthodes.

Tableau 8-1 : Récapitulatif des méthodes d'accès parallèles

Méthode parallèle	Coûts importants	Conditions à prendre en compte	Méthodes en série équivalentes
Balayage reposant sur les partitions	Nombre de pages dans la partition la plus large	Table partitionnée avec une répartition équilibrée des données	Balayage de table en série, balayage d'index en série
Balayage de table avec hachage	Nombre de pages dans la table	N'importe quelle table externe sans index dans une requête de jointure	Balayage de table en série, balayage d'index en série
Balayage de partitions d'index clusterisé	Si le nombre total de pages à balayer $\leq 2 * \text{nombre de pages dans une partition de taille moyenne} : \text{nombre total de pages à analyser} / 2$ Si le nombre total de pages à balayer $> 2 * \text{nombre de pages dans une partition de taille moyenne} : \text{moyenne des pages dans une partition}$	Table partitionnée avec un index clusterisé exploitable ; verrouillage APL	Balayage index en série
Balayage d'index avec hachage	Nombre de pages d'index à analyser au-dessus du niveau feuille + nombre de pages d'index de niveau feuille à analyser + (nombre de pages de données référencées dans les pages d'index de niveau feuille / nombre de processus de travail)	Toute table avec un index non clusterisé exploitable ou table DOL avec un index clusterisé	Balayage d'index en série
Balayage reposant sur des intervalles	Nombre de pages à lire dans les deux tables / nombre de processus de travail, plus coût des opérations de tri	Toute table d'une jointure utilisable dans une jointure par fusion	Jointure à boucle imbriquée, fusion en mode série

Sélection des méthodes d'accès parallèles

Pour une table donnée dans une requête, l'optimiseur évalue d'abord les index disponibles et les partitions afin de déterminer les méthodes d'accès utilisables pour balayer les données. Pour des requêtes impliquant une jointure, Adaptive Server prend en compte une jointure par fusion reposant sur des intervalles et une jointure par fusion en parallèle si la fonction de traitement en parallèle est activée. L'utilisation d'un balayage reposant sur des intervalles ne dépend pas du partitionnement de la table et ce type de balayage peut être réalisé avec des index clusterisés ou non clusterisés. L'optimiseur les prend en compte et, le plus souvent, les applique pour des tables qui ne disposent pas d'index exploitable sur la clé de jointure.

Le [tableau 8-2](#) présente les autres méthodes d'accès en parallèle que l'optimiseur peut envisager pour différentes combinaisons de tables et d'index. Les balayages de table par hachage ne sont utilisés que pour la table externe d'une requête, à moins que celle-ci n'utilise l'option parallel de l'optimiseur.

Tableau 8-2 : Détermination des méthodes d'accès applicables, reposant sur des partitions ou sur le hachage

	Aucun index exploitable	Index clusterisé exploitable	Index exploitable (non clusterisé ou clusterisé, sur une table DOL)
Table partitionnée	Balayage de partition	Balayage de partition d'un index clusterisé	Balayage d'index non clusterisé avec hachage
	Balayage de table avec hachage (si la table n'a pas d'index)	Balayage d'index en série	Balayage d'index en série
	Balayage de tables en série		
Table non partitionnée	Balayage de table avec hachage (si la table n'a pas d'index)	Balayage d'index en série	Balayage d'index non clusterisé avec hachage
	Balayage de tables en série		Balayage d'index en série

Dans un second temps, l'optimiseur peut exclure les méthodes d'accès en parallèle, en fonction du nombre de processus de travail disponibles pour la requête. Cette exclusion intervient lorsque l'optimiseur calcule le degré de parallélisation pour la requête dans sa totalité.

Pour un exemple, reportez-vous à la section « [Tables sans index partitionnées](#) », page 213.

Degré de parallélisation des requêtes parallèles

Le **degré de parallélisation** d'une requête est le nombre de processus de travail choisi par l'optimiseur pour exécuter la requête parallèle. Il dépend à la fois de la limite supérieure du degré de parallélisation de la requête et du niveau de parallélisation suggéré par l'optimiseur.

Le calcul du degré de parallélisation est déterminant pour les raisons suivantes :

- Le degré de parallélisation final influe directement sur les performances d'une requête car il indique le nombre de processus de travail qui doivent s'exécuter en parallèle.
- Lorsqu'il calcule le degré de parallélisation, l'optimiseur exclut les méthodes d'accès en parallèle qui requièrent un nombre de processus de travail supérieur aux limites définies par les paramètres de configuration, la commande `set` ou la clause `parallel`. Cela permet également de réduire le nombre de méthodes parmi lesquelles l'optimiseur doit choisir pour évaluer les coûts et donc de réduire d'autant la durée globale des opérations d'optimisation. L'exclusion de certaines méthodes d'accès est importante pour les jointures portant sur plusieurs tables, lorsque l'optimiseur doit examiner de nombreuses combinaisons d'ordres de jointure et de méthodes d'accès avant de sélectionner un plan d'exécution de requête définitif.

Limite supérieure

L'administrateur système définit la limite supérieure du degré de parallélisation à l'aide des paramètres de configuration du serveur. Les options au niveau d'une session et d'une requête permettent de restreindre davantage le degré de parallélisation. Ces limites définissent le nombre total de processus de travail pouvant être utilisés dans une requête parallèle et le nombre de processus de travail utilisables avec les méthodes d'accès par hachage.

L'optimiseur exclut d'emblée les méthodes d'accès en parallèle qui requièrent un nombre de processus excédant la limite supérieure définie pour la requête. (Si la limite supérieure du degré de parallélisation est 1, l'optimiseur ne sélectionne aucune méthode d'accès en parallèle.)

Pour plus d'informations sur les paramètres de configuration permettant de définir la limite supérieure du degré de parallélisation, reportez-vous à la section « [Paramètres de configuration pour le contrôle de la parallélisation](#) », page 168.

Degré optimisé

En principe, l'optimiseur peut utiliser autant de processus que l'autorise le degré maximum de parallélisation défini au niveau serveur, session ou requête. En pratique, le degré de parallélisation optimisé peut être inférieur à cette limite. Pour les balayages reposant sur des partitions, l'optimiseur définit le degré de parallélisation en fonction du nombre de partitions dans les tables de la requête et du nombre de processus de travail configurés.

Processus de travail pour les balayages reposant sur des partitions

En ce qui concerne les méthodes d'accès fondées sur des partitions, Adaptive Server requiert un processus de travail pour chaque partition d'une table. Si le nombre de partitions excède la valeur de `max parallel degree` ou la limite fixée au niveau de la session ou requête, l'optimiseur adopte une méthode d'accès en série ou fondée sur le hachage ; s'il peut utiliser une jointure par fusion, il la choisit en appliquant le paramètre `max parallel degree`.

Processus de travail pour les balayages avec hachage

En ce qui concerne les méthodes d'accès fondées sur un hachage, l'optimiseur ne calcule pas le degré de parallélisation optimal ; en revanche, il utilise le nombre de processus de travail défini par le paramètre `max scan parallel degree`. L'administrateur système peut attribuer une valeur optimale au paramètre `max scan parallel degree` pour la totalité du système Adaptive Server. En règle générale, la valeur de ce paramètre doit être comprise entre 2 et 3 car il ne faut pas plus de 2 à 3 processus de travail pour utiliser les E/S d'un device physique déterminé.

Processus de travail pour les balayages reposant sur des intervalles

Une jointure par fusion peut utiliser plusieurs processus de travail pour effectuer :

- le balayage qui sélectionne les lignes dans une table de travail pour toute jointure par fusion nécessitant un tri,
- le tri de la table de travail,
- la jointure par fusion et les jointures ultérieures dans la même étape.
- le balayage des intervalles sur les deux tables durant une jointure par fusion complète.

Utilisation pendant la création d'une table de travail

Si une jointure par fusion requiert une table de travail, la phase de la requête qui crée cette table peut utiliser une méthode d'accès en série ou en parallèle pour le balayage. Le nombre de processus de travail dont cette phase a besoin est déterminé par les méthodes normales de sélection. La requête qui sélectionne les lignes dans la table de travail peut être une requête sur une seule table ou une jointure contenant une jointure par fusion ou à boucle imbriquée ou une combinaison de jointures à boucle imbriquée et de jointure par fusion.

Tri parallèle pour les tables de travail des jointures par fusion

Un tri parallèle est appliqué lorsque le nombre de pages dans la table de travail à trier est égal à huit fois la valeur du paramètre de configuration `number of sort buffers`.

Pour plus d'informations sur le tri parallèle, reportez-vous au [Chapitre 9, « Tri parallèle »](#).

Nombre de threads de fusion

Pour la phase de fusion, le nombre de threads est égal à la valeur du paramètre `max parallel degree`, sauf si le nombre des valeurs distinctes est inférieur à ce dernier. Si le nombre de valeurs à fusionner est inférieur à `max parallel degree`, la tâche utilise un processus de travail par valeur, chaque processus fusionnant une valeur. Si les tables à fusionner ont des nombres différents de valeurs distinctes, le nombre le plus bas détermine le nombre de processus de travail utilisés. La formule est la suivante :

$$\text{Processus de travail} = \min(\text{max pll degree}, \min(t1_uniq_vals, t2_uniq_vals))$$

Lorsqu'il n'existe qu'une valeur distincte sur la colonne de jointure ou qu'il y a un argument de recherche d'égalité sur une colonne de jointure, la phase de fusion est exécutée en mode série. Ainsi, si une jointure par fusion s'applique à la requête ci-après, la fusion est réalisée en mode série :

```
select * from t1, t2  
where t1.c1 = t2.c1  
and t1.c1 = 10
```

Utilisation totale pour les jointures par fusion

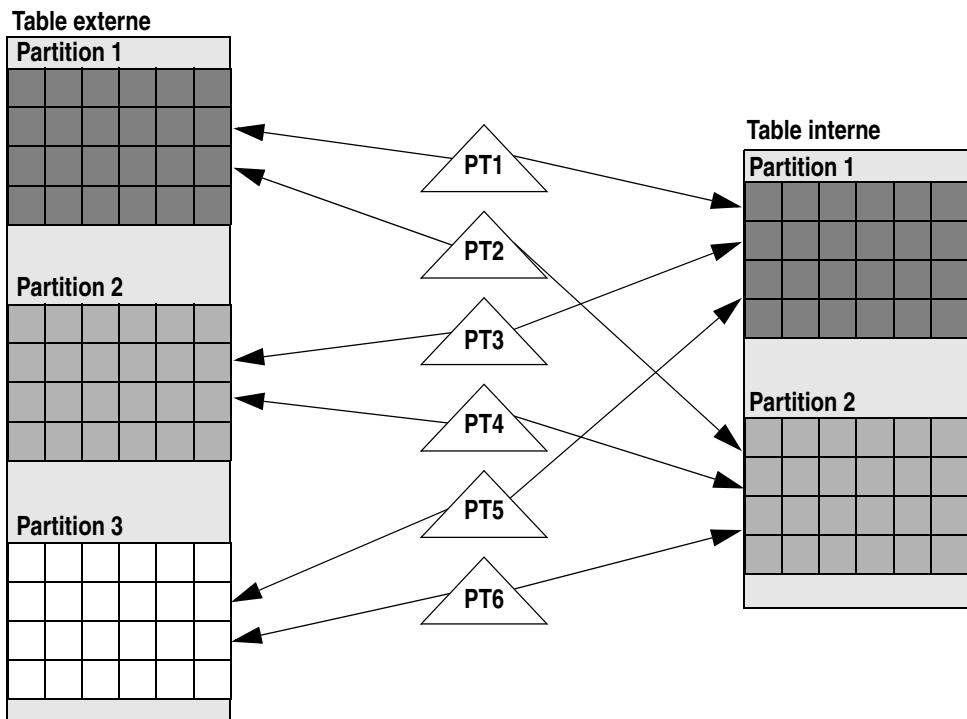
Une jointure par fusion peut utiliser un nombre de threads égal à la valeur indiquée par le paramètre max parallel degree pour la phase de fusion et autant pour chaque tri. Ainsi, une fusion qui effectue un tri parallèle utilisera jusqu'à 2*max parallel degree threads. Les processus de travail utilisés pour les tris sont libérés dès que ceux-ci sont terminés.

Jointures à boucles imbriquées

Si des tables individuelles se trouvent dans une jointure par boucles imbriquées, l'optimiseur calcule le degré de parallélisation en utilisant les règles décrites dans la section « [Degré optimisé](#) », page 208. Cependant, le degré de parallélisation de la requête de jointure est le *résultat* produit par les processus de travail qui accèdent à des tables individuelles dans la jointure. Les processus de travail alloués à une requête de jointure ont accès à toutes les tables de la jointure. L'utilisation des résultats qu'ils génèrent pour définir le degré de parallélisation d'une jointure garantit que le traitement est équitablement réparti entre les partitions et que la jointure ne renvoie pas de lignes dupliquées.

La [figure 8-6](#) illustre cette règle avec deux tables, la table externe disposant de trois partitions et la table interne de deux. Si l'optimiseur sélectionne des méthodes d'accès fondées sur des partitions, la requête requiert 6 processus de travail pour exécuter la jointure. Chacun des 6 processus de travail analyse une partition de la table externe et une partition de la table interne pour traiter la condition de jointure.

Figure 8-6 : Utilisation de processus de travail pour une jointure à boucle imbriquée



Dans la figure 8-6, si l'optimiseur sélectionne une méthode d'accès en série pour analyser la table interne, trois processus seulement sont nécessaires pour exécuter la jointure. Dans ce cas, chaque processus de travail analyse une partition de la table externe, puis tous analysent la table interne pour identifier des lignes correspondantes.

Par conséquent, pour une requête portant sur 2 tables et dont les degrés de balayage sont m et n , les degrés de parallélisation potentiels pour une jointure à boucle imbriquée sont les suivants :

- 1, si l'optimiseur accède en série aux deux tables.
- $m * 1$, si l'optimiseur accède à la première table à l'aide d'une méthode d'accès en parallèle (avec m processus de travail) et à la seconde table à l'aide d'une méthode d'accès en série.

- $n*1$, si l'optimiseur accède à la seconde table avec une méthode d'accès en parallèle (avec n processus de travail) et à la première table avec une méthode d'accès en série.
- $m*n$, si l'optimiseur accède aux deux tables avec des méthodes d'accès en parallèle.

Solutions de remplacement

Les balayages fondés sur une partition et utilisés sur deux tables d'une jointure sont peu fréquents, en raison du coût élevé des balayages répétés de la table interne. L'optimiseur peut également utiliser :

- une jointure par fusion,
- la méthode de reformatage d'index, si son coût est moindre,
- un balayage fondé sur des partitions, plus un balayage d'index avec hachage, lorsqu'une jointure accède à plus de 20 pages de données pour renvoyer les lignes.

Pour une illustration, reportez-vous à la [figure 7-7, page 166](#).

Calcul du degré de parallélisation pour des jointures à boucle imbriquée

Pour calculer le degré de parallélisation d'une jointure entre deux tables (et pour exclure les méthodes d'accès en parallèle qui requièrent trop de processus de travail), l'optimiseur applique les règles suivantes :

- 1 Il évalue l'utilité des méthodes d'accès et calcule des degrés de parallélisation pour la table externe de la jointure. Cette opération est la même que celle qui est effectuée pour les requêtes portant sur une seule table.

Reportez-vous à la section [« Degré optimisé », page 208](#).
- 2 Pour chacune des méthodes évaluées dans l'étape 1, il calcule le nombre de processus de travail disponibles pour la table interne de la jointure. La formule est la suivante :

Processus de travail restants = **max parallel degree** / processus de travail pour la table externe

- 3 Il définit la limite supérieure en fonction des processus de travail restants afin de sélectionner une méthode d'accès et de définir le degré de parallélisation de la table interne de la jointure.

Il répète ce processus pour tous les ordres de jointure et les méthodes d'accès possibles et applique la fonction coût aux jointures de chaque combinaison. Il sélectionne la combinaison d'ordres de jointure et de méthode d'accès la moins coûteuse ; la combinaison définitive permet de calculer le degré de parallélisation de la requête de jointure dans sa totalité.

Pour des exemples de ce processus, reportez-vous à la section « [Jointures à boucles imbriquées](#) », page 210.

Requêtes parallèles et jointures d'existence

Adaptive Server impose une restriction supplémentaire aux sous-requêtes traitées comme jointures de contrôle d'existence. Pour ces requêtes, seul le nombre de partitions dans la table externe détermine le degré de parallélisation. Simplement, il y a autant de processus de travail qu'il y a de partitions dans la table externe. Dans ce type de requête, les accès à la table interne s'effectuent toujours en série. Cette restriction ne s'applique pas aux sous-requêtes qui sont mises à plat et transformées en jointures standard.

Exemples

Les exemples de cette section illustrent l'incidence des limites du degré de parallélisation sur les types de requêtes suivants :

- les tables sans index partitionnées,
- les tables sans index non partitionnées,
- les tables avec un index clusterisé.

Tables sans index partitionnées

Supposons que max parallel degree et max scan parallel degree correspondent respectivement à 10 et 3 processus de travail.

Requêtes sur une seule table

Pour une requête sur une table dotée de 6 partitions et sans index non clusterisé exploitable, l'optimiseur évalue le coût des méthodes d'accès suivantes :

- un balayage d'une partition en parallèle avec 6 processus de travail ;
- un balayage de table en série avec un seul processus de travail.

Si `max parallel degree` indique 5 processus de travail, l'optimiseur ne prend en compte aucun balayage de partition en parallèle pour une table à 6 partitions.

Requêtes avec une jointure

La situation est différente, si la requête comprend une jointure. Supposons que `max parallel degree` ait pour valeur 10, que la requête comporte une jointure et qu'une table avec 6 partitions soit la table externe de la requête ; dans ce cas, l'optimiseur sélectionne l'une des méthodes suivantes :

- un balayage d'une partition avec 6 processus de travail ;
- un balayage de table avec hachage avec 3 processus de travail ;
- une jointure par fusion avec 10 processus de travail ;
- un balayage en série avec un seul processus de travail.

Si `max scan parallel degree` et `max scan parallel degree` ont respectivement pour valeur 5 et 3, l'optimiseur sélectionne l'une des méthodes suivantes :

- un balayage de table avec hachage avec 3 processus de travail ;
- une jointure par fusion avec 5 processus de travail ;
- un balayage en série avec un seul processus de travail.

Enfin, si `max parallel degree` et `max scan parallel degree` ont respectivement pour valeur 5 et 1, l'optimiseur prend uniquement en compte une jointure par fusion comme méthode d'accès en parallèle.

Tables sans index non partitionnées

Si la requête comporte une jointure, si `max scan parallel degree` a pour valeur 3 et si la table sans index non partitionnée est la table externe dans la requête, l'optimiseur utilise l'une des méthodes d'accès suivantes :

- un balayage de table avec hachage avec 3 processus de travail ;
- un balayage fondé sur des intervalles avec 10 processus de travail pour la jointure par fusion ;
- un balayage en série avec un seul processus de travail.

Si `max scan parallel degree` a pour valeur 1, l'optimiseur ne prend pas en compte le balayage avec hachage.

Pour plus d'informations sur la détermination du degré de parallélisation des requêtes, reportez-vous à la section « [Balayages effectués sur une seule table](#) », page 217.

Tables avec index clusterisé

Si la table a un index clusterisé, l'optimiseur étudie la possibilité d'utiliser les méthodes d'accès parallèle suivantes lorsqu'il s'agit d'une table APL :

- un balayage de partition en parallèle ou un balayage d'index clusterisé en parallèle si la table est partitionnée et si `max parallel degree` est défini au moins à 6 ;
- un balayage d'intervalles, en utilisant un nombre de processus de travail égal à la valeur de `max parallel degree` ;
- un balayage en série.

Si la table est verrouillée au niveau des pages de données seulement, l'optimiseur peut recourir aux méthodes suivantes :

- un balayage de partition en parallèle si la table est partitionnée et si `max parallel degree` est défini au moins à 6 ;
- un balayage d'intervalles, en utilisant un nombre de processus de travail égal à la valeur de `max parallel degree` ;
- un balayage en série.

Ajustements lors de l'exécution des processus de travail

Lorsque l'optimiseur a défini le degré de parallélisation pour la totalité de la requête, il arrive qu'Adaptive Server effectue des ajustements en fonction du nombre de processus de travail disponibles au moment de l'exécution. Si le nombre réel de processus disponibles est inférieur à celui qu'indique l'optimiseur, le degré de parallélisation est alors ramené à un niveau conforme au nombre de processus disponibles et aux méthodes d'accès définies dans le plan d'exécution définitif. La section « [Ajustements lors de l'exécution des processus de travail](#) », page 224 décrit les opérations permettant d'ajuster le degré de parallélisation au moment de l'exécution et explique quand elles doivent être effectuées.

Exemples de requête parallèle

Les sections suivantes décrivent et illustrent l'optimisation avec Adaptive Server des requêtes parallèles :

- [Balayages effectués sur une seule table](#)
- [Jointures sur plusieurs tables](#)
- [Sous-requêtes](#)
- [Requêtes nécessitant des tables de travail](#)
- [Requêtes union](#)
- [Requêtes avec agrégats](#)
- [Instructions select into](#)

Les commandes permettant d'insérer, de supprimer ou de modifier des données, ainsi que celles qui sont exécutées dans les curseurs ne sont jamais prises en considération lors de l'optimisation des requêtes parallèles.

Balayages effectués sur une seule table

La moindre optimisation d'une requête parallèle fait intervenir des requêtes qui accèdent à une base de données unique. Adaptive Server optimise ces requêtes en évaluant la table sous-jacente pour sélectionner les méthodes d'accès appropriées, puis en procédant à une évaluation du coût pour sélectionner le plan le plus avantageux.

La manière dont Adaptive Server optimise les requêtes portant sur une table unique permet de mieux comprendre les mécanismes des requêtes parallèles plus complexes. Bien que des requêtes comme les jointures de plusieurs tables et les sous-requêtes utilisent des méthodes d'optimisation supplémentaires, le processus pour accéder aux tables individuelles dans ces requêtes est identique.

L'exemple ci-après décrit des situations dans lesquelles l'optimiseur utilise des méthodes d'accès en parallèle pour des requêtes sur une table unique.

Balayage de partition de tables

Dans cet exemple de requête, l'optimiseur sélectionne un balayage de partition plutôt qu'un balayage en série. La configuration et la structure de la table sont définies comme suit :

Valeurs du paramètre de configuration	
Paramètre	Valeur
max parallel degree	10 processus de travail
max scan parallel degree	2 processus de travail

Structure de la table			
Nom de table	Index utiles	Nombre de partitions	Nombre de pages
authors	Aucun	5	Partition 1 : 50 pages Partition 2 : 70 pages Partition 3 : 90 pages Partition 4 : 80 pages Partition 5 : 10 pages

La requête est la suivante :

```
select *
  from authors
 where au_lname < "L"
```

L'optimiseur suit la logique du [tableau 8-2, page 206](#), pour sélectionner l'une des deux méthodes suivantes :

- balayage de partition,
- balayage de tables en série.

L'optimiseur n'utilise pas de balayage de table avec hachage car la répartition des pages dans les partitions est correcte et la limite supérieure du degré de parallélisation (10) est suffisamment élevée pour permettre un balayage de partition.

L'optimiseur calcule les ressources nécessaires pour chaque méthode d'accès de la manière suivante :

Coût du balayage de partition de la table = nbre de pages dans une grande partition = 90 pages

Coût d'un balayage de table en série = nbre de pages dans la table = 300 pages

L'optimiseur va effectuer le balayage d'une partition de page avec 90 E/S physiques et logiques. La table ayant 5 partitions, l'optimiseur choisit d'utiliser 5 processus de travail. Les résultats définitifs fournis par showplan pour cette requête sont :

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 5 worker
processes.
STEP 1
The type of query is SELECT.
Executed in parallel by coordinating process and 5
worker processes.
FROM TABLE
authors
Nested iteration.
Table Scan.
Forward scan.
Positioning at start of table.
Executed in parallel with a 5-way partition scan.
Using I/O Size 16 Kbytes for data pages.
With LRU Buffer Replacement Strategy for data pages.
Parallel network buffer merge.
```

Jointures sur plusieurs tables

L'optimiseur prend en compte le meilleur ordre de jointures dans toutes les combinaisons de tables et de méthodes d'accès pour les optimiser. Il utilise une autre technique de sélection des méthodes d'accès s'il s'agit de tables internes et externes et calcule différemment le degré de parallélisation dans le cas d'une requête de jointure dans sa totalité.

De même que dans le traitement en série, il évalue diverses solutions d'accès à une table spécifique. Il définit les coûts d'une exécution en parallèle par rapport à d'autres facteurs ayant une incidence sur les requêtes de jointure, tels que : un index clusterisé, les jointures à boucle imbriquée ou par fusion, la possibilité de redéfinir les index des tables internes, l'ordre des jointures et la stratégie d'E/S et de cache. L'analyse ci-après est consacrée uniquement à la comparaison entre la méthode d'accès en parallèle et celle en série.

Optimisation des jointures parallèles et ordre des jointures

Dans l'exemple suivant, l'optimiseur élabore un plan pour une requête susceptible d'être exécutée en parallèle. La configuration et la structure de la table sont définies comme suit :

Valeurs du paramètre de configuration	
Paramètre	Valeur
max parallel degree	15 processus de travail
max scan parallel degree	3 processus de travail

Structure de la table			
Nom de table	Nombre de partitions	Nombre de pages	Nombre de lignes
publishers	1 (non partitionnée)	1 000	80 000
titles	10	10 000 (distribuées uniformément entre les partitions)	800 000

La requête suivante comporte une simple jointure entre ces deux tables :

```
select *
  from publishers, titles
 where publishers.pub_id = titles.pub_id
```

En principe, l'optimiseur prend en compte les coûts de toutes les combinaisons possibles :

- titles en tant que table externe, publishers comme table interne et l'accès en parallèle à titles
- titles en tant que table externe, publishers comme table interne et l'accès en série à titles
- publishers en tant que table externe, titles comme table interne et l'accès en parallèle à titles
- publishers en tant que table externe, titles comme table interne et l'accès en série à titles
- publishers en tant que table externe, titles comme table interne et l'accès en parallèle à publishers

Par exemple, le coût d'un ordre de jointure où l'accès à la table externe titles s'effectue en parallèle est calculé de la manière suivante :

Si publishers est la table externe, le calcul sera le suivant :

Cela étant, d'autres facteurs que l'adaptation d'une table à un accès en parallèle jouent un rôle plus déterminant pour le choix de l'ordre de jointure.

Cas de figure A : index clusterisé sur publishers

Un index clusterisé exploitable est souvent le facteur le plus important pour la création d'un plan d'exécution de requêtes. Si publishers dispose d'un index clusterisé sur pub_id et si titles n'en contient pas, l'optimiseur peut choisir la table indexée (publishers) comme table interne. Avec ce type d'ordre de jointures, il suffit de quelques lectures pour trouver des lignes dans la table interne.

Si publishers sert de table interne, l'optimiseur évalue les méthodes d'accès adaptées à chaque table. Si titles est la table externe, il prévoit :

- un balayage de partition en parallèle (le coût est le nombre de pages dans la partition la plus large) ;
- un balayage de table en série (le coût est le nombre de pages dans la table).

Si publishers est la table interne, il envisage uniquement un balayage d'index clusterisé en série.

Il envisage également une jointure par fusion, en triant la table dans l'ordre adéquat à partir des titres sur `titles`, soit à droite soit à gauche.

Le coût final de la requête est obtenu en multipliant le coût d'accès en parallèle à `titles` par le nombre d'accès de l'index clusterisé sur `publishers`.

Cas de figure B : index clusterisé dans `titles`

Si `titles` dispose d'un index clusterisé sur `pub_id` et si `publishers` n'en contient pas, l'optimiseur choisit `titles` comme table interne de la requête.

Avec cet ordre de jointures déterminé, l'optimiseur évalue les méthodes d'accès adaptées à chaque table. Si `publishers` est la table externe, il prévoit :

- un balayage avec hachage (le coût initial est le même que dans un balayage en série).

Si `titles` est la table interne, il envisage uniquement un balayage d'index clusterisés en série.

Dans cet exemple, l'optimiseur préfère l'exécution en parallèle à celle en série de `publishers`. Les coûts sont identiques pour les deux types de balayage ; cependant, le temps de traitement est réduit d'un tiers dans le cas d'un balayage en hachage puisque les processus de travail peuvent lire l'index clusterisé de la table interne simultanément.

Cas de figure C : aucune des deux tables ne comporte d'index exploitable

Dans ce cas, il convient de recourir à la jointure par fusion. Si les jointures par fusion sont désactivées, la taille de la table et l'espace de cache disponibles comptent davantage que le mode d'accès en parallèle.

Les avantages d'une table plus réduite, telle la table interne, ont plus d'impact que les caractéristiques des diverses méthodes d'accès.

L'optimiseur choisit `publishers` comme table interne, en raison de sa taille qui lui permet d'être contenue dans le cache et de n'avoir besoin que d'une lecture, ce qui réduit les coûts des E/S physiques.

L'optimiseur évalue alors les méthodes d'accès adaptées à chaque table. Si `titles` est la table externe, il prévoit :

- un balayage de partition en parallèle (le coût est le nombre de pages dans la partition la plus large) ;
- un balayage de table en série (le coût est le nombre de pages dans la table).

Si `publishers` est la table interne, il envisage uniquement un balayage de table en série chargé dans le cache.

L'optimiseur choisit d'accéder à titles en parallèle, ce qui lui permet de réduire le coût de la requête d'un facteur de 10.

En cas d'absence d'index exploitable dans les deux types de tables, l'optimiseur opte pour la redéfinition d'index, en créant une table provisoire et un index clusterisé au lieu d'analyser indéfiniment la table interne.

Sous-requêtes

Lorsqu'une requête contient une sous-requête, Adaptive Server utilise différentes méthodes d'accès pour en réduire le coût de traitement.

L'optimisation parallèle dépend du type de la sous-requête et des méthodes d'accès :

- Sous-requêtes matérialisées : les méthodes en parallèle ne sont pas envisagées pour la phase de matérialisation.
- Sous-requêtes mises à plat : l'optimisation en parallèle est envisagée uniquement lorsque la sous-requête est transformée en jointure standard. Elle n'est pas utilisée pour les jointures d'existence ou autres stratégies de mise à plat.
- Sous-requêtes imbriquées : l'optimiseur n'envisage une méthode d'accès en parallèle que pour le bloc le moins imbriqué dans une requête comportant une sous-requête ; les requêtes internes et imbriquées sont toujours exécutées en série. Bien que l'optimiseur n'utilise des méthodes d'accès en parallèle que pour le bloc de requêtes le plus excentré dans une sous-requête, tous les processus de travail qui accèdent au bloc externe accèdent également aux tables internes figurant dans les sous-requêtes imbriquées.

Chaque processus de travail accède en série au bloc de requêtes interne imbriqué. Même si la sous-requête est exécutée une seule fois pour chaque ligne de la table externe, chaque processus n'effectue qu'un cinquième des traitements. Le résultat de showplan relatif à la sous-requête indique que la requête imbriquée est exécutée par 5 processus de travail (« Executed by 5 worker processes »), puisque chaque processus utilisé dans le bloc de requêtes externes analyse la table sélectionnée dans le bloc de requêtes internes.

Chaque processus de travail conserve dans un cache séparé les résultats de la sous-requête, ce qui permet d'exécuter cette dernière un peu plus souvent que dans le cadre d'un traitement en série.

Requêtes nécessitant des tables de travail

Les requêtes parallèles qui nécessitent des tables de travail créent celles-ci avec des partitions et les remplissent en parallèle. Lorsque les requêtes requièrent un tri, le gestionnaire de tri détermine s'il convient d'utiliser le mode série ou parallèle.

Pour plus d'informations sur le tri parallèle, reportez-vous au [Chapitre 9, « Tri parallèle »](#).

Requêtes union

L'optimiseur utilise les méthodes d'accès en parallèle pour chaque partie d'une requête union. Chaque instruction select dans une union est optimisée séparément, de sorte qu'une requête peut utiliser un plan parallèle, une autre un plan série, une troisième un plan parallèle avec un nombre différent de processus de travail. Lorsqu'une requête union requiert une table de travail, celle-ci peut également être partitionnée et remplie en parallèle par les processus de travail.

Lorsqu'une requête union ne doit pas renvoyer de lignes dupliquées, il est préférable d'effectuer un tri parallèle dans la table de travail interne pour supprimer les valeurs en double.

Pour plus d'informations sur le tri parallèle, reportez-vous au [Chapitre 9, « Tri parallèle »](#).

Requêtes avec agrégats

Adaptive Server utilise également des méthodes d'accès en parallèle pour les requêtes qui renvoient des résultats agrégés. Dans les requêtes qui utilisent la clause group by pour renvoyer des résultats agrégés groupés, Adaptive Server crée plusieurs tables de travail dotées d'index clusterisés (une table de travail pour chaque processus de travail exécutant la requête). Chaque processus de travail stocke des résultats agrégés partiels dans sa table de travail. Lorsque les processus ont fini de calculer leurs résultats partiels, ils les regroupent dans une table commune. Une fois ce regroupement terminé, la table de travail commune contient l'ensemble des résultats agrégés de la requête.

Instructions `select into`

`select into` crée une nouvelle table pour y stocker le jeu de résultats de la requête. Pour optimiser la partie requête d'une commande `select into`, Adaptive Server procède de la même façon qu'avec une requête standard, en prenant en considération les méthodes d'accès en série et en parallèle. Une instruction `select into` exécutée en parallèle :

- 1 crée la nouvelle table en utilisant les colonnes définies dans l'instruction `select into` ;
- 2 crée n partitions dans la nouvelle table, n étant le degré de parallélisation que l'optimiseur applique à la totalité de la requête ;
- 3 remplit la nouvelle table avec les résultats de la requête, à l'aide de n processus de travail ;
- 4 annule les partitions de la nouvelle table.

L'exécution d'une instruction `select into` en parallèle requiert des étapes supplémentaires, par rapport au mode en série. L'exécution d'une instruction `select into` en parallèle est effectuée avec 4 transactions discontinues, au lieu des 2 transactions en mode série. Pour savoir dans quelle mesure cette modification influence le processus de restauration d'une base de données, reportez-vous aux explications relatives à `select` dans le *Manuel de référence d'Adaptive Server*.

Ajustements lors de l'exécution des processus de travail

Le résultat de `showplan` décrit le plan optimisé relatif à une requête déterminée. Un plan d'exécution de requête optimisé indique les méthodes d'accès et le degré de parallélisation sélectionnés par l'optimiseur lors de la compilation de la requête. Au moment de l'exécution, il est possible que les processus de travail soient moins nombreux que ceux qui sont prévus dans le plan d'exécution. C'est notamment le cas lorsque :

- le nombre de processus de travail disponibles pour le plan d'exécution de requête est insuffisant ;
- les limites au niveau serveur ou session ont été réduites après la compilation de la requête. Ce cas peut se présenter lorsque les requêtes sont exécutées depuis les procédures stockées.

Dans ces conditions, Adaptive Server crée éventuellement un plan d'exécution des requêtes pour compenser le manque de processus. Cette création d'un **plan d'exécution des requêtes ajusté** a lieu pendant l'exécution. Il peut utiliser moins de processus de travail que le plan d'exécution de requête optimisé et prévoir même une méthode d'accès en série pour une ou plusieurs tables.

Ajustement d'un plan d'exécution de requête avec Adaptive Server

Adaptive Server utilise deux règles élémentaires pour réduire le nombre de processus de travail requis dans un plan d'exécution de requête ajusté :

- 1 Si le plan d'exécution de requête optimisé prévoit une méthode d'accès reposant sur une partition et s'il n'y a pas assez de processus pour analyser chaque partition, le plan ajusté utilise une méthode d'accès en série.
- 2 Si le plan d'exécution de requête optimisé prévoit une méthode d'accès fondée sur un hachage et s'il n'y a pas assez de processus pour couvrir le degré de parallélisation optimisé, le plan ajusté ramène le degré de parallélisation à un niveau conforme avec le nombre de processus de travail disponibles.

Dans le premier cas, supposons qu'un plan d'exécution de requête optimisé prévoie l'analyse des 5 partitions d'une table à l'aide d'un balayage de table reposant sur une partition. Si 4 processus de travail seulement sont disponibles au moment où la requête est exécutée, Adaptive Server génère un plan d'exécution de requête ajusté permettant d'accéder à la table en série avec un seul processus.

Dans le deuxième cas, si le plan d'exécution de requête optimisé prévoit l'analyse de la table avec une méthode d'accès fondée sur un hachage et 5 processus de travail, le plan d'exécution de requête ajusté aura toujours recours à un accès avec hachage mais il n'utilisera pas plus de 4 processus de travail.

Incidences des ajustements lors de l'exécution

Bien que les plans d'exécution de requête optimisés surpassent généralement les plans d'exécution de requête ajustés, la différence en termes de performances n'est pas toujours significative. Les incidences déterminantes sur les performances sont fonction du nombre de processus de travail qu'Adaptive Server utilise dans le cadre du plan ajusté et de l'éventuel remplacement d'une méthode d'accès en parallèle par une méthode en série. Naturellement, l'utilisation par Adaptive Server d'une méthode d'accès en série au lieu d'une méthode en parallèle a les conséquences les plus négatives.

Lorsqu'il crée des plans d'exécution de requête ajustés, Adaptive Server ne modifie pas l'ordre de jointure ; c'est pourquoi ces plans peuvent considérablement réduire les performances des requêtes de jointure comportant plusieurs tables. Lorsqu'un plan d'exécution de requête ajusté est exécuté en série, la requête sera vraisemblablement exécutée plus lentement qu'une jointure en série optimisée. Ce cas peut se présenter lorsque l'ordre de jointure en parallèle optimisé pour une requête est différent de l'ordre de jointure en série optimisé.

Identification et gestion des ajustements lors de l'exécution

Adaptive Server propose deux mécanismes pour vous aider à analyser les ajustements des plans de requête effectués au moment de l'exécution.

- `set process_limit_action` permet d'annuler les batch ou les procédures lors des ajustements ou de l'impression de messages d'avertissement.
- `showplan` imprime un plan d'exécution de requête ajusté lorsque les ajustements sont effectués et il est actif.

Utilisation de `set process_limit_action`

L'option `process_limit_action` associée à la commande `set` vous permet d'utiliser les plans d'exécution des requêtes ajustés au niveau d'une session ou d'une procédure stockée. Lorsque vous définissez `process_limit_action` sur « `abort` », Adaptive Server enregistre l'erreur 11015 et annule la requête, dans le cas où un plan d'exécution des requêtes ajusté est nécessaire. Si vous définissez l'option sur « `warning` », Adaptive Server enregistre l'erreur 11014 mais exécute quand même la requête.

La commande suivante annule le batch lorsqu'une requête est ajustée au moment de l'exécution :

```
set process_limit_action abort
```

En examinant les messages d'erreur 11014 et 11015 dans le journal des erreurs, vous pouvez savoir si Adaptive Server utilise des plans d'exécution de requête ajustés au lieu des plans optimisés. Pour supprimer les restrictions et permettre les ajustements au moment de l'exécution, utilisez :

```
set process_limit_action quiet
```

Pour plus d'informations sur `process_limit_action`, reportez-vous aux explications relatives à la commande `set` dans le *Manuel de référence d'Adaptive Server*.

Utilisation de `showplan`

Lorsque vous utilisez `showplan`, Adaptive Server affiche le plan optimisé d'une requête avant de l'exécuter. Si ce plan prévoit un traitement en parallèle et si un ajustement est effectué, `showplan` affiche le message ci-dessous, ainsi que le plan d'exécution de requête ajusté :

```
AN ADJUSTED QUERY PLAN WILL BE USED FOR STATEMENT 1  
BECAUSE NOT ENOUGH WORKER PROCESSES ARE AVAILABLE AT  
THIS TIME.
```

Adaptive Server n'exécute pas de requête lorsque l'option `set noexec` est active, de sorte qu'il n'affiche pas non plus les plans au moment de l'exécution.

Réduction de la probabilité d'ajustements lors de l'exécution

Pour réduire le nombre d'ajustements au moment de l'exécution, vous devez augmenter le nombre de processus de travail disponibles pour les requêtes parallèles. Pour ce faire, vous devez ajouter des processus ou bien restreindre ou supprimer l'exécution en parallèle pour les requêtes non prioritaires, comme indiqué ci-dessous :

- Utilisez `set parallel_degree` et/ou `set scan_parallel_degree` pour définir les limites du degré de parallélisation au niveau de la session ; ou
- Utilisez les clauses `parallel 1` et `parallel N` au niveau de la requête pour limiter l'utilisation d'instructions individuelles par les processus de travail.

Pour restreindre le nombre d'ajustements effectués au moment de l'exécution pour les procédures système, veillez à recompiler les procédures après avoir modifié le degré de parallélisation au niveau du serveur ou de la session. Pour de plus amples informations, reportez-vous aux explications relatives à `sp_recompile` dans le *Manuel de référence d'Adaptive Server*.

Vérification des ajustements avec `sp_sysmon`

La procédure système `sp_sysmon` indique combien de fois une demande de processus de travail a été rejetée en raison d'un nombre insuffisant de processus et combien de fois le nombre de processus de travail prévus pour une requête a été réduit. Les sections ci-après fournissent les informations suivantes :

- La section « [Gestion des processus de travail](#) », page 231 du guide *Performances et optimisation : contrôle et analyse* décrit les résultats provenant des demandes de processus de travail qui sont lancées et rejetées, ainsi que l'aboutissement et l'échec des requêtes d'allocation de mémoire pour des processus de travail.
- La section « [Gestion des requêtes parallèles](#) », page 234 du guide *Performances et optimisation : contrôle et analyse* décrit le résultat de `sp_sysmon`, notamment le nombre d'ajustements au moment de l'exécution et le nombre de verrouillages de requêtes parallèles.

Si le problème semble dû à un nombre de processus insuffisant dans le groupe, comparez le nombre de processus de travail utilisés avec le nombre de processus configurés. Si le nombre maximal de processus utilisés est égal à la valeur de `number of worker processes` et si les rejets des demandes de processus représentent plus de 80 pour cent, augmentez la valeur de `number of worker processes` et relancez `sp_sysmon`. Si le nombre maximal de processus utilisés est inférieur à la valeur de `number of worker processes` et si le nombre de thread de production rejetées est de 0 pour cent, réduisez la valeur de `number of worker processes` pour libérer de la mémoire.

Identification des problèmes liés aux performances des requêtes parallèles

Les sections suivantes comportent des instructions pour détecter et résoudre les problèmes liés aux requêtes parallèles. Elles concernent deux types de situations :

- la requête est exécutée en série, alors que vous souhaitez l'exécuter en parallèle ;
- la requête est exécutée en parallèle mais vous n'obtenez pas de bons résultats.

Impossibilité d'exécuter la requête en parallèle

La requête ne s'exécute pas en parallèle comme prévu, parce que :

- Le paramètre de configuration max parallel degree a pour valeur 1 ou le paramètre set parallel_degree au niveau de la session a pour valeur 1, ce qui exclut tout accès en parallèle.
- Le paramètre de configuration max scan parallel degree a pour valeur 1 ou le paramètre set scan_parallel_degree au niveau de la session a pour valeur 1, ce qui exclut les accès en parallèle fondés sur un hachage.
- Il n'y a pas suffisamment de threads de production au moment de l'exécution. Vérifiez les ajustements lors de l'exécution à l'aide des outils décrits dans la section « [Ajustements lors de l'exécution des processus de travail](#) », page 216.
- La portée du balayage est inférieure à 20 pages de données. Vous pouvez y remédier avec la clause (parallel).
- Le plan prévoit un balayage de table mais :
 - la table n'a pas d'index,
 - la table n'est pas partitionnée,
 - la partition est mal répartie ou
 - la table est un segment de mémoire mais il ne s'agit pas de la table externe d'une jointure.

Dans les deux derniers cas, il suffit d'utiliser la clause parallel.

- Le plan prévoit un balayage d'index clustuterisé mais :
 - la table n'est pas partitionnée ou
 - la partition est mal répartie. Vous pouvez y remédier avec la clause (parallel).
- Le plan prévoit un balayage d'index non clusterisé et l'index sélectionné couvre les colonnes requises.
- La table est une table temporaire ou une table système.
- La table est la table interne d'une jointure externe.
- Une limite a été fixée par l'intermédiaire du gestionnaire de ressources et tous les plans d'exécution en parallèle dépassent cette limite, en termes de charge globale.
- Il s'agit d'un type de requête non parallélisé, tel qu'une commande insert, update ou delete, d'une requête imbriquée (excepté la plus excentrée) ou d'un curseur.

Performances en parallèle peu satisfaisantes

Il peut y avoir plusieurs explications :

- Il y a trop de partitions pour les devices physiques sous-jacents.
- Il y a trop de devices par contrôleur.
- La clause (parallel) n'est pas utilisée de manière appropriée.
- La valeur de max scan parallel degree est trop élevée ; elle doit être comprise entre 2 et 3.

Assistance technique

Si les instructions ci-dessus ne suffisent pas pour résoudre le problème, vous devez fournir les informations suivantes au personnel du Support Technique de Sybase :

- Le type de table et d'index (les instructions create table, alter table...partition et create index sont très utiles). Décrivez le résultat de sp_help si les commandes create et alter ne sont pas disponibles.
- La requête.

- Le résultat de la requête obtenu à l'aide des commandes :
 - dbcc traceon (3604, 302, 310)
 - set showplan on
 - set noexec on
- Les résultats de statistics io pour la requête.

Limites d'utilisation des ressources dans les requêtes parallèles

Lorsque Adaptive Server est configuré pour un traitement en parallèle, l'identification des limites de coût des E/S peut s'avérer moins précise avec des tables partitionnées qu'avec des tables non partitionnées.

Lorsque vous lancez une requête sur une table partitionnée, le traitement est réparti entre les partitions. Par exemple, si la table comporte 3 partitions, le traitement de la requête sera réparti entre 3 processus de travail. Si l'utilisateur a limité les ressources d'E/S en définissant une limite supérieure de 6000, l'optimiseur attribue une limite de 2000 à chaque processus de travail.

Cependant, rien ne garantit que deux threads traitent une charge de travail identique ; c'est pourquoi le processeur parallèle ne peut pas répartir avec précision la charge entre les processus de travail. Il est possible qu'un message d'erreur s'affiche pour vous signaler que vous avez dépassé les limites d'E/S, alors que les résultats de showplan ou de statistics io n'indiquent aucun dépassement. Inversement, une partition peut dépasser légèrement la limite sans que la restriction ne s'applique.

Pour plus d'informations sur les limites d'utilisation des ressources, reportez-vous au *Guide d'administration système*.

Tri parallèle

Ce chapitre explique comment configurer le serveur pour améliorer les performances des commandes permettant d'effectuer des tris parallèles.

Le tri des données est un processus inhérent aux systèmes de gestion de base de données. Il permet de créer des index et de traiter des requêtes complexes. Le gestionnaire de tri parallèle d'Adaptive Server fournit une méthode en parallèle performante pour trier les lignes de données. Toutes les commandes Transact-SQL qui requièrent un tri interne peuvent utiliser le tri parallèle.

Ce chapitre décrit également les mécanismes du tri parallèle, ainsi que les facteurs qui influent sur ses performances. Il est indispensable de bien comprendre ces mécanismes pour obtenir des performances optimales et éviter que les ressources requises par le tri parallèle n'interfèrent avec d'autres besoins en ressources.

Sujet	Page
Commandes pouvant bénéficier du tri parallèle	233
Présentation des conditions et des ressources	234
Présentation de la méthode de tri parallèle	236
Configuration des ressources pour le tri parallèle	240
Considérations de reprise	255
Outils permettant de contrôler et d'optimiser les tris	256
Utilisation de sp_sysmon pour optimiser la création d'index	261

Commandes pouvant bénéficier du tri parallèle

Toutes les commandes Transact-SQL qui requièrent un tri de lignes de données peuvent employer les techniques de tri parallèle.

Ces commandes sont les suivantes :

- commandes `create index` et `alter table...add constraint` permettant de générer des index, unique et primary key,
- requêtes utilisant la clause `order by`,

- requêtes utilisant distinct,
- requêtes effectuant des jointures par fusion qui nécessitent des tris,
- requêtes utilisant union (sauf union all),
- requêtes utilisant la méthode de **reformatage d'index**.

En outre, tous les curseurs qui utilisent les commandes ci-dessus peuvent faire l'objet d'un tri parallèle.

Présentation des conditions et des ressources

A l'instar du traitement des requêtes en parallèle, le tri parallèle requiert des ressources importantes. Le temps de réponse nécessaire pour créer un index ou trier les résultats d'une requête s'en trouve amélioré, mais la charge de travail du serveur est beaucoup plus lourde, en raison des overhead.

Le gestionnaire de tri d'Adaptive Server permet de savoir si les ressources requises pour effectuer un tri sont disponibles et s'il faut sélectionner un tri en série ou en parallèle, compte tenu de la taille de la table et de certains autres facteurs. Pour un tri parallèle, les conditions suivantes doivent être réunies :

- L'option de base de données `select into/bulk copy/pllsort` doit être associée au paramètre `true`, avec `sp_dboption` dans la base de données de destination.
- S'il s'agit d'index, l'option doit être activée dans la base de données où se trouve la table. Pour créer un index clusterisé sur une table partitionnée, cette option doit être activée, faute de quoi le tri n'aboutira pas. Pour créer d'autres index, il est possible d'utiliser des tris en série lorsque les tris parallèles ne sont pas réalisables.
- Pour trier les tables de travail, cette option doit être activée dans la base `tempdb`. Les tris en série sont effectués lorsque le mode parallèle n'est pas autorisé.

- Les tris parallèles doivent comporter un nombre minimum de processus de travail. Ce nombre dépend du nombre de partitions dans la table et/ou du nombre de devices sur le segment cible. Le degré de parallélisation au niveau du serveur ou de la session doit être suffisamment élevé pour que le tri puisse utiliser le nombre de processus de travail minimum requis par une opération de tri parallèle. Les tris des index clusterisés sur des tables partitionnées doivent être effectués en parallèle ; des tris en série peuvent être utilisés lorsque le nombre de processus de travail disponibles est insuffisant. « [Processus de travail requis pour le tri parallèle](#) », [page 240](#), et « [Processus de travail requis pour le tri de la requête select](#) », [page 244](#).
- Pour les commandes `select` qui requièrent un tri et en cas de création d'index non clusterisé, la taille de la table doit être au moins huit fois la taille des buffers de tri (la valeur du paramètre de configuration `number of sort buffers`), faute de quoi le tri est effectué en série. Cette condition évite à Adaptive Server d'effectuer un tri parallèle sur de petites tables, ce qui ne donnerait aucune indication intéressante sur les améliorations de performance. Cette règle ne s'applique pas à la création d'index clusterisés sur les tables partitionnées, qui exigent toujours un tri parallèle.

Reportez-vous à la section « [Instructions pour configurer les buffers de tri](#) », [page 247](#).

- Pour des commandes `create index`, la valeur du paramètre de configuration `number of sort buffers` doit être au moins égale au nombre de processus de travail disponibles pour un tri parallèle.

Reportez-vous à la section « [Instructions pour configurer les buffers de tri](#) », [page 247](#).

Remarque Vous ne pouvez pas utiliser la commande `dump transaction` après avoir créé des index en utilisant un tri parallèle. Vous devez sauvegarder la base de données. Les commandes `create index` en série peuvent être récupérées, mais uniquement par une relance de la commande d'indexation, d'où un temps de reprise qui risque d'augmenter considérablement. Il est recommandé d'effectuer des sauvegardes de bases de données après la création d'index en série pour accroître la vitesse de la reprise, même si elles ne sont pas nécessaires pour utiliser la commande `dump transaction`.

Présentation de la méthode de tri parallèle

A l'instar de l'optimiseur d'Adaptive Server, son gestionnaire de tri parallèle analyse les processus de travail disponibles, la table d'entrée et les autres ressources pour définir le nombre de processus de travail nécessaires au tri.

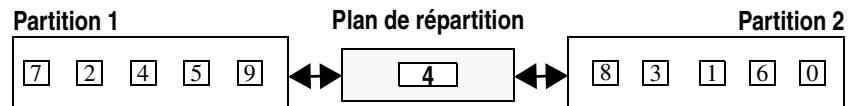
Lorsque le nombre de ces derniers est défini, Adaptive Server exécute le tri parallèle. Le processus du tri parallèle se déroule de la même façon pour les commandes `create index` et pour les requêtes qui requièrent des tris. Adaptive Server effectue les opérations suivantes :

- 1 Création d'un plan de répartition. Pour une jointure par fusion avec des statistiques sur une colonne de jointure, les données d'histogramme servent à l'établissement de ce plan. Dans les autres cas, la table d'entrée est analysée.
 - 2 Lecture des données de la table et partitionnement dynamique des clés dans un ensemble de buffers, conformément au plan de répartition.
 - 3 Tri de chaque intervalle de valeurs clés et création de sous-index.
 - 4 Regroupement des sous-index triés sous la forme d'un résultat final.
- Chacune de ces étapes est décrite dans les sections ci-après.

La [figure 9-1](#) illustre le tri parallèle d'une table dotée de deux partitions et de deux devices physiques sur son segment.

Figure 9-1 : Méthode de tri parallèle

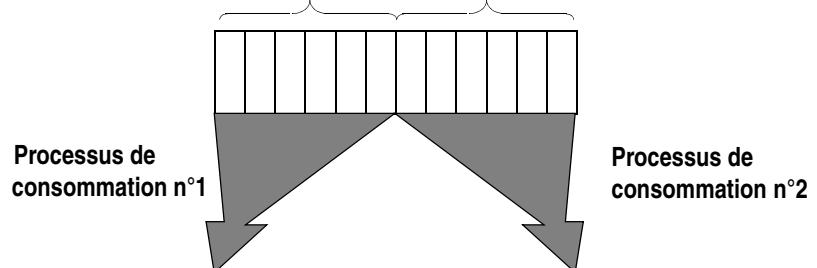
Etape n°1. Analyse des données et création du plan de répartition.



Etape n°2. Partitionnement des données en intervalles discontinus.



Etape n°3. Tri de chaque intervalle et création d'index.



Etape n°4. Fusion des données triées.

Résultat de la fusion ou index

0 1 2 3 4 5 6 7 8 9

Création d'un plan de répartition

En premier lieu, Adaptive Server crée un plan de répartition. Si le tri est réalisé dans le cadre d'une jointure par fusion et s'il existe des statistiques sur les colonnes de jointure, les histogrammes sont utilisés pour générer ce plan. Pour les autres tris, Adaptive Server sélectionne et trie un échantillon de données qu'il extrait de la table d'entrée. Ces informations, appelées plan de répartition, sont utilisées au cours de la deuxième étape de tri pour diviser les données d'entrée en intervalles de tailles égales.

Ce plan contient la valeur de la clé la plus élevée dans chaque intervalle, sauf le dernier de la table. Dans la [figure 9-1](#), le plan de répartition indique que toutes les valeurs inférieures ou égales à 4 sont affectées au premier intervalle et celles supérieures à 4 au deuxième intervalle.

Partitionnement dynamique des intervalles

Après avoir généré le plan de répartition, Adaptive Server utilise deux types de processus de travail pour traiter les différents aspects du tri. Ces processus de travail sont appelés **processus de production** et **processus de consommation** :

- Les processus de production lisent des données dans la table d'entrée et utilisent le plan de répartition pour définir l'intervalle auquel appartient chaque valeur clé. Ils répartissent les données en les copiant dans des buffers de tri affectés à l'intervalle approprié.
- Chaque processus de consommation lit les données dans un intervalle de buffers et les trie dans des sous-index, comme décrit dans la section « [Tri des intervalles](#) », page 239.

Dans la [figure 9-1](#), deux processus de production lisent des données dans la table d'entrée. Chaque processus analyse une partition de la table et répartit les données dans des intervalles, à l'aide du plan de répartition. Par exemple, le premier processus de production lit les valeurs 7, 2, 4, 5 et 9. Conformément au plan de répartition, il transmet les valeurs 2 et 4 au premier processus de consommation et les valeurs 7, 5 et 9 au deuxième processus de consommation.

Tri des intervalles

Chaque intervalle partitionné a un processus de consommation dédié qui trie les données, indépendamment des autres intervalles. Selon la taille de la table et le nombre de buffers disponibles pour le tri, les processus de consommation peuvent effectuer plusieurs fusions, en écrivant directement les résultats sur le disque et en lisant, puis en fusionnant ces résultats, jusqu'à ce que toutes les données de l'intervalle soient complètement triées.

- En ce qui concerne les commandes `create index`, chaque processus de consommation de chaque intervalle de données partitionné écrit sur un device de base de données distinct. Cela permet d'améliorer les performances avec une parallélisation d'E/S accrue, lorsque les devices de base de données se trouvent sur des devices physiques et des contrôleurs distincts. Le processus de consommation génère également un index, appelé sous-index, à partir des données triées.
- Pour les jointures par fusion, chaque processus de consommation inscrit les lignes triées dans un groupe distinct de pages de données liées entre elles, une pour chaque processus de travail qui effectue la fusion.
- Dans le cas d'une requête, le processus de consommation classe simplement les données par ordre de grandeur dans l'intervalle.

Fusion des résultats

Une fois que tous les processus de consommation ont terminé le tri des données dans chaque intervalle partitionné :

- Dans le cas de commandes `create index`, le processus de coordination fusionne les sous-index et crée un index définitif.
- Dans le cas de jointures par fusion, les processus de travail effectuent la fusion avec les autres tables de la jointure.
- Pour les autres requêtes, le processus de coordination fusionne les résultats du tri et les renvoie au client.

Configuration des ressources pour le tri parallèle

Les sections suivantes décrivent les différentes ressources utilisées par Adaptive Server lors du tri parallèle des données :

- Les processus de travail lisent les données et effectuent le tri.
- Les buffers de tri transmettent les données du cache des processus de production aux processus de consommation, d'où une réduction des E/S physiques.
- La présence dans le cache de grandes zones d'E/S utilisées pour le tri permet également de réduire les E/S physiques.

Remarque Les références aux E/S étendues sont sur un serveur de pages logiques dont la taille est de 2 Ko. Si la taille des pages de votre serveur est de 8 Ko, l'unité de base de l'E/S sera de 8 Ko. Si elle est de 16 Ko, l'unité de base de l'E/S sera de 16 Ko.

- Plusieurs devices physiques permettent d'accroître la parallélisation des E/S et d'évaluer le nombre de processus de travail requis par la plupart des tris.

Processus de travail requis pour le tri parallèle

Pour effectuer un tri en mode parallèle, Adaptive Server a besoin d'un nombre minimum de processus de travail. Plus ils sont nombreux, plus le tri sera rapide. Le nombre minimum requis et le nombre maximum de processus de travail utilisables sont déterminés en fonction du nombre de :

- partitions sur la table, pour la création d'index clusterisés,
- devices, pour la création d'index clusterisés,
- threads utilisées pour créer la table de travail et de devices dans tempdb, pour les jointures par fusion,
- devices dans tempdb, pour les autres requêtes nécessitant un tri.

S'il est impossible de disposer du nombre minimum de processus de travail :

- Les tris des index clusterisés sur des tables partitionnées doivent être effectués en parallèle ; ils n'aboutissent pas lorsque le nombre de processus de travail disponibles est insuffisant.
- Les tris des index non clusterisés et des index clusterisés dans des tables non partitionnées peuvent se dérouler en mode série.
- Tous les tris des requêtes peuvent être effectués en série.

La disponibilité des processus de travail est limitée au niveau du serveur et de la session. Au niveau du serveur, les paramètres de configuration number of worker processes et max parallel degree restreignent la taille de la zone des processus de travail, ainsi que le nombre maximum utilisable par les commandes `create index` ou `select`.

Au moment de l'exécution, il est possible que le nombre de processus disponibles soit inférieur à la valeur de `max parallel degree` ou à la limite définie pour la session, en raison de l'exécution en parallèle d'autres requêtes. Il revient au gestionnaire de tri et non à l'optimiseur de décider le nombre de processus de travail à utiliser pour un tri. Cette décision étant prise au moment de l'exécution, celles concernant le tri parallèle s'appuient sur le nombre réel de processus de travail disponibles au moment où commence le tri.

Pour plus d'informations sur les limites au niveau du serveur et de la session, reportez-vous à la section « [Contrôle du degré de parallélisation](#) », page 167.

Conditions requises pour créer des index avec des processus de travail

Le [tableau 9-1](#) indique le nombre de processus de production et de consommation requises pour créer des index. Le **segment cible** est le segment sur lequel l'index est stocké lorsque l'exécution de la commande `create index` est terminée. Lorsque vous créez un index, vous pouvez définir l'emplacement à l'aide de la clause `on nom_segment`. Si vous n'indiquez pas de segment particulier, l'index est stocké sur le segment `default`.

Tableau 9-1 : Nombre de processus de production et de consommation requis pour créer des index

Type d'index	de production	de consommation
Index non clusterisé	Nombre de partitions ou 1	Nombre de devices sur le segment cible
Index clusterisé sur une table non partitionnée	1	Nombre de devices sur le segment cible
Index clusterisé sur une table partitionnée	Nombre de partitions ou 1	Nombre de partitions

L'essentiel du tri parallèle repose sur les processus de consommation. Ils utilisent le temps CPU pour effectuer le tri, ainsi que les E/S pour lire et écrire les résultats intermédiaires et pour écrire l'index définitif sur le disque. En premier lieu, le gestionnaire de tri affecte un processus de consommation à chaque device cible. Ensuite, si le nombre de processus de travail disponibles est suffisant, il affecte un processus de production à chaque partition de la table. S'il n'y a pas assez de processus de travail, la totalité de la table est analysée par un seul processus de production.

Index clusterisé sur des tables partitionnées

Pour créer un index clusterisé sur une table partitionnée, Adaptive Server a besoin d'au moins un processus de consommation par partition sur la table et d'un processus de travail supplémentaire pour balayer la table. Si le nombre de processus de travail ne suffit pas, la commande `create clustered index` ne s'exécute pas et affiche le nombre de processus de travail disponibles par rapport à celui requis.

Si le nombre de processus de travail est suffisant, le gestionnaire de tri affecte un processus de production et un processus de consommation par partition. Cette méthode permet d'accélérer la lecture des données.

Minimale	1 processus de consommation par partition, plus 1 processus de production
Maximale	2 processus de travail par partition
Peut être effectué en série	Non

Index clusterisé sur une table non partitionnée

Il n'est pas possible d'utiliser plus d'un processus de production pour analyser les données entrées destinées aux tables non partitionnées. Le nombre de processus de consommation est déterminé par le nombre de devices sur le segment où l'index doit être stocké. Si le nombre de processus de travail disponibles est insuffisant, le tri peut être effectué en série.

Minimale	1 processus de consommation par device, plus 1 processus de production
Maximale	1 processus de consommation par device, plus 1 processus de production
Peut être effectué en série	Oui

Index non clusterisés

Le nombre de processus de consommation est déterminé par le nombre de devices sur le segment cible. Si le nombre de processus de travail est suffisant et que la table est partitionnée, un processus de production est utilisé par partition ou bien un processus de production unique analyse la totalité de la table. Si le nombre de processus de travail disponibles est insuffisant, le tri peut être effectué en série.

Minimale	1 processus de consommation par device, plus 1 processus de production
Maximale	1 processus de consommation par device, plus 1 processus de production par partition
Peut être effectué en série	Oui

Utilisation de la clause *with consumers* lors de la création d'index

Adaptive Server considère les devices RAID comme des devices de base de données. Même si ces devices peuvent gérer les E/S des tris parallèles, Adaptive Server n'affecte par défaut qu'un seul processus de consommation à chaque device.

La clause *with consumers* dans l'instruction *create index* permet de définir le nombre de processus de consommation utilisés par *create index*. En testant la capacité d'E/S des devices sauvegardés, vous pouvez évaluer le nombre de processus simultanés que le device RAID peut supporter et utiliser ce nombre pour définir un degré de parallélisation. A titre de référence, utilisez un processus de consommation pour chaque device physique sous-jacent. L'exemple suivant porte sur huit processus de consommation :

```
create index order_ix on orders (order_id)
with consumers = 8
```

Vous pouvez également utiliser la clause `with consumers` avec les clauses `alter table...add constraint` qui permettent de créer les index, primary key et unique :

```
alter table orders
  add constraint prim_key primary key (order_id) with
    consumers = 8
```

La clause `with consumers` permet de créer des index (vous ne pouvez pas contrôler le nombre de processus de consommation utilisés lors des tris internes pour des requêtes parallèles). Vous ne pouvez pas utiliser cette clause lorsque vous créez un index clusterisé sur une table partitionnée. Si vous créez un index clusterisé sur une table partitionnée, Adaptive Server doit utiliser un processus de consommation pour chaque partition afin d'assurer une répartition uniforme des données triées entre les partitions.

Adaptive Server ne tient pas compte de la clause `with consumers` lorsque le nombre de processus défini est supérieur au nombre de processus de travail disponibles ou lorsqu'il dépasse les limites de parallélisation au niveau du serveur ou de la session.

Processus de travail requis pour le tri de la requête select

Les plans d'exécution des requêtes qui requièrent le tri des tables de travail comprennent plusieurs étapes. La définition du nombre de processus de travail pour le tri d'une table de travail intervient à la fin du balayage de la table sous-jacente. Au cours de l'étape qui consiste à sélectionner des données dans une table de travail, chaque processus de travail sélectionne des données et les copie dans une partition distincte.

Une fois la table de travail créée, des processus de travail supplémentaires sont affectés pour effectuer le tri. `showplan` n'affiche pas cette valeur ; le gestionnaire de tri indique uniquement si le tri est effectué en série ou en parallèle. Les processus de travail utilisés dans la phase précédente n'interviennent pas dans le tri, mais ils demeurent affectés à la tâche en parallèle jusqu'à la fin de l'opération.

Processus de travail pour des tris dans des jointures par fusion

Pour des jointures par fusion, un processus de consommation est affecté à chaque device dans tempdb ; s'il n'existe dans tempdb qu'un seul device, deux processus de consommation sont utilisés. Le nombre de processus de production dépend du nombre de partitions dans la table de travail et du paramètre max parallel degree :

- Si la table de travail n'est pas partitionnée, un seul processus de production est utilisé.
- Si le nombre de processus de consommation ajoutés au nombre de partitions dans la table de travail est inférieur ou égal à la valeur de max parallel degree, un seul processus de production est alloué à chaque partition de la table de travail.
- Si le nombre de processus de consommation ajoutés au nombre de partitions de la table de travail est supérieur à la valeur de max parallel degree, un processus de production est utilisé.

Autres tris de tables de travail

Pour tous les autres tris de tables de travail, celles-ci ne sont pas partitionnées après avoir été créées. Les processus de travail sont alloués comme suit :

- Si tempdb ne contient qu'un seul device, le tri est assuré par deux processus de consommation et un processus de production ; sinon, un processus de consommation est affecté à chaque device de tempdb et un seul processus de production analyse la table de travail.
- Si tempdb comporte plus de devices que le nombre de processus de travail disponibles au début du tri, ce dernier est effectué en série.

Caches, buffers de tri et tris parallèles

Une configuration optimale du cache et du paramètre number of sort buffers permet d'accélérer considérablement les tris parallèles. Les options d'optimisation sont les suivantes :

- liaisons au cache,
- buffers de tri,
- E/S étendues.

Dans la plupart des cas, la configuration que vous choisissez dans le cadre d'une exécution normale doit répondre aux besoins des requêtes qui effectuent le tri des tables de travail. Vous devez donc savoir précisément combien de tris simultanés sont requis et quelle est la taille approximative des tables de travail, puis configurer le cache utilisé par tempdb pour optimiser le tri.

Si vous créez et supprimez des index lorsque le système est peu utilisé, vous pouvez reconfigurer les caches et les zones, puis modifier les liaisons au cache afin d'optimiser les tris et de réduire le temps d'exécution. En revanche, si vous devez remanier les index en pleine période d'activité, vous devez prévoir les incidences sur le temps de réponse. La configuration d'un pourcentage important du cache pour qu'il soit exclusivement affecté au tri ou l'annulation temporaire des liaisons d'objets au cache peut avoir des conséquences déterminantes sur les performances des autres tâches.

Liaisons au cache

Les tris associés à la commande `create index` ont lieu dans le cache auquel une table est liée. Si la table est liée à un cache mais pas la base de données, le cache est utilisé. S'il n'existe pas de liaison au cache explicite, le cache de données par défaut est utilisé. Les tris des tables de travail utilisent le cache auquel tempdb est liée, ou le cache de données par défaut.

Lorsque vous configurez le nombre de buffers de tri et des E/S étendues pour un tri spécifique, vérifiez toujours les liaisons au cache. Pour voir les liaisons d'une table, utilisez la procédure `sp_help`. Pour afficher toutes les liaisons au cache sur un serveur, utilisez `sp_helpcache`. Lorsque vous avez défini les liaisons d'une table au cache, utilisez `sp_cacheconfig` pour vérifier l'espace libre dans les zones de 2 Ko et 16 Ko du cache.

Incidence du nombre de buffers sur les performances du tri

Les processus de production effectuent des E/S disque pour lire la table d'entrée et les processus de consommation effectuent des E/S disque pour lire et écrire les résultats intermédiaires des tris sur le disque. Durant le tri, les processus de production transmettent des données aux processus de consommation en utilisant les buffers de tri. Ceci évite les E/S sur disque grâce à la copie des lignes de données en mémoire. Durant le tri, les buffers réservés ne sont pas disponibles pour d'autres tâches.

Le paramètre de configuration number of sort buffers détermine l'espace maximum utilisable pour un tri en série. Chaque instance de tri peut utiliser une valeur inférieure ou égale à celle de number of sort buffers pour chaque tri. Si les tris actifs monopolisent tous les buffers du cache et si un autre tri requiert des buffers, celui-ci doit attendre que des buffers se libèrent dans le cache.

Instructions pour configurer les buffers de tri

Etant donné que le paramètre number of sort buffers permet de définir le volume de données pouvant être lu et trié dans un batch, il est possible de configurer plus de buffers pour accroître la taille du batch, réduire le nombre de fusions requises et accélérer l'exécution du tri. La modification de la valeur de number of sort buffers est dynamique et vous n'avez donc pas besoin de redémarrer le serveur.

Les instructions relatives à la configuration des buffers de tri sont les suivantes :

- Le gestionnaire de tri sélectionne les tris en série lorsque le nombre de pages dans une table est inférieur à 8 fois la valeur de number of sort buffers. Dans la plupart des cas, la valeur par défaut (500) convient parfaitement pour les requêtes select et les index de petite taille. Le gestionnaire de tri sélectionne le tri en série pour toutes les commandes create index et les tris des tables de travail de 4 000 pages ou moins ; il sélectionne les tris parallèles lorsqu'il s'agit de résultats nombreux afin de réserver les processus de travail pour le traitement des requêtes et les tris plus importants. Il peut également permettre à plusieurs processus de tri d'utiliser jusqu'à 500 buffers simultanément.

La taille d'une table de travail temporaire doit être très importante pour justifier une valeur plus élevée afin de réduire le nombre de fusions au cours d'un tri. Reportez-vous à la section « [Taille de tempdb](#) », page 419 du guide *Performances et optimisation : concepts de base* pour de plus amples informations.

- Si vous créez des index sur de grandes tables alors que d'autres utilisateurs sont actifs, vous devez définir le nombre de buffers de tri de manière à ne pas interrompre les autres activités qui ont besoin d'accéder au cache de données.

- Si vous reconstruisez des index pendant une période de maintenance au cours de laquelle il y a peu d'utilisateurs, vous pouvez attribuer une valeur élevée aux buffers de tri. Pour accélérer la maintenance et optimiser l'activité de votre index, vous pouvez comparer les performances obtenues avec des valeurs de buffer élevées, les E/S étendues et les liaisons au cache.
- La réduction du nombre de fusions est une fonction logarithmique. Si la valeur de `number of sort buffers` augmente de 500 à 600, les conséquences sur le nombre de fusions sont négligeables. En revanche, une augmentation significative de la taille, par exemple 5 000, accélère considérablement le tri en réduisant le nombre de fusions et le nombre d'E/S requis.
- Si `number of sort buffers` est inférieur à la racine carrée de la taille de la table de travail, les performances du tri se dégradent. Dans la mesure où les tables de travail n'incluent que les colonnes spécifiées dans la liste de sélection, plus celles nécessaires aux jointures ultérieures, leur taille pour des jointures par fusion est en général largement inférieure à celle des tables d'origine.
- Configuration d'un nombre suffisant de buffers de tri.

Les buffers de tri déterminent le nombre de pages de données que vous pouvez trier au cours de chaque exécution. Ce nombre sert de base à la fonction logarithmique de calcul du nombre d'exécutions nécessaires pour terminer le tri.

Par exemple, si vous avez 500 buffers, le nombre d'exécution est calculé à l'aide de « `log (nombre de pages dans la table) with 500 as the log base` ».

Notez également que le nombre de buffers de tri est partagé par les threads dans le tri parallèle. Ce qui signifie que si vous n'avez suffisamment de buffers de tri, le tri parallèle risque d'être plus lent que prévu.

Lorsque le nombre de buffers configurés est suffisant, les tris requièrent moins d'étapes intermédiaires, de fusions et d'E/S physiques. Lorsque `number of sort buffers` est égal ou supérieur au nombre de pages dans la table, le tri peut être intégralement exécuté dans le cache sans E/S physiques pour les étapes intermédiaires : les seules E/S requises sont celles qui permettent de lire et d'écrire les pages de données et d'index.

- Pour avoir des E/S étendues, configurez des zones de buffers de grande taille dans un cache nommé et liez ce cache à la table.

Utilisation d'un nombre de buffers inférieur à celui qui est défini

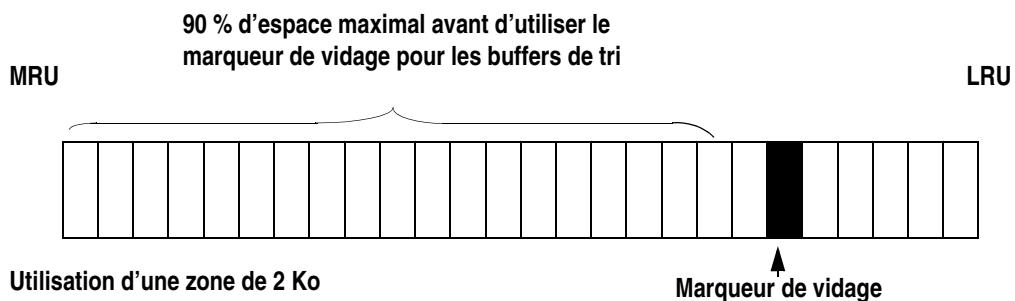
Il existe deux types de tri qui peuvent utiliser moins de buffers que prévu :

- La création d'un index clusterisé sur une table partitionnée requiert toujours un tri parallèle. Si la taille de la table est inférieure au nombre de buffers de tri prévu, le nombre de pages est alors réservé pour le tri.
- Les petits tris en série réservent juste le nombre de buffers requis pour conserver la table dans le cache.

Définition du paramètre *number of sort buffers*

Lorsque vous créez des index en parallèle, la valeur de *number of sort buffers* doit être égale ou inférieure à 90 pour cent du nombre de buffers figurant dans la zone de 2 Ko avant le marqueur de vidage, comme le montre la [figure 9-2](#).

Figure 9-2 : Zone disponible pour les buffers de tri



La limite de 90 pour cent de la taille de la zone n'est pas prise en compte lorsque vous configurez le paramètre *number of sort buffers* ; en revanche, elle l'est lorsque vous exécutez la commande `create index`, car elle s'applique à la zone de la table faisant l'objet du tri. La valeur maximale autorisée pour le paramètre *number of sort buffers* est 32 767 ; elle est appliquée par `sp_configure`.

Calcul de la valeur d'un buffer de tri pour une zone déterminée

sp_cacheconfig renvoie la taille de la zone en méga-octets et celle du vidage en kilo-octets. Le résultat ci-dessous décrit la taille des zones dans le cache de données par défaut :

```
Cache: default data cache, Status: Active, Type: Default
Config Size: 0.00 Mb, Run Size: 38.23 Mb
Config Replacement: strict LRU, Run Replacement: strict LRU
Config Partition: 2, Run Partition: 2
IO Size Wash Size Config Size Run Size APP Percent
----- -----
2 Kb    4544 Kb      0.00 Mb    22.23 Mb    10
16 Kb   3200 Kb     16.00 Mb    16.00 Mb    10
```

La procédure suivante reprend les paramètres relatifs à la taille de la zone de 2 Ko et du vidage, convertit ces deux valeurs en pages et calcule le nombre de pages maximum pouvant être affecté aux buffers de tri :

```
create proc bufs @poolsize numeric(6,2), @wash int
as
select "90% of non-wash 2k pool" =
((@poolsize * 512) - (@wash/2)) *.9
```

L'exemple suivant illustre l'exécution de bufs avec les valeurs 22,23 Mo pour la taille de la zone et 4 544 Ko pour la taille du vidage :

```
bufs 22.23, 4544
```

La procédure bufs renvoie les résultats suivants :

```
90% of non-wash 2k pool
-----
8198.784
```

La commande suivante définit le nombre de buffers de tri avec 8 198 pages :

```
sp_configure "number of sort buffers", 8198
```

Si la table sur laquelle vous souhaitez créer l'index est liée à un cache défini par l'utilisateur, vous devez définir le nombre de buffers de tri approprié. Vous pouvez également dissocier la table du cache, créer l'index et recréer la liaison :

```
sp_unbindcache pubtune, titles
create clustered index title_ix
on titles (title_id)
sp_bindcache pubtune_cache, pubtune, titles
```

Avertissement ! Les buffers utilisés lors d'un tri sont exclusivement affectés à ce tri pendant toute la durée de l'opération. Ils ne peuvent pas être utilisés par une autre tâche sur le serveur. Si vous définissez un nombre de buffers qui représente 90 pour cent de la taille de la zone, vous risquez de réduire les performances des requêtes lorsque vous créez des index alors que d'autres transactions sont actives.

Procédure d'évaluation des fusions et des E/S

La procédure suivante permet d'évaluer le nombre de fusions et le volume d'E/S requis pour créer un index :

```
create proc merge_runs @pages int, @bufs int
as
declare @runs int, @merges int, @maxmerge int

select @runs = ceiling ( @pages / @bufs )

/* if all pages fit into sort buffers, no merge runs needed */
if @runs <=1
    select @merges = 0
else
begin
    if @runs > @bufs select @maxmerge = @bufs
    else select @maxmerge = @runs

    if @maxmerge < 2 select @maxmerge = 2

    select @merges = ceiling(log10(@runs) / log10(@maxmerge))
end
select @merges "Merge Levels",
      2 * @pages * @merges + @pages "Total IO"
```

Les paramètres applicables à la procédure sont les suivants :

- *pages* : le nombre de pages dans la table ou le nombre de pages niveau feuille dans un index non clusterisé.
- *bufs* : le nombre de buffers de tri à configurer.

Cet exemple fait intervenir le nombre de buffers de tri par défaut pour une table de 2 000 000 pages :

```
merge_runs 2000000, 500, 20
```

La procédure merge_runs permet d'évaluer 2 fusions et 10 000 000 E/S pour créer l'index :

```
Merge Levels Total IO
----- -----
2      10000000
```

Si le nombre de buffers de tri est 1 500, cette augmentation réduit le nombre de fusions et d'E/S requises :

```
merge_runs 2000000, 1500
Merge Levels Total IO
----- -----
1      6000000
```

Il se peut que le total d'E/S prévu par cette procédure ne soit pas compatible avec la configuration de votre système, selon la taille et la configuration du cache et des zones utilisées par le tri.

Configuration de caches avec de grandes tailles d'E/S pendant le tri parallèle

Les tris peuvent utiliser des E/S étendues :

- lors des phases d'échantillonnage,
- pour les processus de production qui analysent les tables d'entrées,
- pour les processus de consommation qui exécutent des E/S disque sur les résultats intermédiaires et définitifs.

Lors de ces étapes, les tris peuvent utiliser la taille de zone la plus grande dans le cache utilisé par la table triée ; ils utilisent la zone de 2 Ko s'il n'y a pas de buffer d'E/S étendues disponible.

Buffers de tri et configuration des E/S étendues

La configuration d'une zone pour buffers de 16 Ko dans le cache utilisé par le tri permet d'accroître considérablement les E/S et de réduire le nombre d'E/S physiques. Cette réduction d'E/S est due en partie à l'utilisation d'E/S étendues pour balayer la table d'entrée.

Les E/S supplémentaires, qu'il s'agisse de lectures ou d'écritures, interviennent lors des fusions pendant le tri. Le nombre d'E/S lors de cette étape dépend du nombre de phases de fusion requises. Au cours du tri et de l'étape de fusion, les buffers sont lus une fois et ne sont pas réutilisés ; dans le cas contraire, ils sont remplis avec les résultats des tris intermédiaires, font l'objet d'une écriture sur le disque et sont prêts à être réutilisés. Le taux de présence dans le cache lors des tris est toujours faible, de sorte que la configuration d'un cache de 16 Ko occupe un espace qui peut être mieux utilisé pour les buffers de tri afin de réduire le nombre de fusions.

Par exemple, la création d'un index clusterisé sur une table de 250 Mo dotée d'un cache de 32 Mo est effectuée avec 4 Mo seulement dans la zone de 16 Ko et avec 10 000 buffers de tri. Des zones plus larges n'ont pas d'incidence sur le taux de présence dans le cache ou le nombre d'E/S. Le paramétrage de la taille de vidage d'une zone de 16 Ko sur la valeur maximale autorisée améliore légèrement les performances puisque la petite taille de la zone permet aux buffers d'atteindre l'extrémité LRI du cache avant la fin des opérations d'écriture. La formule suivante permet de calculer la taille de vidage maximale autorisée pour une zone de 16 Ko :

```
select floor((size_in_MB * 1024 /16) * .8) * 16
```

Conditions requises au niveau du disque

Les conditions requises au niveau du disque par le tri parallèle sont les suivantes :

- De l'espace est nécessaire pour stocker l'index achevé.
- La présence de plusieurs devices dans le segment cible augmente le nombre de processus de consommation pour les tris des tables de travail et pour la création d'index non clusterisés et d'index clusterisés sur des tables non partitionnées.

Espace requis pour la création d'index

La création d'index requiert de l'espace pour stocker l'index trié. Dans le cas des index clusterisés, il est nécessaire de copier les lignes de données dans de nouveaux emplacements en respectant l'ordre de la clé d'index. Ces lignes de données remaniées et les niveaux supérieurs de l'index doivent faire l'objet d'une écriture avant que la table sous-jacente ne soit supprimée. A moins d'utiliser la clause `with sorted_data` pour supprimer le tri, la création d'un index clusterisé requiert approximativement 120 pour cent de l'espace occupé par la table.

La création d'un index non clusterisé requiert de l'espace pour stocker le nouvel index. Pour évaluer la taille des objets et l'espace libre, utilisez les procédures système suivantes :

- `sp_spaceused` permet de voir la taille de la table. Reportez-vous à la section « [Affichage de la taille des objets à l'aide de sp_spaceused](#) », page 273 du guide *Performances et optimisation : concepts de base*.
- `sp_estspace` permet de prévoir la taille de l'index. Reportez-vous à la section « [Evaluation de la taille des objets à l'aide de sp_estspace](#) », page 275 du guide *Performances et optimisation : concepts de base*.
- `sp_helpsegment` permet d'évaluer l'espace libre sur un segment de base de données. Reportez-vous à la section « [Contrôle de la distribution des données sur les devices avec sp_helpsegment](#) », page 124 du guide *Performances et optimisation : concepts de base*.

Espace requis pour les tris des tables de travail

Les requêtes qui trient des tables de travail (jointures par fusion et `order by`, `distinct`, `union` et redéfinition d'index) commencent par copier les colonnes requises dans les tables de travail, puis effectuent le tri. Ces tables de travail sont stockées sur le segment `system` dans `tempdb`, qui devient alors le segment cible pour les requêtes avec un tri. Pour savoir quel est l'espace libre et le nombre de devices, utilisez :

```
tempdb..sp_helpsegment system
```

Pour exécution en parallèle, l'insertion des lignes dans la table de travail et le tri parallèle n'exigent pas plusieurs devices. Toutefois, les performances sont meilleures lorsque le segment `system` dans `tempdb` s'étend sur plusieurs devices de base de données.

Nombre de devices dans le segment cible

Comme indiqué dans la section « [Processus de travail requis pour le tri parallèle](#) », page 240, le nombre de processus de consommation affectés aux opérations de tri dépend du nombre de devices dans le segment cible, sauf lorsqu'il s'agit de créer un index sur une table partitionnée.

Les aspects relatifs aux performances des requêtes, tels que l'amélioration des E/S lorsque les index résident sur des devices séparés des données, sont plus importants pour définir les allocations des devices et l'emplacement des objets que les conditions requises en matière de tri.

Si les tris d'une table de travail sont suffisamment importants pour être traités en parallèle, la présence de plusieurs devices dans le segment système de tempdb permettra d'accélérer ces tris et d'augmenter la parallélisation des E/S lors de l'insertion de lignes dans la table de travail.

Considérations de reprise

La création d'index est une opération qui requiert une connexion minimale à une base de données. Les tris en série sont récupérés dans le journal des transactions et entièrement retraités. En revanche, les commandes `create index` en parallèle ne sont pas récupérables dans le journal de transactions : après avoir exécuté un tri parallèle, vous devez sauvegarder la base de données pour pouvoir ensuite utiliser la commande `dump transaction`.

Adaptive Server n'effectue pas automatiquement un tri parallèle pour les commandes `create index`, sauf lorsque l'option de base de données `select into/bulk copy/pllsort` est associée au paramètre `on`. La création d'un index clusterisé sur une table partitionnée requiert toujours un tri parallèle.

Lorsque l'option `select into/bulk copy/pllsort` est désactivée, vous pouvez effectuer le tri en série.

Outils permettant de contrôler et d'optimiser les tris

Adaptive Server intègre plusieurs outils permettant de contrôler les opérations de tri :

- `set sort_resources on` permet de simuler l'exécution d'une commande `create index`, sans créer d'index. Reportez-vous à la section « [Utilisation de la commande set sort_resources on](#) », page 256.
- Plusieurs procédures système permettent d'évaluer la taille, l'espace et la durée :
 - `sp_configure` : affiche des paramètres de configuration. Reportez-vous à la section « [Paramètres de configuration pour le contrôle de la parallélisation](#) », page 168.
 - `sp_helppartition` : affiche des informations sur les tables partitionnées. Reportez-vous à la section « [Informations sur les partitions](#) », page 121 du guide *Performances et optimisation : concepts de base*.
 - `sp_helpsegment` : affiche des informations sur les segments, les devices et l'utilisation de l'espace. Reportez-vous à la section « [Contrôle de la distribution des données sur les devices avec sp_helpsegment](#) », page 124 du guide *Performances et optimisation : concepts de base*.
 - `sp_sysmon` : affiche un rapport sur les nombreuses ressources système utilisées pour les tris parallèles, notamment l'utilisation de la CPU, les E/S physiques et la mise en mémoire cache. Reportez-vous à la section « [Utilisation de sp_sysmon pour optimiser la création d'index](#) », page 261.

Utilisation de la commande `set sort_resources on`

La commande `set sort_resources on` permet de mieux comprendre comment le gestionnaire de tri effectue le tri parallèle dans le cadre des instructions `create index`. Vous pouvez l'utiliser avant de créer un index pour décider si vous souhaitez augmenter la valeur des paramètres de configuration ou définir des processus de consommation supplémentaires pour un tri.

Après avoir utilisé la commande `set sort_resources on`, Adaptive Server ne crée pas d'index à proprement parler ; il analyse les ressources, procède à l'échantillonnage et imprime des rapports détaillés sur la manière dont il convient d'utiliser le tri parallèle pour exécuter la commande `create index`. Le tableau 9-2 décrit les messages qui s'impriment pour les opérations de tri.

Tableau 9-2 : Messages élémentaires sur les opérations de tri

Message	Explication	Se reporter à
The Create Index is done using <code>type_tri</code>	<code>type_tri</code> correspond soit à « Parallel Sort » (tri parallèle), soit à « Serial Sort » (tri en série).	« Présentation des conditions et des ressources », page 234
Sort buffer size: <code>N</code>	<code>N</code> est la valeur définie pour le paramètre de configuration <code>number of sort buffers</code> .	« Instructions pour configurer les buffers de tri », page 247
Parallel degree: <code>N</code>	<code>N</code> est le nombre maximum de processus de travail que le tri parallèle peut utiliser, conformément aux paramètres de configuration.	« Caches, buffers de tri et tris parallèles », page 245
Number of output devices: <code>N</code>	<code>N</code> est le nombre total de devices de base de données sur le segment cible.	« Conditions requises au niveau du disque », page 253
Number of producer threads: <code>N</code>	<code>N</code> est le nombre optimal de processus de production défini par le gestionnaire de tri.	« Processus de travail requis pour le tri parallèle », page 240
Number of consumer threads: <code>N</code>	<code>N</code> est le nombre optimal de processus de consommation défini par le gestionnaire de tri.	« Processus de travail requis pour le tri parallèle », page 240
The distribution map contains <code>M</code> element(s) for <code>N</code> partitions.	<code>M</code> est le nombre d'éléments qui définit les limites d'intervalle dans le plan de répartition. <code>N</code> est le nombre total de partitions (intervalles) dans le plan de répartition.	« Création d'un plan de répartition », page 238
Partition Element: <code>N</code> <code>valeur</code>	<code>N</code> est le numéro de l'élément du plan de répartition. <code>valeur</code> est l'élément du plan de répartition qui définit les limites de chaque partition.	« Création d'un plan de répartition », page 238
Number of sampled records: <code>N</code>	<code>N</code> est le nombre d'enregistrements d'échantillonnage utilisés pour générer le plan de répartition.	« Création d'un plan de répartition », page 238

Exemples

Les exemples suivants illustrent le résultat de l'exécution de la commande `set sort_resources`.

Index non clusterisé sur une table non partitionnée

Cet exemple décrit comment Adaptive Server effectue un tri parallèle pour une commande `create index` sur une table non partitionnée. L'exemple tient compte des détails suivants :

- Le segment default s'étend sur 4 devices de base de données.
- `max parallel degree` est égal à 20 processus de travail.
- `number of sort buffers` est égal à la valeur par défaut, c'est-à-dire 500 buffers.

Les commandes suivantes permettent de définir `sort_resources` on et d'exécuter une commande `create index` sur la table `orders` :

```
set sort_resources on
create index order_ix on orders (order_id)
```

Adaptive Server imprime les résultats suivants :

```
The Create Index is done using Parallel Sort
Sort buffer size: 500
Parallel degree: 20
Number of output devices: 4
Number of producer threads: 1
Number of consumer threads: 4
The distribution map contains 3 element(s) for 4
partitions.
Partition Element: 1

458052

Partition Element: 2

909063

Partition Element: 3

1355747

Number of sampled records: 2418
```

Dans cet exemple, les 4 devices sur le segment default désignent le nombre de processus de consommation. Etant donné que la table d'entrée n'est pas partitionnée, le gestionnaire de tri alloue un processus de production, avec un degré de parallélisation de 5.

Le plan de répartition utilise 3 valeurs pour les 4 intervalles. Les valeurs d'entrée les plus faibles jusqu'à la valeur 458 052 (inclus) dépendent du premier intervalle. Les valeurs supérieures à 458 052 et inférieures ou égales à 909 063 appartiennent au second intervalle. Les valeurs supérieures à 909 063 et inférieures ou égales à 1 355 747 appartiennent au troisième intervalle. Les valeurs supérieures à 1 355 747 dépendent du quatrième intervalle.

Index non clusterisé sur une table partitionnée

Cet exemple utilise les mêmes tables et devices que le premier exemple. Cependant, dans ce cas, la table d'entrée est partitionnée avant la création de l'index non clusterisé. Les commandes sont les suivantes :

```
set sort_resources on
alter table orders partition 9
create index order_ix on orders (order_id)
```

Dans ce cas, la commande `create index` dans l'option `sort_resources` imprime les résultats suivants :

```
The Create Index is done using Parallel Sort
Sort buffer size: 500
Parallel degree: 20
Number of output devices: 4
Number of producer threads: 9
Number of consumer threads: 4
The distribution map contains 3 element(s) for 4
partitions.
Partition Element: 1

458464
Partition Element: 2

892035
Partition Element: 3

1349187
Number of sampled records: 2448
```

A présent que la table d'entrée est partitionnée, le gestionnaire de tri alloue 9 processus de production, sur un total de 13 processus de travail. Le nombre d'éléments dans le plan de répartition reste le même, même si les valeurs sont légèrement différentes par rapport à celles des exemples précédents.

Index clusterisé sur une table partitionnée exécutée en parallèle

Dans cet exemple, un index clusterisé est créé sur orders et le nom du segment est order_seg.

```
set sort_resources on
alter table orders partition 9
create clustered index order_ix
    on orders (order_id) on order_seg
```

Il y a 20 processus de travail disponibles et cette commande peut donc utiliser 9 processus de production et 9 processus de consommation, comme indiqué ci-dessous :

```
The Create Index is done using Parallel Sort
Sort buffer size: 500
Parallel degree: 20
Number of output devices: 9
Number of producer threads: 9
Number of consumer threads: 9
The distribution map contains 8 element(s) for 9
partitions.
Partition Element: 1

199141
Partition Element: 2

397543
Partition Element: 3

598758
Partition Element: 4

800484
Partition Element: 5

1010982
Partition Element: 6

1202471
Partition Element: 7
```

```
1397664
Partition Element: 8
```

```
1594563
Number of sampled records: 8055
```

Ce plan de répartition comprend 8 éléments pour les 9 partitions faisant l'objet du tri. Le nombre de processus de travail est 18.

Remarque Créez d'abord un index clusterisé, puis des index non clusterisés. En effet, lorsque vous créez un index clusterisé, tous les index non clusterisés existants sont reconstruits.

Echecs lors d'un tri

S'il n'y avait eu que 10 processus de travail disponibles pour la commande, celle-ci aurait pu aboutir en lisant la table entière avec un seul processus de production. S'il y avait eu moins de 10 processus de travail disponibles, le système aurait imprimé un message d'avertissement à la place du résultat de `sort_resources` :

```
Msg 1538, Level 17, State 1:
Server 'snipe', Line 1:
Parallel degree 8 is less than required parallel
degree 10 to create clustered index on partition
table. Change the parallel degree to required
parallel degree and retry.
```

Utilisation de `sp_sysmon` pour optimiser la création d'index

Vous pouvez employer la syntaxe de « `begin_sample` » et de « `end_sample` » avec `sp_sysmon` pour obtenir des résultats avec des commandes `create index` :

```
sp_sysmon begin_sample
create index ...
sp_sysmon end_sample
```

Les sections du rapport à vérifier sont les suivantes :

- « Sample Interval », qui correspond au temps total nécessaire à la création de l'index.
- Les statistiques relatives au cache utilisé par la table.
 - Vérifiez la valeur de « Buffer Grabs » pour les zones de 2 Ko et de 16 Ko afin d'évaluer l'efficacité des E/S étendues.
 - Vérifiez la valeur de « Dirty Buffer Grabs ». Si elle est différente de zéro, définissez la taille de vidage dans la zone supérieure et/ou augmentez la taille de la zone en utilisant sp_poolconfig.
- Les E/S disque utilisées par la table et les index : vérifiez la valeur de « Total Requested I/Os ».

Utilisation du tri parallèle pour accélérer la commande *create index*

Pour utiliser le tri parallèle pour accélérer la commande *create index*, définissez le segment cible sur plusieurs devices. Grâce à l'utilisation de plusieurs devices, le tri parallèle est en mesure d'utiliser pleinement l'E/S parallèle et Adaptive Server détermine le nombre de processus de consommation pour le tri et les opérations *create index* en fonction du nombre de devices.

Il n'est pas nécessaire de diviser la table pour exécuter la commande *create index* avec le tri parallèle. A la place, utilisez *create index* avec la clause *consumer*. Cependant, si le segment cible n'est pas présent sur plusieurs devices, Adaptive Server risque d'ignorer le nombre de processus de consommation que vous avez définis dans la clause *consumer*.

Optimisation de la prélecture asynchrone

Ce chapitre explique comment la prélecture asynchrone permet d'améliorer les performances des E/S pour de nombreux types de requête, grâce à la lecture des pages de données et d'index dans le cache avant qu'elles ne soient utilisées par une requête.

Sujet	Page
Amélioration des performances à l'aide de la prélecture asynchrone	263
Désactivation automatique de la prélecture	271
Optimisation des objectifs de la prélecture asynchrone	274
Autres fonctions d'optimisation des performances d'Adaptive Server	276
Paramètres spéciaux pour les limites de la prélecture asynchrone	279
Activités de maintenance pour des performances de prélecture élevées	280
Analyse des performances et prélecture asynchrone	281

Amélioration des performances à l'aide de la prélecture asynchrone

La prélecture asynchrone permet d'améliorer les performances, en prévoyant les pages requises pour des activités de base de données déterminées dont les modalités d'accès sont prévisibles. Les demandes d'E/S pour ces pages interviennent avant que la requête ne les sollicite afin que la plupart de ces pages se trouvent dans le cache au moment où la requête doit y accéder. La prélecture asynchrone permet d'améliorer les performances pour :

- les balayages séquentiels, tels que les balayages de tables, d'index clusterisés et d'index restreints non clusterisés ;
- les accès à l'aide d'index non clusterisés ;
- certaines vérifications effectuées à l'aide de dbcc et update statistics ;
- le processus de reprise.

La prélecture asynchrone permet d'améliorer les performances des requêtes qui accèdent à un grand nombre de pages, comme dans le cas des applications d'aide à la décision, à condition que les sous-systèmes d'E/S sur la machine ne soient pas saturés.

La prélecture asynchrone n'est pas utile ou très peu utile lorsque le sous-système d'E/S est saturé ou lorsque Adaptive Server consomme fortement la CPU. Elle est utilisable avec certaines applications transactionnelles, mais dans une moindre mesure car les requêtes transactionnelles exécutent généralement moins d'E/S.

Lorsqu'une requête dans Adaptive Server doit effectuer un balayage de table, elle exécute les opérations suivantes :

- Elle analyse les lignes sur une page et les valeurs dans les lignes.
- Elle cherche dans le cache la page suivante à lire dans une table. Si cette page figure dans le cache, le traitement de la tâche se poursuit. Dans le cas contraire, la tâche lance une demande d'E/S et se met en veille jusqu'à la fin de l'E/S.
- Lorsque l'E/S est terminée, la tâche se déplace de la file d'attente vers la file d'exécution. Lorsqu'elle est programmée sur un moteur, Adaptive Server analyse les lignes situées sur la page récemment extraite.

Ce cycle d'exécution et d'attente dans les lectures de disque se poursuit jusqu'à la fin du balayage de table. De la même manière, les requêtes qui utilisent un index non clusterisé traitent une page de données, lancent l'E/S pour la page suivante référencée par l'index et se mettent en veille jusqu'à ce que l'E/S soit terminée, si la page ne se trouve pas dans le cache.

Ce processus d'exécution et de mise en veille freine les performances des requêtes qui exécutent des E/S physiques pour un grand nombre de pages. Outre le délai d'attente jusqu'à l'achèvement des E/S physiques, la tâche s'active et se désactive alternativement sur le moteur. Il en résulte une augmentation de l'overhead de traitement.

Amélioration des performances des requêtes grâce à la prélecture des pages

La prélecture asynchrone lance des demandes d'E/S portant sur des pages avant qu'elles ne soient sollicitées par la requête afin que la plupart de ces pages se trouvent dans le cache au moment où la requête doit y accéder. Si les pages requises figurent déjà dans le cache, la requête n'arrête pas le moteur pour attendre la lecture physique. (Des arrêts peuvent se produire pour d'autres raisons, mais ils sont moins fréquents.)

Selon le type de requête exécuté, la prélecture asynchrone génère un **ensemble de pages préalablement lues** susceptible d'être utilisé très prochainement. Adaptive Server définit un ensemble de pages préalablement lues pour chaque type de traitement impliquant la prélecture asynchrone.

Les ensembles de pages préalablement lues sont parfois très précis mais il peut arriver que certaines hypothèses provoquent la prélecture de pages qui ne sont jamais lues. Lorsque le cache ne contient qu'un petit pourcentage de pages inutiles, les gains de performances en matière de prélecture asynchrone surpassent largement les inconvénients imputables aux lectures inutiles. Si le nombre de pages inutilisées s'accroît, Adaptive Server le détecte et réduit la taille de l'ensemble de pages préalablement lues ou désactive temporairement la prélecture.

Mécanismes de contrôle de la prélecture dans un environnement multi-utilisateur

Lorsque plusieurs requêtes simultanées exécutent la prélecture d'un grand nombre de pages dans une zone de buffers, il se peut que les buffers extraits pour une requête soient éliminés de la zone avant d'être utilisés.

Adaptive Server cherche à identifier les buffers placés dans chaque zone par la prélecture asynchrone, ainsi que le nombre de buffers utilisés. Il garde en mémoire le nombre de buffers par zone qui font l'objet d'une prélecture mais qui ne sont pas utilisés. Par défaut, Adaptive Server définit une limite de prélecture asynchrone de 10 % pour chaque zone. Par ailleurs, il est possible de limiter dans chaque zone le nombre de buffers inutilisés soumis à une prélecture.

Les restrictions par zone et les statistiques d'utilisation régissent la prélecture asynchrone pour maintenir un taux de présence élevé dans le cache et réduire les E/S inutiles. Le principal objectif est d'accroître le taux de présence des requêtes dans le cache et de réduire au minimum les attentes dues aux mises en veille des E/S sur le disque.

Les sections suivantes décrivent comment un ensemble de pages préalablement lues est généré pour les activités et les types de requête qui utilisent la prélecture asynchrone. Lors de certaines optimisations de prélecture asynchrone, des pages d'allocation sont utilisées pour générer l'ensemble de pages préalablement lues.

Pour plus d'informations sur la manière dont les pages d'allocation enregistrent les informations relatives au stockage d'objets, reportez-vous à la section « [Pages d'allocation](#) », page 170.

Ensemble de pages préalablement lues lors d'une reprise

Pendant la reprise, Adaptive Server lit chacune des pages du journal qui contiennent les enregistrements d'une transaction, puis il lit toutes les pages de données et d'index référencées par cette transaction pour vérifier l'estampille et pour annuler ou valider les transactions. Il effectue ensuite la même opération pour la transaction suivante, jusqu'au traitement complet de toutes les transactions d'une base de données. Il existe deux types de prélecture asynchrone permettant d'accélérer la reprise : la prélecture asynchrone sur les pages de journal proprement dites et la prélecture asynchrone sur les pages de données et d'index référencées.

Prélecture des pages de journal

Le journal de transactions est stocké en différé sur le disque et remplit des extents dans chaque unité d'allocation. Chaque fois que le processus de reprise lit une page de journal dans une nouvelle unité d'allocation, il effectue la prélecture de toutes les pages de cette unité qui sont utilisées par le journal.

Dans les bases de données qui ne comportent pas de segments de journal distincts, les extents de journal et de données peuvent être combinés sur la même unité d'allocation. La prélecture asynchrone continue à extraire toutes les pages de journal sur l'unité d'allocation, mais les ensembles de pages préalablement lues peuvent être plus petits.

Prélecture de pages de données et d'index

Pour chaque transaction, Adaptive Server effectue un balayage du journal et génère l'ensemble de pages préalablement lues à partir de chaque page de données et d'index référencée. Pendant que les enregistrements de journal d'une transaction sont traités, la prélecture asynchrone lance des demandes pour les pages de données et d'index qui sont référencées par des transactions ultérieures dans le journal ; la prélecture permet de lire des transactions postérieures à la transaction en cours.

Remarque Une reprise n'utilise que la zone dans le cache de données par défaut. Pour de plus amples informations, reportez-vous à la section « [Limites de reprise](#) », page 279.

Ensemble de pages préalablement lues lors de balayages séquentiels

Les balayages séquentiels sont les balayages de tables, d'index clusterisés et d'index restreints non clusterisés.

Lors des balayages de tables et d'index clusterisés, la prélecture asynchrone utilise les informations de la page d'allocation sur les pages utilisées par l'objet pour générer l'ensemble de pages préalablement lues. Chaque fois qu'une page est extraite d'une nouvelle unité d'allocation, l'ensemble de pages préalablement lues est généré à partir de toutes les pages figurant sur cette unité qui sont utilisées par l'objet.

Le système enregistre la fréquence à laquelle un balayage séquentiel passe d'une unité d'allocation à une autre pour mesurer la fragmentation de chaînage de pages. Cette valeur permet d'ajuster la taille de l'ensemble de pages préalablement lues pour qu'une prélecture des volumes de pages importants soit effectuée lorsque la fragmentation est faible et pour que les volumes moins importants soient extraits lorsque la fragmentation est importante. Pour de plus amples informations, reportez-vous à la section « [Fragmentation des chaînages de pages](#) », page 273.

Ensemble de pages préalablement lues lors de l'accès à un index non clusterisé

Lorsque vous utilisez un index non clusterisé pour accéder à des lignes, la prélecture asynchrone recherche les numéros de page de toutes les valeurs d'index correspondantes sur une page de niveau feuille d'index non clusterisé. Elle permet de générer l'ensemble de pages préalablement lues d'après la liste de toutes les pages requises.

La prélecture asynchrone n'est utilisée que lorsque deux lignes au moins correspondent à une requête.

Si l'accès à un index non clusterisé requiert plusieurs pages de niveau feuille, les demandes de prélecture asynchrone portent également sur ces pages.

Ensemble de pages préalablement lues et vérifications dbcc

La prélecture asynchrone est utilisée lors des vérifications dbcc suivantes :

- dbcc checkalloc, qui contrôle l'allocation de la totalité des tables et des index dans une base de données, et les commandes dbcc tablealloc et dbcc indexalloc correspondantes au niveau objet.
- dbcc checkdb, qui contrôle tous les liens des tables et des index dans une base de données, et dbcc checktable, qui contrôle les tables individuelles et les index correspondants.

Contrôle des allocations

Les commandes dbcc checkalloc, tablealloc et indexalloc, qui contrôlent les allocations de pages, valident les informations contenues dans la page d'allocation. L'ensemble de pages préalablement lues pour les opérations dbcc permettant de contrôler l'allocation est semblable à l'ensemble de pages préalablement lues pour les autres balayages séquentiels. Lorsque le balayage porte sur une unité d'allocation différente, l'ensemble de pages préalablement lues est généré à partir des pages de l'unité d'allocation qui sont utilisées par l'objet.

checkdb et checktable

Les commandes dbcc checkdb et dbcc checktable permettent de contrôler les chaînages de pages d'une table et de générer l'ensemble de pages préalablement lues comme pour les autres balayages séquentiels.

Si la table contrôlée comporte des index non clusterisés, ceux-ci sont lus de manière récursive, depuis la page racine jusqu'aux pages de données, en passant par tous les pointeurs. Lorsque vous contrôlez les pointeurs des pages de niveau feuille aux pages de données, les commandes dbcc utilisent la prélecture asynchrone comme pour les balayages d'index non clusterisés. Lorsque vous accédez à une page d'index au niveau feuille, l'ensemble de pages préalablement lues est généré à partir des ID de toutes les pages référencées sur la page d'index au niveau feuille.

Tailles maximale et minimale d'un ensemble de pages préalablement lues

La taille d'un ensemble de pages préalablement lues pour une requête à un moment donné est fonction d'un certain nombre de facteurs :

- le type de requête, tel qu'un balayage séquentiel ou un balayage d'index non clusterisé ;
- la taille des zones utilisées par les objets référencés par la requête et la limite de prélecture définie pour chaque zone ;
- la rupture des tables ou des index, dans le cas des opérations de balayage ;
- le dernier taux de réussite en matière de demandes de prélecture asynchrone, ainsi que les conditions de surcharge dans les files d'E/S et les restrictions applicables aux E/S du serveur.

Le [tableau 10-1](#) regroupe les tailles minimales et maximales applicables aux différents types d'utilisation de la prélecture asynchrone.

Tableau 10-1 : Tailles des ensembles de pages préalablement lues

Type d'accès	Action	Tailles des ensembles de pages préalablement lues
Balayage de table	Lecture d'une page depuis une nouvelle unité d'allocation	Minimum de 8 pages nécessaires à la requête
Balayage d'index clusterisé		Le maximum est le plus petit des nombres suivants :
Balayage restreint au niveau feuille		<ul style="list-style-type: none"> • nombre de pages d'une unité d'allocation appartenant à un objet, • limite de prélecture de la zone.
Balayage d'index non clusterisé	Identification des lignes correspondantes sur la page de niveau feuille et préparation de l'accès aux pages de données	<p>Minimum de 2 lignes correspondantes</p> <p>Le maximum est le plus petit des nombres suivants :</p> <ul style="list-style-type: none"> • nombre de numéros de page uniques sur les lignes correspondantes de la page d'index de niveau feuille, • limite de prélecture de la zone.
Reprise	Reprise d'une transaction	<p>Le maximum est le plus petit des nombres suivants :</p> <ul style="list-style-type: none"> • nombre des pages de données et d'index concernées par une transaction en cours de reprise, • limite de prélecture dans la zone se trouvant dans le cache de données par défaut.
dbcc tablealloc, indexalloc et checkalloc	Balayage du chaînage de pages	Au maximum, toutes les pages sur une unité d'allocation appartenant au journal.
dbcc checktable et checkdb	Balayage du chaînage de pages Vérification des liaisons des index non clusterisés aux pages de données	<p>Identique au balayage de table.</p> <p>Toutes les pages de données référencées sur une page au niveau feuille.</p>

Désactivation automatique de la prélecture

La prélecture asynchrone a pour but d'extraire et de placer les pages requises dans des zones de buffers sans saturer ces zones ni le sous-système d'E/S et sans lire les pages inutiles. Lorsque Adaptive Server détecte la lecture dans le cache de pages extraites mais inutilisées, réduit temporairement ou interrompt la prélecture asynchrone.

Saturation des zones

Pour chaque zone figurant dans les caches de données, un pourcentage variable de buffers peut être lu avec la prélecture asynchrone, puis conservé en attendant une première utilisation. Par exemple, si une zone de 2 Ko comporte 4 000 buffers et que la limite applicable à cette zone est de 10 %, 400 buffers peuvent être lus par la prélecture asynchrone et conservés dans la zone sans être utilisés. Quand le nombre de buffers inutilisés dans la zone atteint 400, Adaptive Server interrompt temporairement la prélecture asynchrone dans cette zone.

A mesure que les requêtes accèdent aux pages de la zone, le nombre de buffers inutilisés diminue et la prélecture asynchrone se poursuit. Si le nombre de buffers disponibles est inférieur à celui de l'ensemble de pages préalablement lues, seul ce nombre de prélectures est effectué. Par exemple, si une zone d'une capacité de 400 buffers contient 350 buffers inutilisés et que l'ensemble de pages préalablement lues de la requête comprend 100 pages, seules les 50 premières prélectures asynchrones sont lancées.

Cette méthode permet d'éviter l'encombrement de la zone par de nombreuses demandes de prélecture qui expulsent les pages du cache avant qu'elles ne soient lues. Le nombre d'E/S asynchrones qui ne peuvent pas être exécutées en raison des restrictions par zone est indiqué par `sp_sysmon`.

Surcharges du système d'E/S

Adaptive Server et le système d'exploitation restreignent le nombre d'E/S en attente pour le serveur et pour chaque moteur. Les paramètres de configuration max async i/os per server et max async i/os per engine permettent de contrôler ces restrictions pour Adaptive Server. Pour plus d'informations sur la configuration de votre équipement, consultez la documentation de votre système d'exploitation.

Le paramètre de configuration disk i/o structures définit le nombre de blocs de contrôle du disque utilisés par Adaptive Server. Chaque E/S physique (chaque buffer, en lecture ou en écriture) requiert un bloc de contrôle lorsqu'elle se trouve dans la file d'attente des E/S.

Reportez-vous au *Guide d'administration système*.

Si Adaptive Server tente de lancer des demandes de prélecture asynchrone qui dépassent les limites définies par max async i/os per server, max async i/os per engine ou disk i/o structures, il traite les demandes jusqu'à la limite et rejette les autres. Par exemple, si 50 structures d'E/S disque seulement sont disponibles et si le serveur tente d'extraire 80 pages, 50 demandes sont traitées et les 30 autres rejetées.

sp_sysmon indique la fréquence à laquelle ces limites sont dépassées par des demandes de prélecture asynchrone. Reportez-vous à la section « [Rapport sur l'activité de prélecture asynchrone](#) », page 299 du guide *Performances et optimisation : contrôle et analyse*.

Lectures inutiles

La prélecture asynchrone est prévue pour éviter les lectures physiques inutiles. Lors d'une reprise ou de balayages d'index non clusterisés, les ensembles de pages préalablement lues sont très précis et seules sont extraites les pages référencées par des numéros dans le journal de transactions ou sur les pages d'index.

Les ensembles de pages préalablement lues pour les balayages de tables, les balayages d'index clusterisés et les contrôles dbcc sont moins fiables et peuvent engendrer des lectures inutiles. Lors des balayages séquentiels, des E/S inutiles peuvent survenir dans les cas suivants :

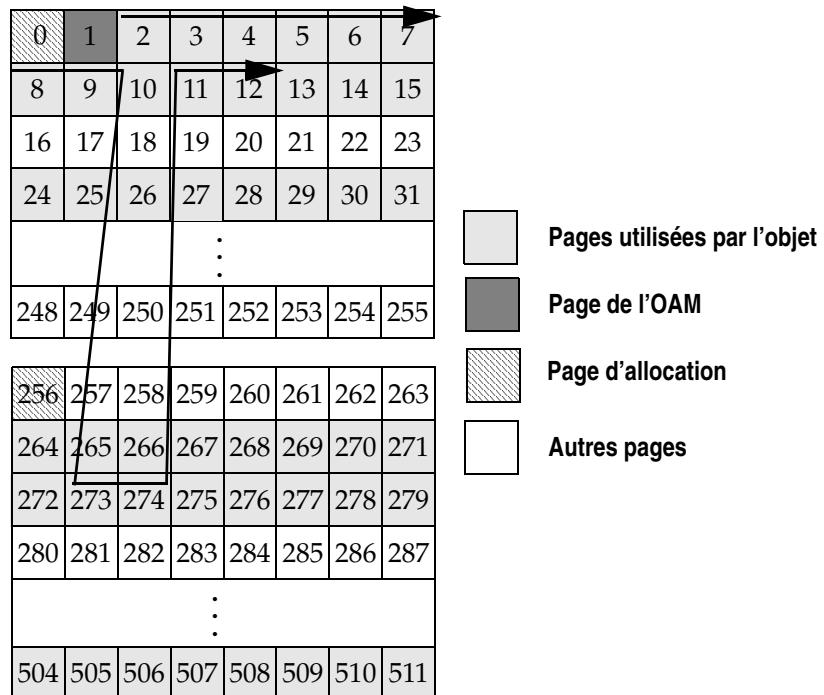
- fragmentation des chaînages de pages dans les tables verrouillées au niveau de toutes les pages (tables APL),
- usage intensif du cache par plusieurs utilisateurs.

Fragmentation des chaînages de pages

Le mécanisme d'allocation des pages d'Adaptive Server vise à regrouper les pages appartenant au même objet dans la mémoire physique, en allouant de nouvelles pages sur un extent déjà alloué à l'objet et en allouant de nouveaux extents sur les unités d'allocation qui sont déjà utilisées par l'objet.

Cependant, au fur et à mesure que les pages sont allouées et libérées, les chaînages de type page dans les tables verrouillées au niveau des pages de données seulement peuvent engendrer des anomalies. La [figure 10-1](#) montre un exemple de chaînage de pages anormal entre des extents de deux unités d'allocation.

Figure 10-1 : Chaînage de pages anormal entre des unités d'allocation



La [figure 10-1](#) montre que lorsqu'un balayage doit d'abord accéder à une page depuis l'unité d'allocation 0, il vérifie la page d'allocation et exécute des E/S asynchrones pour toutes les pages utilisées par l'objet analysé, jusqu'à la limite définie pour la zone. Au fur et à mesure que les pages se libèrent dans le cache, la requête les traite dans l'ordre, en suivant le chaînage de pages. Lorsque le balayage atteint la page 10, la page suivante dans le chaînage (page 273) appartient à l'unité d'allocation 256.

Lorsque la page 273 est requise, la page d'allocation 256 est analysée et des demandes de prélecture asynchrone sont lancées pour toutes les pages de l'unité d'allocation qui appartiennent à l'objet.

Lorsque le chaînage de pages revient sur une page de l'unité d'allocation 0, les deux cas suivants peuvent se présenter :

- Les pages extraites de l'unité d'allocation 0 figurent toujours dans le cache et la requête se poursuit sans E/S physiques inutiles.
- Les pages extraites de l'unité d'allocation 0 ont été expulsées du cache par les lectures de l'unité d'allocation 256 et par d'autres E/S générées par d'autres requêtes qui utilisent la zone. La requête doit relancer les demandes de prélecture. Cette situation peut être détectée de deux manières :
 - Le nombre de passages d'une page d'allocation à l'autre est égal à 2. Adaptive Server utilise le rapport entre ce chiffre et les pages extraites pour réduire la taille de l'ensemble de pages préalablement lues, afin de lancer moins d'E/S.
 - Le nombre de pages extraites mais inutilisées de la zone est élevé, auquel cas la prélecture asynchrone est temporairement interrompue ou réduite, conformément aux restrictions applicables à la zone.

Optimisation des objectifs de la prélecture asynchrone

Le choix de la taille des zones et des pourcentages de prélecture pour les zones de buffers est déterminant pour optimiser les performances de la prélecture asynchrone. Lorsque plusieurs applications sont simultanément actives, un système de prélecture optimisé combine les tailles de zone et les limites de prélecture pour obtenir les résultats suivants :

- amélioration du débit du système ;
- performances accrues des applications qui utilisent la prélecture asynchrone ;
- pas de dégradation des performances pour les applications qui n'utilisent pas la prélecture asynchrone.

Les modifications des tailles de zone et des limites de prélecture sont dynamiques et vous permettent de faire face à des charges de travail variées. Par exemple, vous pouvez configurer la prélecture asynchrone pour obtenir de bonnes performances lors d'une reprise ou d'un contrôle dbcc et vous pouvez la reconfigurer ultérieurement sans redémarrer Adaptive Server.

Pour de plus amples informations, reportez-vous aux sections « [Limites de reprise](#) », page 279, et « [Limites de dbcc](#) », page 280.

Commandes de configuration

Les limites de la prélecture asynchrone sont définies sous la forme d'un pourcentage de la zone dans laquelle des pages extraites mais inutilisées peuvent être stockées. La configuration peut être effectuée à deux niveaux :

- La configuration par défaut sur le serveur, à l'aide du paramètre de configuration global `async prefetch limit`. Lors de l'installation initiale, la valeur par défaut de `global async prefetch limit` est de 10 %.
Pour de plus amples informations, reportez-vous au *Guide d'administration système*.
- Un remplacement par zone, défini à l'aide de la procédure système `sp_poolconfig`. Pour vérifier les limites applicables à chaque zone, utilisez `sp_cacheconfig`.
Pour de plus amples informations, reportez-vous au *Guide d'administration système*.

Toute modification des limites de prélecture asynchrone a un effet immédiat, sans qu'il soit nécessaire de redémarrer. Les limites globales et par zone peuvent également être définies dans le fichier de configuration.

Autres fonctions d'optimisation des performances d'Adaptive Server

Cette section décrit les interactions entre la prélecture asynchrone et les autres fonctions d'Adaptive Server associées aux performances.

E/S étendues

La combinaison d'E/S étendues avec la prélecture asynchrone permet d'accélérer le traitement des requêtes avec de faibles overheads pour celles qui effectuent des balayages de tables et pour les opérations exécutées avec l'utilitaire dbcc.

Lorsque des E/S étendues exécutent une prélecture sur toutes les pages d'une unité d'allocation, le minimum d'E/S pour la totalité de l'unité est :

- 31 E/S de 16 Ko
- 7 E/S de 2 Ko pour les pages qui ont un extent commun avec la page d'allocation

Remarque Les références aux E/S étendues sont sur un serveur de pages logiques dont la taille est de 2 Ko. Si la taille des pages de votre serveur est de 8 Ko, l'unité de base de l'E/S sera de 8 Ko. Si elle est de 16 Ko, l'unité de base de l'E/S sera de 16 Ko.

Dimensions et limites applicables à une zone de 16 Ko

L'exécution de 31 prélectures asynchrones de 16 Ko limitées à 10 % des buffers de la zone requiert une zone d'au moins 310 buffers de 16 Ko. Si la zone est plus petite ou si la restriction est inférieure, des demandes de prélecture seront rejetées. Pour accroître l'activité de la prélecture asynchrone dans la zone, vous pouvez définir une zone plus grande ou accroître les limites de la prélecture asynchrone.

Lorsque plusieurs requêtes effectuent simultanément des balayages de tables par l’intermédiaire de la même zone, le nombre de pages extraites inutilisées autorisé dans la zone doit être plus élevé. Les requêtes lancent des demandes de prélecture à des intervalles irréguliers et ne sont pas au même stade lors de la lecture des pages. Par exemple, une requête vient d’extraire 31 pages et la zone comporte 31 pages inutilisées pour cette requête, tandis qu’il ne reste que 2 ou 3 pages inutilisées pour la requête précédente. Pour optimiser ces requêtes, vous devez prendre en compte la moitié du nombre de pages d’une demande de prélecture, multipliée par le nombre de requêtes actives dans la zone.

Restrictions pour la zone de 2 Ko

Les requêtes qui utilisent des E/S étendues lors des balayages séquentiels peuvent néanmoins effectuer des E/S de 2 Ko :

- Lorsqu’un balayage porte sur une nouvelle unité d’allocation, il effectue des E/S de 2 Ko sur les 7 pages de l’unité qui partagent de l’espace avec la page d’allocation.
- Si les pages de l’unité d’allocation se trouvent déjà dans la zone de 2 Ko lors du lancement des requêtes de prélecture, les pages qui se partagent l’extent doivent être lues dans la zone de 2 Ko.

Si la limite de prélecture asynchrone de la zone de 2 Ko est 0, les 7 premières lectures sont effectuées par des E/S asynchrones normales et la requête est mise en veille à chaque lecture si les pages ne sont pas dans le cache. Vous devez définir des limites suffisamment élevées pour la zone de 2 Ko pour ne pas réduire les performances de la prélecture.

Balayages de lecture-élimination (MRU)

Si un balayage utilise une stratégie de remplacement MRU (« most recently used », utilisation la plus récente), les buffers sont traités d’une manière spéciale lorsqu’ils sont lus dans le cache en prélecture asynchrone. Tout d’abord, les pages sont liées à l’extrémité MRU de la chaîne et non au marqueur de vidage. Lorsque la requête accède à la page, les buffers sont liés au marqueur de vidage dans la zone. Lors d’une utilisation intensive du cache, cette stratégie permet d’éviter l’élimination des buffers extraits liés au marqueur de vidage avant qu’ils ne soient utilisés. Elle a peu d’incidences sur les performances, sauf si la prélecture porte sur un grand nombre de pages inutilisées. Dans ce cas : les pages extraites ont tendance à expulser d’autres pages du cache.

Balayages parallèles et E/S étendues

Les requêtes parallèles peuvent solliciter davantage les zones de buffers. Si des requêtes en série sont effectuées dans les mêmes zones, il est fort probable qu'elles soient lancées à des intervalles et à des stades d'exécution différents : certaines accèdent à des pages qui se trouvent déjà dans le cache et d'autres attendent des E/S.

L'exécution en parallèle traite les zones de buffers d'une manière différente, selon le type de balayage et le degré de parallélisation. Certaines requêtes en parallèle peuvent lancer simultanément de nombreuses demandes de prélecture.

Balayages de tables reposant sur un hachage

Sur les tables APL, les balayages de tables en hachage comportent plusieurs processus de travail ayant accès au même chaînage de pages. Chaque processus vérifie l'ID de chaque page figurant dans la table, en ne tenant compte que des lignes sur les pages dont l'ID correspond à la valeur de hachage du processus de travail.

Le premier processus de travail qui a besoin d'une page provenant d'une nouvelle unité d'allocation lance une demande de prélecture asynchrone portant sur toutes les pages de cette unité. Lorsque les balayages des autres processus de travail requièrent également des pages de cette unité d'allocation, celles-ci se trouvent déjà dans les E/S ou dans le cache. Lorsque le premier balayage aborde l'unité suivante, le processus se répète.

Tant que l'un des processus de travail qui effectuent un balayage de hachage n'est pas interrompu (en attente d'un verrou, par exemple), les balayages des tables de hachage ne sollicitent pas plus les zones que les processus en série. Les processus peuvent lire les pages beaucoup plus rapidement que les processus en série ; c'est pourquoi les pages à l'état « inutilisé » passent à l'état « utilisé » plus rapidement.

Balayages reposant sur les partitions

Les balayages reposant sur les partitions ont tendance à intensifier l'utilisation des zones, car il peut arriver que plusieurs processus de travail effectuent une prélecture asynchrone sur des unités d'allocation différentes. Contrairement aux limites de prélecture par zone, il est rare que les limites d'E/S par serveur et par moteur soient atteintes sur des tables partitionnées sur plusieurs devices.

Lorsqu'une requête parallèle est analysée et compilée, elle lance des processus de travail. Si 4 processus analysent une table comportant 4 partitions, chacun de ces processus tente d'extraire toutes les pages dans la première unité d'allocation. Pour optimiser les performances de cette requête, il est préférable que la taille et les limites applicables à la zone de 16 Ko soient suffisantes pour permettre l'exécution de 124 ($31 * 4$) demandes de prélecture asynchrone. Chaque processus de travail analyse rapidement les pages dans le cache et passe à d'autres unités d'allocation, en lançant davantage de demandes de prélecture lorsque les pages sont nombreuses.

Paramètres spéciaux pour les limites de la prélecture asynchrone

Vous pouvez modifier temporairement la configuration de la prélecture asynchrone dans les cas suivants :

- la reprise,
- les opérations dbcc qui utilisent la prélecture asynchrone.

Limites de reprise

Lors d'une reprise, Adaptive Server n'utilise que la zone de 2 Ko du cache de données par défaut. Si vous arrêtez le serveur à l'aide de shutdown with nowait ou s'il s'arrête en raison d'une coupure de courant ou d'une panne, les enregistrements des journaux risquent d'être très nombreux lors de la reprise.

Pour accélérer la reprise, vous pouvez modifier le fichier de configuration en effectuant l'une des opérations suivantes ou les deux :

- Augmentez la taille de la zone de 2 Ko dans le cache de données par défaut en réduisant la taille des autres zones dans le cache.
- Augmentez la limite de prélecture pour la zone de 2 Ko.

Ces deux modifications sont dynamiques, de sorte que vous pouvez utiliser sp_poolconfig pour restaurer les valeurs d'origine après la reprise sans redémarrer Adaptive Server. Le processus de reprise permet de se connecter au serveur dès que la reprise de la base de données master est terminée. Les bases de données sont récupérées l'une après l'autre et les utilisateurs peuvent se connecter à l'une d'entre elles dès qu'elle est récupérée. Il peut y avoir des conflits si la reprise de certaines bases de données n'est pas terminée et si la zone de 2 Ko du cache de données par défaut fait l'objet d'une utilisation intensive.

Limites de dbcc

Si vous effectuez un contrôle de cohérence sur une base de données à un moment où l'activité sur le serveur est faible, vous pouvez l'accélérer en définissant des limites de prélecture asynchrone élevées pour les zones utilisées par dbcc.

dbcc checkalloc peut utiliser des buffers internes spéciaux de 16 Ko si le cache ne comporte pas de zone de 16 Ko pour la base de données appropriée. Si vous disposez d'une zone de 2 Ko au lieu d'une zone de 16 Ko, définissez une limite de prélecture locale nulle lorsque vous exécutez dbcc checkalloc. L'utilisation de la zone de 2 Ko au lieu des buffers internes de 16 Ko peut réduire les performances.

Activités de maintenance pour des performances de prélecture élevées

Dans les chaînages de pages des tables APL et dans les niveaux feuille des index, des anomalies apparaissent à mesure que des modifications de données sont effectuées dans la table. En général, les tables récemment créées comportent peu d'anomalies. Celles dans lesquelles vous avez effectué des modifications, des suppressions et des insertions ayant généré des ruptures de page, des allocations de pages et des libérations de pages, sont susceptibles de comporter des anomalies dans les chaînages de pages de différentes allocations. Lorsque vous avez modifié 10 à 20 % des lignes d'origine dans une table, vous devez vérifier si elle comporte des chaînages de pages anormaux qui réduisent l'efficacité de la prélecture asynchrone. Si c'est le cas, vous risquez de devoir recréer des index ou recharger des tables.

Elimination d'anomalies dans les tables sans index

En ce qui concerne les tables APL sans index, l'allocation des pages est généralement séquentielle, sauf lorsqu'elle est annulée par la suppression de toutes les lignes d'une page. Ces pages peuvent être réutilisées lorsqu'un espace supplémentaire est alloué à l'objet. Vous pouvez créer un index clusterisé (et le supprimer, si vous souhaitez stocker la table comme une table sans index) ou effectuer un bulkcopy pour sortir les données, tronquer la table et recopier les données dans la table. Ces deux opérations permettent de comprimer l'espace utilisé par la table et d'éliminer les anomalies de chaînage de pages.

Elimination d'anomalies dans les tables avec index clusterisés

En ce qui concerne les index clusterisés, les ruptures et les libérations de page peuvent engendrer des anomalies de chaînage de pages. La reconstitution des index clusterisés ne permet pas toujours d'éliminer les liens entre les allocations de pages différentes. Vous devez utiliser fillfactor pour réduire le nombre d'anomalies résultant des modifications des données dans l'index clusterisé, où vous prévoyez une croissance.

Elimination d'anomalies dans les index non clusterisés

Si une combinaison de requêtes utilise des balayages d'index restreints, vous pouvez améliorer les performances de la prélecture asynchrone en annulant et en recréant des index non clusterisés, lorsque le chaînage de pages au niveau feuille est fragmenté.

Analyse des performances et prélecture asynchrone

Le résultat de statistics io indique le nombre de lectures physiques effectuées par la prélecture asynchrone et le nombre de lectures effectuées par les E/S asynchrones normales. De plus, statistics io indique le nombre de fois où la recherche d'une page dans le cache a abouti avec la prélecture asynchrone sans verrou d'attente.

Pour de plus amples informations, reportez-vous à la section « [Statistiques relatives aux E/S physiques et logiques](#) », page 67 du guide *Performances et optimisation : contrôle et analyse*.

Le rapport sp_sysmon comprend des informations sur la prélecture asynchrone dans les sections « Gestion du cache des données » et « Gestion des E/S disque ».

Si vous utilisez sp_sysmon pour évaluer les performances de la prélecture asynchrone, vous pouvez observer des améliorations dans d'autres domaines :

- accroissement des taux de présence dans le cache, dans les zones où la prélecture asynchrone est efficace ;
- réduction des changements de contexte en raison des non-présences dans le cache, avec un accroissement des arrêts volontaires ;
- réduction éventuelle des conflits de verrouillage. Les tâches verrouillent les pages pendant l'exécution d'E/S pour la page suivante nécessaire à la requête. Si cette durée est raccourcie en raison d'un accroissement du taux de présence dans le cache, la durée du verrouillage est réduite en conséquence.

Pour de plus amples informations, reportez-vous aux sections « [Gestion des caches de données](#) », page 294 et « [Gestion des E/S disque](#) », page 316 du guide *Performances et optimisation : contrôle et analyse*.

Bases de données temporaires multiples

Ce chapitre traite des bases de données temporaires multiples.

Sujet	Page
Après la création d'une base de données temporaire	287
Utilisation de sp_tempdb	288
Liaison à des bases de données temporaires	289
Bases de données temporaires multiples dans le système	290
Remarques concernant l'installation	303

Présentation

Adaptive Server vous permet de créer et de gérer des bases de données temporaires multiples en plus de la base de données système tempdb, qui était la seule base temporaire dans les versions précédentes d'Adaptive Server.

Ces bases de données temporaires multiples, également appelées **tempdbs**, réduisent les conflits au niveau des catalogues système et des journaux de la base de données système tempdb. Elles vous permettent de :

- créer des bases de données temporaires sur des devices à accès rapide ;
- supprimer une base de données temporaire pour récupérer de l'espace de stockage ;
- partitionner des tâches qui créent des objets temporaires dans des tempdbs spécifiques, afin d'éviter que ces tâches n'interfèrent avec d'autres sessions ayant besoin d'espace dans les bases de données temporaires.

La fonction de bases de données temporaires multiples est activée pour :

- les nouvelles installations ;
- les installations mises à niveau depuis une version d'Adaptive Server antérieure à la version 12.5 ;
- les bases de données chargées depuis une version d'Adaptive Server antérieure à la version 12.5.

La base de données tempdb est la base de données temporaire créée par le système. Avant la version 12.5.0.3 d'Adaptive Server, tempdb était la seule base de données temporaire du serveur. Des tables temporaires et de travail sont créées dans tempdb.

Adaptive Server vous permet de créer des bases de données temporaires multiples, que vous pouvez ensuite utiliser pour créer des objets temporaires, tels que des tables temporaires privées et des tables de travail. Les administrateurs système des bases de données peuvent lier (c'est-à-dire créer des associations entre) le login « sa » et des applications à des bases de données temporaires spécifiques ou au groupe de bases de données **default** à l'aide de `sp_tempdb`. Le groupe **default** est un groupe créé par le système et qui compte toujours au moins la base de données système tempdb en tant que membre. Vous pouvez ajouter d'autres bases de données temporaires à ce groupe.

Remarque Vous ne pouvez pas lier explicitement des objets à tempdb.

Une application liée au groupe peut se voir affecter n'importe quelle base de données temporaire du groupe selon un mode d'attribution circulaire.

Remarque Les groupes d'utilisateurs et le groupe de bases de données temporaires **default** ne sont pas liés.

Bases de données temporaires utilisateur

Les bases de données temporaires utilisateur sont créées par l'utilisateur, qui est le plus souvent l'administrateur de la base de données. Elles sont généralement générées pour limiter les conflits de ressources (tels que les conflits de catalogues système et de journaux) dans la tempdb système. Ces bases de données ont de nombreux points communs avec la base de données système tempdb dans la mesure où elles sont :

- principalement utilisées pour créer des objets temporaires ;
- recréées, plutôt que restaurées, lors d'un processus de restauration du système.

Tous les objets présents dans une base de données temporaire avant un arrêt ou une interruption de l'activité sont perdus au moment de la restauration car les bases temporaires sont écrasées à l'aide de la base de données model. Ces restrictions sont valables tant pour la base de données système tempdb que pour les bases de données temporaires utilisateur. Pour de plus amples informations, reportez-vous à la section « [Annulation et restauration](#) », [page 292](#).

Contrairement à la tempdb système, vous pouvez supprimer des bases de données temporaires utilisateur.

Bases de données temporaires et liaisons

Lors de la connexion, les sessions sont assignées à une base de données temporaire en fonction des liaisons existantes actives :

- Si la liaison se fait à une base de données temporaire spécifique qui est en ligne et disponible, la session lui est attribuée.
- Si la liaison se fait au groupe default, une base de données temporaire de ce groupe est sélectionnée selon un mode d'attribution circulaire.
- Si aucune liaison n'est spécifiée, une base de données temporaire est sélectionnée dans le groupe default.

La base de données temporaire choisie pour une session reste active pendant toute la durée de la session et ne change jamais, même en cas de modification des liaisons.

Une fois qu'une base de données temporaire est affectée à une session, tous les objets temporaires créés au cours de cette session sont intégrés dans cette base de données. Ils sont supprimés de manière implicite lors de l'arrêt de la session ou du serveur. De la même manière, les tables temporaires partageables sont supprimées implicitement lors de l'arrêt du serveur.

Remarque Les tables temporaires peuvent être supprimées explicitement par la session.

Des tables temporaires serveur ou partageables continuent d'être créées dans latempdb système si elles sont qualifiées en tant que « tempdb..server temptab » afin d'inclure le nom de la base de données et de la table.

Les applications existantes qui transmettent des informations entre les sessions à l'aide de tables temporaires partageables continuent ainsi de fonctionner.

Les nouvelles applications peuvent toutefois utiliser des tempdbs utilisateur pour créer des tables temporaires partageables.

Tables temporaires privées

Les tables temporaires privées sont créées par session et utilisent le symbole « # » au début de leur nom (par exemple, #pubs). Elles ne peuvent pas être partagées par plusieurs sessions. Elles résident, en compagnie des tables de travail, dans la base de données temporaire affectée à la session. Il existe deux types de tables temporaires privées, qui se différencient également par leur portée visible et par leur durée de vie implicite :

- Table temporaire de session : créée au niveau du batch en dehors d'une procédure, elle est :
 - visible pour toutes les commandes, y compris les procédures exécutées dans la session à l'origine de la création de la table ;
 - supprimée de manière implicite lors de l'interruption de la session.

L'instruction `create` suivante, exécutée au niveau du batch, crée une table temporaire privée :

```
create table #t1(id int, desc varchar(250))
```

- Table temporaire procédurale : créée dans le cadre d'une procédure, elle est :
 - visible pour la procédure qui la crée et toutes les procédures imbriquées qu'elle appelle ;
 - supprimée de manière implicite lors de la fermeture de la procédure qui l'a créée.

L'instruction suivante crée deux tables temporaires procédurales :

```
create procedure SetupTempTables as  
    create table #pt1( . . . )  
    create table #pt2( . . . )
```

Les applications peuvent créer des tables temporaires partageables dans des bases de données temporaires utilisateur exactement de la même manière qu'elles le font dans la base de données système tempdb. Des processus de coordination peuvent communiquer via ces tables.

Remarque Les tables temporaires procédurales peuvent également être supprimées de façon explicite.

Pour créer des tables temporaires privées ou y accéder, les procédures stockées doivent passer par la base de données temporaire affectée à la session.

Tables temporaires partageables

Des tables temporaires partageables peuvent être créées dans des bases de données temporaires utilisateur ou dans la base de données système tempdb. Elles peuvent être partagées par plusieurs sessions et sont supprimées de manière implicite lors de la réinitialisation du serveur.

Remarque Contrairement à la base de données système tempdb, les bases de données temporaires utilisateur peuvent être supprimées. Toute application dépendante d'une base de données temporaire utilisateur supprimée cessera de fonctionner si cette base contenait des tables temporaires partagées.

Après la création d'une base de données temporaire

L'ID dbid d'une nouvelle base de données temporaire est automatiquement enregistrée dans une liste globale de toutes les bases temporaires disponibles. Vous ne pourrez lier des objets à cette base qu'après son enregistrement dans la liste globale. Lors du redémarrage d'un serveur, les bases de données temporaires sont ajoutées à la liste globale au fur et à mesure de leur restauration.

Le nombre de bases de données temporaires pris en charge est statique et n'est pas configurable. Le nombre de bases de données temporaires pouvant être enregistrées dans la liste globale des bases disponibles pour des liaisons (et, par conséquent, pour l'affectation à des sessions) est de 512, tempdb compris.

Lorsque la liste globale est pleine, toute tentative d'ajout d'une base de données temporaire se solde par un avertissement. Pour afficher le contenu de la liste, exécutez la commande dbcc pravailabletempdbs. Pour de plus amples informations, reportez-vous à la section « [dbcc pravailabletempdbs](#) », page 301.

Vous pouvez créer une base de données même s'il est impossible d'enregistrer son dbid dans la liste globale. Si la liste globale est pleine, vous pouvez exécuter la commande dbcc addtempdb afin d'ajouter le dbid à la liste dès qu'une place se libère. Pour plus d'informations sur la commande dbcc addtempdb, reportez-vous à la section « [dbcc addtempdb](#) », page 302.

En outre, si une place se libère dans la liste globale à la suite de la suppression d'une base de données temporaire, vous pouvez également :

- supprimer et recréer la base de données temporaire, après quoi celle-ci sera enregistrée dans l'espace disponible dans la liste et disponible pour la liaison ;
- redémarrer le serveur.

Remarque La base de données système tempdb, dont le dbid est 2, est enregistrée dans la liste globale lors du redémarrage du serveur. Son enregistrement ne peut pas être annulé.

Adaptive Server part du principe que les bases de données temporaires que vous créez ne sont pas liées au groupe default. Pour ajouter une nouvelle base de données à ce groupe, utilisez `sp_tempdb` (reportez-vous à l'entrée de `sp_tempdb` dans le *Manuel de référence : procédures stockées* pour de plus amples informations). Lorsqu'une base de données est ajoutée au groupe, elle est directement disponible pour l'affectation en mode d'attribution circulaire.

Même si la base de données ne fait pas partie du groupe default, vous pouvez toujours l'affecter à une session via une liaison d'application ou de login. Reportez-vous à l'option bind dans l'entrée de `sp_tempdb` du *Manuel de référence : procédures stockées* pour de plus amples informations.

Utilisation de `sp_tempdb`

`sp_tempdb` permet aux utilisateurs de :

- créer le groupe de bases de données temporaires default ;
- lier des bases de données temporaires au groupe default ;
- lier le login « sa » et des applications au groupe default ou à des bases de données temporaires spécifiques.

La syntaxe de la commande `sp_tempdb` est la suivante :

```
sp_tempdb [  
    [{ create | drop }, nom_groupe] |  
    [{ add | remove }, nom_base_temp, nom_groupe] |  
    [{ bind, type_objet, nom_objet, type_liaison, objet_liaison [, portée,  
        rigidité ] } |  
        { unbind, type_objet, nom_objet [, portée ] } ] |  
    [ unbindall_db, nom_base_temp ] |  
    [ show [, "all" | "gr" | "db" | "login" | "app" [, nom ] ] ] |  
    [ who, nom_base ]  
    [ help ] ]
```

Pour obtenir des informations détaillées sur les paramètres et leur utilisation, reportez-vous au *Manuel de référence : procédures stockées*.

Liaison à des bases de données temporaires

Les conditions suivantes sont nécessaires à l'aboutissement de la commande sp_tempdb bind :

Si	Alors
<i>type_objet</i> est login_name	<i>nom_objet</i> doit être un nom de connexion correct et <i>portée</i> doit être NULL.
<i>type_objet</i> est application	<i>nom_objet</i> doit être un nom d'application et <i>portée</i> doit être NULL.

Si	Alors
<i>type_liaison</i> est group	<i>objet_liaison</i> doit être le nom du groupe existant avec lequel vous établissez la liaison (dans ce cas-ci, default).
<i>type_liaison</i> est database	<i>objet_liaison</i> doit être le nom d'une base de données temporaire existante. Il ne peut pas s'agir de tempdb dans la mesure où tempdb ne peut pas avoir de liaisons explicites.

Lorsque la commande sp_tempdb bind est exécutée avec succès, elle ajoute une nouvelle entrée dans sysattributes pour représenter cette liaison.

S'il existe déjà une entrée pour la combinaison *nom_objet/type_objet/portée* spécifiée dans la commande sp_tempdb bind, l'entrée dans sysattributes est mise à jour avec les nouvelles informations fournies par la combinaison *type_liaison* et *objet_liaison* donnée.

Bien que la nouvelle liaison créée entre directement en vigueur, toute session à laquelle une base de données temporaire est déjà affectée maintient l'affectation initiale. Seules les nouvelles sessions sont concernées par la nouvelle liaison.

Remarque Vous pouvez modifier le nom d'une application via ct_lib et d'autres interfaces telles que jConnect, même après avoir connecté et démarré une session. Cela ne modifie en rien l'affectation de la base de données temporaire à la session. Cela vaut aussi pour la commande setuser.

Liaison d'un login « sa » à sa propre base de données temporaire

Vous pouvez lier le login « sa » à une base de données temporaire distincte à des fins de maintenance ou de restauration après incident. En isolant l'utilisateur « sa » des activités de la base de données temporaire d'autres applications et utilisateurs, vous lui permettez d'avoir accès aux ressources de la base chaque fois que nécessaire.

Liaison d'une session

Au moment de la connexion, une session est affectée à une base de données temporaire, qui reste active pendant toute la durée de la session et ne peut pas être modifiée. Les liaisons sont lues depuis `sysattributes` et sont choisies en fonction des paramètres suivants :

- S'il existe une liaison de type LG (login), utilisez-la.
- S'il existe une liaison de type AP (nom d'application), utilisez-la.
- Liez la session à une base de données temporaire du groupe `default`.

Les liaisons peuvent être strictes ou souples :

- Liaisons souples : les connexions n'échouent jamais malgré les possibles échecs de l'affectation d'une base de données temporaire à la session sur la base de la liaison active. Si rien ne marche, une session finira toujours par être affectée à la `tempdb` système.
- Liaisons strictes : si l'affectation d'une base de données temporaire à une session ne peut pas être effectuée sur la base de la liaison active, la connexion échoue.

Bases de données temporaires multiples dans le système

Cette section décrit les implications de la fonction de bases de données temporaires multiples sur Adaptive Server.

Modifications des tables système

La fonction de bases de données temporaires multiples affecte les tables `sysattributes` et `sysdatabases`.

sysattributes

Le [tableau 11-1](#) montre la représentation des groupes de bases de données temporaires et des liaisons tels qu'ils apparaissent dans la table système `sysattributes`. Seules les colonnes pertinentes sont présentées. Toutes les autres ont la valeur NULL. Les groupes sont représentés dans des lignes dont la valeur de l'attribut est 0. Les liaisons de login et d'applications, ainsi que les liaisons base de données-groupe, sont représentées dans les lignes dont la valeur de l'attribut est 1.

Tableau 11-1 : Représentation de sysattributes

class	attribute	object_type	object_cinfo	object	object_cinfo1	int_value	char_value
16	0	GR	nom de groupe	NULL	NULL	ID de groupe	NULL
16	0	D	nom de base de données	ID de groupe	NULL	NULL	NULL
16	1	LG	NULL	ID d'utilisateur	0 pour couple, 1 pour stricte	0 pour base de données, 1 pour groupe	nom de base de données ou de groupe
16	1	AP	Application Name	NULL	0 pour couple, 1 pour stricte	0 pour base de données, 1 pour groupe	nom de base de données ou de groupe

sysdatabases

`sysdatabases` prend en charge un nouvel octet dans le champ `status3`. L'état temporaire d'une base de données est indiqué par la valeur 0x00000100 (256 décimales) dans le champ `status3` d'une entrée `sysdatabases`.

Variable globale @@tempdbid

La variable `@@tempdbid` renvoie l'ID de la base de données temporaire (`dbid`) affectée à la session.

Exemples

Exemple 1 Renvoie le `dbid` de `mytempdb`, la base de données temporaire affectée à la session, qui est 7 :

```
select @@tempdbid from mytempdb
7
```

Exemple 2 Renvoie le nom de la base de données temporaire, qui est mytempdb :

```
select db_name(@@tempdbid)
mytempdb
```

Fonction **tempdb_id()**

La fonction **tempdb_id()** affiche la base de données temporaire à laquelle une session donnée est affectée. L'entrée de la fonction **tempdb_id()** est un ID de processus serveur et sa sortie est la base de données temporaire à laquelle le processus est affecté. Si vous ne fournissez pas de processus serveur, **tempdb_id()** renvoie le dbid de la base de données temporaire affectée au processus en cours.

Pour retrouver tous les processus serveur affectés à une base de données temporaire spécifique, exécutez l'instruction suivante :

```
select spid from master..sysprocesses
where tempdb_id(spid) = db_id("tempdatabase")
```

Remarque select **tempdb_id()** donne le même résultat que
select **@@tempdbid**.

Troncature du journal

Adaptive Server tronque les bases de données temporaires utilisateur de la même manière que le journal **tempdb**. Lorsque vous exécutez le processus de point de reprise du système sur une base de données temporaire, le journal **tempdb** et la base de données temporaire utilisateur sont tronqués à cause de l'option **trunc log on chkpt**. Ce n'est vrai que lorsque c'est le système, et non l'utilisateur, qui lance le point de reprise.

Annulation et restauration

Le processus de restauration des bases de données temporaires utilisateur est considérablement différent de celui des bases de données standard.

Il n'y a pas de différence au niveau des annulations d'exécution entre la base de données système **tempdb** et les bases de données temporaires utilisateur.

Le processus de restauration au redémarrage est similaire à celui de tempdb. Une base de données temporaire utilisateur est créée à l'aide des entrées disponibles dans sysusages et la base de données model est copiée dedans. Tous les objets utilisateur présents dans la base de données temporaire avant la fermeture sont perdus.

Les bases de données temporaires sont restaurées dans l'ordre dans lequel elles apparaissent dans la table sysdatabases. Utilisez sp_dbrecovery_order pour définir un ordre différent.

Les sessions qui se connectent avant la restauration de la base de données temporaire à laquelle elles auraient normalement dû être affectées sont attribuées à une autre base temporaire, sauf en cas d'existence d'une liaison stricte, auquel cas la connexion échoue. Pour minimiser cet impact, utilisez sp_dbrecovery_order pour spécifier la restauration de certaines bases de données temporaires, voire de toutes, avant le reste des bases de données utilisateur.

Lorsque la restauration se déroule correctement, une base de données temporaire est ajoutée à la liste globale des bases temporaires disponibles, ainsi qu'au groupe default, en cas de liaison à celui-ci.

La date et l'heure de création de l'entrée de la base de données temporaire dans sysdatabases correspondent au moment où la base a été recréée. Cette date est mise à jour lors de la restauration, chaque fois que la base de données temporaire est recréée.

La base de données model est verrouillée pendant la restauration du système, de sorte qu'elle peut être copiée dans la base de données temporaire.

Remarque Dans la mesure où la base model est verrouillée, vous ne pouvez pas créer de nouvelle base de données ou utiliser la commande use model tant que le processus de restauration n'est pas terminé.

Suppression d'une base de données temporaire

Vous ne pouvez supprimer une base de données temporaire que dans les cas suivants :

- Aucune liaison n'est associée à la base.
- Aucune session active ne lui est affectée.

Cela signifie que la base de données :

- ne doit pas faire partie du groupe de bases de données temporaires default ;
- ne doit pas être liée à des logins ou des applications.

Si des liaisons existent, drop database signale une erreur. Supprimez la base de données du groupe default et les liaisons aux logins et applications. Utilisez l'interface sp_tempdb "show" pour retrouver toutes les liaisons impliquant la base de données temporaire.

Si des sessions actives sont connectées à la base de données supprimée, drop database échoue. Des sessions de ce type peuvent exister si elles ont été instanciées avant que ne soient supprimées les liaisons de la base de données. Pour pouvoir poursuivre le processus de suppression, l'utilisateur doit soit attendre que toutes les sessions actives soient purgées du système, soit les supprimer. Utilisez sp_tempdb "who" nom_base pour connaître les sessions actuellement connectées à une base de données temporaire spécifique.

Suppression d'autres bases de données

Pour pouvoir supprimer une base de données, aucune classe Java de la base de données supprimée ne doit être référencée par des objets présents dans des bases de données temporaires. De telles références peuvent exister à la suite de commandes select into #temptable. Dans ces cas-là, les colonnes d'une table temporaire font référence à des classes de la base de données source.

Le système balaye toutes les bases de données afin de s'assurer qu'aucune référence de ce type n'existe. S'il en trouve, la base de données n'est pas supprimée.

Remarque Ce comportement est cohérent avec celui d'un Adaptive Server dépourvu de la fonction de bases de données temporaires multiples.

Alter database

Adaptive Server effectue une copie de données pour certaines opérations alter database. En général, la journalisation du contenu des pages n'est pas effectuée pour ces opérations. C'est la raison pour laquelle le serveur transfère des pages vers le disque afin de pouvoir restaurer ces modifications après une interruption d'activité de la base de données. Cette opération n'est toutefois pas nécessaire dans le cas de tempdb, dans la mesure où le contenu de cette base de données est toujours recréé lors du processus de restauration.

Les bases de données temporaires utilisateur sont également recréées lors de la restauration. Par conséquent, ces bases sont traitées d'une manière comparable à tempdb. Les explications suivantes décrivent certains cas spécifiques où cela se vérifie pour alter database. En outre, il existe d'autres cas de alter database où le comportement des bases de données temporaires utilisateur est identique à celui de tempdb.

Extension d'une base de données temporaire

Lorsque la base de données système tempdb est étendue, les nouvelles pages de base de données ne sont pas mises à zéro. Seules les pages d'allocation sont écrites sur le disque. Les bases de données temporaires utilisateur sont cohérentes avec cette approche.

Extension de la base de données *model*

La base de données model ne peut pas être plus grande que la plus petite base de données temporaire car model est copiée dans la base de données temporaire lors du redémarrage du système. Lorsque vous tentez de modifier la taille de model, le serveur lance un contrôle pour vérifier que la nouvelle taille n'est pas supérieure à celle de la plus petite base de données temporaire. Si une base plus petite est trouvée, la commande alter database signale une erreur.

Caractéristiques de la mise en mémoire cache

Les caches des bases de données temporaires d'un groupe doivent être configurés de manière similaire du point de vue des caractéristiques de mise en mémoire cache. Lors de la sélection d'un plan d'exécution des requêtes efficace, l'optimiseur évalue les caractéristiques de mise en mémoire cache d'une base de données. Si le plan fait partie d'une procédure, il peut être réutilisé par une autre session affectée à une base de données temporaire différente. Si les caractéristiques de mise en mémoire cache des deux bases varient considérablement, les performances risquent d'être fort réduites.

Liaison de bases de données temporaires utilisateur à un cache de données

Utilisez sp_bindcache pour lier une base de données à un cache de données. La liaison d'une base de données temporaire utilisateur se fait de la même manière que pour les autres bases utilisateur, sauf par rapport au point suivant :

Une base de données temporaire utilisateur est considérée comme en cours d'utilisation tant qu'une session lui est affectée, même si aucune activité n'est en cours pour le compte de la session active. Cependant, pour modifier la liaison du cache de la base de données, celle-ci doit être verrouillée exclusivement. Vous ne pouvez pas verrouiller la base de données tant qu'une session active lui est affectée. Les étapes nécessaires pour résoudre ce problème sont similaires à celles requises pour supprimer une base de données.

❖ **Modification de la liaison du cache d'une base de données**

- 1 Utilisez sp_tempdb pour supprimer toutes les liaisons impliquant la base de données temporaire, y compris les liaisons base de données-groupe default, ainsi que les liaisons d'applications et de login à la base de données.
- 2 Attendez que toutes les sessions actives déjà affectées à la base de données temporaire soient purgées ou interrompez-les. Utilisez sp_tempdb pour répertorier les sessions actives affectées à la base de données.
- 3 Passez à la liaison base de données-cache.
- 4 Restaurez les liaisons supprimées à la première étape.

Traitement des procédures stockées

Dans certains cas, le dbid d'un objet temporaire est remappé vers l'ID de la base de données temporaire de la session en cours pour s'assurer que ces objets sont créés et accessibles dans la base de données temporaire correcte. Par exemple :

- Lorsqu'une procédure stockée est créée par une session attachée à une base de données temporaire et est ensuite compilée par une autre session dans une autre base temporaire.
- Lorsqu'une procédure stockée est compilée par une session différente de la base de données temporaire de la session qui l'exécute.

Le remappage des ID de bases de données temporaires ne permet toutefois pas de garantir que la procédure se comportera de la même manière dans les différentes bases temporaires. Une procédure compilée dans une base de données temporaire avec certaines caractéristiques de mise en mémoire cache peut se comporter de manière très différente lorsqu'elle est exécutée dans une base temporaire avec d'autres caractéristiques de mise en mémoire cache. De la même manière, une procédure compilée dans une base de données temporaire avec certains paramètres dboption risque d'avoir une sémantique très différente en cas d'exécution dans une base avec d'autres paramètres dboptions.

Optimisation de l'écriture de tempdb

Les bases de données temporaires ne sont pas restaurables car Adaptive Server les supprime et les recrée lors de sa réinitialisation. Adaptive Server en profite pour retarder l'écriture de buffers de données ou de journal.

Dans des bases de données non temporaires normales, si vous utilisez une commande telle que `select` qui n'est pas journalisée, Adaptive Server sauvegarde les données sur le disque à des fins de restauration.

Dans les versions 12.5.0.3 et ultérieures, Adaptive Server n'effectue pas cette sauvegarde pour les bases de données temporaires. Cela signifie que si vous utilisez une commande telle que l'instruction `select into` suivante avec une base de données temporaire, Adaptive Server n'impose pas l'écriture de buffers de données sur le disque :

```
select * into tempdb..temp_table from foo
```

En outre, l'exécution de commandes telles que `insert`, `update` et `delete` sur des bases de données temporaires n'oblige pas Adaptive Server à écrire un journal à la fin de l'opération de validation. Par exemple, la commande `insert into` suivante génère moins de modifications de contexte, une charge plus légère sur les devices de journaux ou de données et un débit plus rapide :

```
insert into tempdb..temp_table select * from foo
```

Considérations en matière de haute disponibilité

Les sections suivantes abordent la question des bases de données temporaires multiples dans une configuration haute disponibilité.

Configuration haute disponibilité

Dans les versions d'Adaptive Server dépourvues de bases de données temporaires multiples, il ne pouvait pas y avoir de bases de données utilisateur dans le compagnon secondaire lors de la configuration haute disponibilité initiale. L'introduction de bases de données temporaires utilisateur permet ce qui suit :

- Le serveur secondaire peut avoir des bases de données temporaires utilisateur tant que celles-ci portent des noms uniques.
- Le serveur secondaire peut avoir des bases de données temporaires utilisateur dont le dbid est en conflit avec un dbid du serveur primaire tant que les dbids en conflit concernent des bases de données temporaires sur les deux serveurs.

Les recommandations de Sybase sont les suivantes :

- Chargez les bases de données utilisateur lors de la configuration haute disponibilité. De cette façon, des dbid uniques sont générés pour les bases de données temporaires du serveur secondaire et les utilisateurs peuvent les supprimer et les reconfigurer sans devoir supprimer les bases de données temporaires utilisateur.
- Installez des liaisons d'applications et de login spécifiques sur le compagnon secondaire pour les bases de données temporaires créées sur celui-ci et susceptibles d'être utilisées par l'application du serveur primaire en cas de reprise sur le serveur secondaire. Cependant, si vous n'installez pas de liaison, l'application de serveur en mode reprise sur le serveur secondaire utilise les bases de données temporaires du groupe default du compagnon secondaire, ce qui a peu d'impact sur les performances.

Prise en charge des bases de données proxy

Aucune base de données proxy n'est créée pour les bases de données temporaires multiples et ce, pour les raisons suivantes :

- Dans le cas des bases de données temporaires utilisateur, les bases de données doivent être recréées, ce qui affecte les performances de la reprise sur le serveur secondaire et du retour au serveur primaire.
- Sans forcer le montage, les utilisateurs peuvent déployer des disques/RAM haute performance et des disques locaux pour des bases de données temporaires qui ne sont pas nécessairement à double accès.
- Dans la mesure où aucun proxy haute disponibilité n'est créé, la comptabilisation de l'espace qui évalue l'espace nécessaire au succès de la configuration n'inclut pas les bases de données temporaires utilisateur.

Scénarios de reprise sur le serveur secondaire

Lors de la reprise sur le serveur secondaire, Adaptive Server ne monte pas les bases de données temporaires utilisateur. Si Adaptive Server autorise la création de bases de données utilisateur (si l'option set proxy n'est pas activée) en mode reprise sur le serveur secondaire, il interdit par contre les bases de données temporaires utilisateur.

Comportement du mode compagnon normal

Dans le mode compagnon normal, il est possible de créer des bases de données temporaires utilisateur en suivant les mêmes règles que pour les bases de données utilisateur (par exemple, elles doivent avoir des noms uniques).

En mode configuration, aucune base de données proxy n'est créée pour des bases de données temporaires multiples, même si l'option `with_proxydb` a été utilisée.

Montage/démontage

En mode reprise sur le serveur secondaire, aucune base de données temporaire n'est montée. Un client qui passe en mode reprise sur un serveur secondaire suit le processus de connexion normal et se voit attribuer une nouvelle base de données temporaire en fonction des liaisons existantes dans le compagnon secondaire.

Sauvegarde et chargement des bases de données temporaires

Vous pouvez sauvegarder les bases de données temporaires, mais cette opération ne sert à rien. Gardez les considérations suivantes à l'esprit :

- Ces bases de données sont recréées chaque fois que le serveur est redémarré et les noms d'objet des tables temporaires sont générés en interne et sont propres à la session. Cela signifie qu'un même utilisateur peut se connecter au serveur via deux sessions isql différentes et créer une table temporaire #t1. Un nom généré en interne est créé pour chaque instance de #t1. Ce nom unique est stocké dans la table sysobjects de la base de données temporaire. De cette façon, plusieurs sessions actives peuvent avoir leur propre table temporaire portant le même nom.
- La base de données système tempdb créée dans un serveur de base de données pré-multiple peut être sauvegardée, mais pas chargée.
- Les bases de données temporaires utilisateur peuvent être sauvegardées, mais pas chargées.

Procédure stockée **sp_dboption**

Toutes les bases de données temporaires d'un groupe doivent utiliser la même procédure dboptions.

Si une partie des options suivantes diffèrent, les applications risquent de ne pas fonctionner correctement. Par exemple, si une base de données temporaire d'un groupe, appelée mtdb1, active cette option, tandis qu'une autre base du même groupe, appelée mtdb2, ne le fait pas, la procédure suivante, qui crée une table et insère une ligne en laissant une valeur de champ en dehors, ne fonctionnera pas de la même façon dans les deux bases temporaires :

```
create procedure P1 as
    create table #t1 (c1 int, str char(250))
    insert #t1 values (1)
go
```

L'instruction insert dans mtdb1 a réussi parce que des valeurs NULL sont autorisées, contrairement à mtdb2, où l'instruction échoue.

Configuration du nombre de bases de données ouvertes

Chaque fois que vous créez une base de données temporaire, incrémentez la valeur de configuration « bases de données ouvertes » d'une unité.

Lorsqu'une session est affectée à une base de données temporaire, la ressource qui représente la base de données est conservée et marquée comme étant en cours d'utilisation de manière à ne pas pouvoir être réutilisée dans la session en cours.

Procédures modifiées

Plusieurs procédures stockées ont été modifiées afin de pouvoir être utilisées avec les bases de données temporaires multiples :

- sp_helpdb indique désormais si une base de données est ou non une base de données temporaire utilisateur. Cette indication apparaît sous la colonne status.

```
sp_helpdb "mytempdb3"
```

name	db_size	owner	dbid	created	status
mytempdb	32.0 MB	sa	7	Dec 12, 2001	select into/bulkcopy/pllsort, trunc log on chkpt, user created temp db

- `sp_bindcache` a été étendue afin d'empêcher la liaison de tables individuelles à un cache nommé dans des bases de données temporaires utilisateur.
- `sp_doption` a été étendue afin d'empêcher les bases de données temporaires utilisateur d'être paramétrées sur le mode mono-utilisateur. Par ailleurs, lorsque vous tentez de modifier les options d'une base de données temporaire, un avertissement s'affiche car les options de toutes les bases de données temporaires doivent être cohérentes.
- `sp_dropuser` a été étendue afin d'empêcher la suppression d'un utilisateur guest d'une base de données temporaire utilisateur.

Procédures inchangées mais néanmoins importantes

Bien que les procédures suivantes n'aient pas été modifiées, elles sont significatives pour les bases de données temporaires :

Nom de la procédure	Description
<code>sp_changedbowner</code>	Vous pouvez modifier l'appartenance des bases de données temporaires utilisateur, ce qui est impossible avec les bases de données système, dont <code>tempdb</code> .
<code>sp_defaultloc</code>	Vous ne pouvez pas mapper les bases de données système, et notamment <code>tempdb</code> , vers un emplacement par défaut, contrairement aux bases de données temporaires utilisateur.
<code>sp_renamedb</code>	Vous ne pouvez pas renommer les bases de données système, y compris <code>tempdb</code> , contrairement aux bases de données temporaires utilisateur. Vous devez d'abord vérifier que les liaisons existantes de la base de données temporaire renommée ont été supprimées, puis les recréer à l'aide de la nouvelle base.

Commandes dbcc modifiées et nouvelles commandes dbcc

Plusieurs nouvelles commandes dbcc ont été ajoutées.

dbcc pravailabletempdbs

`dbcc pravailabletempdbs` imprime la liste globale des bases de données temporaires disponibles.

Exemple

```
1> dbcc pravailabletempdbs
2> go
```

```
Available temporary databases are:  
Dbid: 2  
Dbid: 4  
Dbid: 5  
Dbid: 6  
Dbid: 7  
DBCC execution completed. If DBCC printed error  
messages, contact a user with System Administrator (SA)  
role.
```

dbcc addtempdb

dbcc addtempdb ajoute une base de données temporaire à la liste globale.
La syntaxe de cette commande est la suivante :

```
dbcc addtempdb( id_base | nom_base )
```

Si la base de données n'existe pas ou n'est pas temporaire, une erreur est générée. Si elle est déjà reprise dans la liste, un message d'information s'imprime.

Autres modifications

Les limites du gestionnaire de ressources en matière d'utilisation des bases de données temporaires continuent d'être d'application pour les bases de données temporaires utilisateur, de même que pour tempdb.

La libération à grande échelle n'est pas consignée pour les bases de données temporaires.

Le Replication Agent n'est jamais démarré pour une base de données temporaire.

Requêtes parallèles

Les threads enfants pour les requêtes parallèles sont affectés à la même base de données temporaire que le parent.

Transactions portant sur plusieurs bases de données

Une transaction portant sur plusieurs bases de données tant temporaires que non temporaires ne peut pas être lancée dans une base de données temporaire, dans la mesure où il est impossible de restaurer une telle transaction. Cette restriction ne concerne cependant pas les transactions portant sur plusieurs bases de données qui sont uniquement temporaires car celles-ci sont recréées chaque fois que le serveur est redémarré.

Remarques concernant l'installation

Une nouvelle ligne correspondant au groupe de bases de données temporaires default est ajoutée à la table sysattributes dans le cadre de la mise à niveau vers un serveur 12.5, ainsi que lors de la création d'un nouveau device master.

Si vous utilisez déjà un serveur 12.5 et qu'aucune mise à niveau n'est dès lors nécessaire, vous pouvez ajouter le groupe de bases de données temporaires default en exécutant la commande `sp_tempdb create "default"`. Pour plus d'informations sur la procédure stockée, reportez-vous à l'entrée de `sp_tempdb` dans le *Manuel de référence : procédures stockées*.

Taille et configuration de bases de données temporaires pour des applications

Les exigences en matière de ressources et d'espace pour les bases de données temporaires varient d'une application à l'autre. Ces bases doivent avoir la même taille sauf si vous :

- comprenez parfaitement vos besoins en matière de ressources et d'espace ;
- gérez des liaisons applications-base de données et de groupe de sorte que les affectations de la base de données répondent aux besoins en matière de ressources et d'espace.

En attribuant la même taille à toutes les bases de données temporaires, vous devriez pouvoir exécuter des applications sans tomber à court de ressources ou d'espace, quelle que soit la base de données affectée à une instance donnée d'une application ou d'une session.

Configurez également les caches de tempdb d'un groupe de la même manière pour garantir à un plan d'exécution des requêtes donné des performances équivalentes, quelle que soit la tempdb utilisée.

Dans des configurations haute disponibilité, configurez les propriétés de tempdb pour les serveurs primaire et secondaire de la même manière.

Tables temporaires partageables

Vous pouvez créer des tables temporaires partageables dans les bases de données temporaires utilisateur et dans la tempdb système. Les applications existantes qui utilisent la tempdb lors de la création de tables temporaires partageables continueront de créer ces tables dans la tempdb système. Cela permet aux applications de coordination et aux sessions qui communiquent via les tables partageables de la base de données temporaire de continuer de fonctionner comme avant.

Les nouvelles applications peuvent créer leurs tables temporaires partageables dans une base de données temporaire utilisateur. Celles-ci fonctionnent de la même manière que celles créées dans la tempdb. Autrement dit, elles sont accessibles à d'autres sessions et sont conservées jusqu'au redémarrage du serveur, sauf si vous les supprimez de manière explicite.

Remarque Les applications qui utilisent des bases de données temporaires utilisateur ne fonctionneront pas si les bases en question sont supprimées.

Mises à jour des procédures stockées créées par l'utilisateur

Vous devez modifier toutes les procédures stockées existantes créées par l'utilisateur qui considèrent que les tables temporaires sont toujours dans la tempdb.

Par exemple, une procédure stockée qui vérifie dans le catalogue sysobjects de la tempdb l'existence d'une table temporaire privée n'est plus correcte dans la mesure où une telle table existe dans la base de données temporaire affectée, qui peut être tempdb. L'exemple suivant illustre ce cas :

```
select db_name(@@tempdbid)
go
-----
a_tempdb1
(1 row affected)

create table #t1 (c1 int)
go
```

#t1 n'est pas trouvé dans le catalogue sysobjects de la base de données système tempdb :

```
use tempdb
select name from sysobjects where name like "#%"
name
-----
(0 rows affected)
```

L'entrée se trouve en fait dans le catalogue de la tempdb qui lui est affectée.

```
declare @tempdb_name varchar(32)
select @tempdb_name = db_name(@@tempdbid)
use @tempdb_name
go
(1 row affected)

select name from sysobjects where name like "#%"
go
name
-----
#t1_____00000270012069406
(1 row affected)
```

Retour à une version antérieure

Cette section explique comment passer d'une version 12.5.0.3 d'Adaptive Server dans laquelle la fonction de bases de données temporaires multiples est activée à la version 12.5.0.2.

Remarque Le passage de la version 12.5.0.3 à une version antérieure à 12.5.0.2 n'est pas pris en charge.

- ❖ **Rétrogradation à partir d'un Adaptive Server sur lequel la fonction tempdb multiple est activée**
 - 1 Utilisez sp_tempdb pour supprimer toutes les liaisons.
 - 2 Supprimez toutes les bases de données temporaires utilisateur.
 - 3 Eteignez le serveur.
 - 4 Démarrlez la version 12.5.0.2 d'Adaptive Server.

- 5 Utilisez `installmaster` et `instmsgs.ebf` qui faisait partie de la version 12.5.0.2 d'Adaptive Server pour réinstaller les procédures stockées et les messages système.

Le passage de la version 12.5.0.3 à la version 12.5.0.2 d'Adaptive Server ne requiert aucune étape supplémentaire.

La version 12.5.0.2 a été modifiée afin de prendre en charge la redéfinition dynamique d'arborescences à partir de texte lorsque celles-ci ont été normalisées à l'aide de bases de données temporaires utilisateur dans la version 12.5.0.3.

Performances de tempdb

Ce chapitre aborde les exigences en matière de performances associées à l'utilisation de la base de données tempdb. Cette dernière permet à tout utilisateur de créer des objets. De nombreux processus l'utilisent sans que vous en soyez informé. Il s'agit d'une ressource à l'échelle du serveur, utilisée principalement pour le traitement interne des tris, la création de tables de travail, le reformatage et le stockage temporaire des tables et des index créés par les utilisateurs.

De nombreuses applications utilisent des procédures stockées créant des tables dans tempdb pour accélérer les jointures complexes ou exécuter d'autres analyses de données complexes, difficiles à exécuter en une seule étape.

Sujet	Page
Incidence de la gestion de tempdb sur les performances	307
Types et utilisations de tables temporaires	308
Allocation initiale de tempdb	310
Taille de tempdb	311
Positionnement de tempdb	312
Libération du device master de segments de tempdb	313
Liaison de tempdb à son propre cache de données	314
Tables temporaires et verrouillage	315
Réduction de l'activité de journalisation dans tempdb	316
Optimisation des tables temporaires	317

Incidence de la gestion de tempdb sur les performances

La gestion de tempdb influe grandement sur les performances globales d'Adaptive Server ; tempdb ne doit pas être négligée ou laissée dans un état par défaut. Il s'agit de la base de données la plus dynamique sur de nombreux serveurs et il convient de lui accorder une attention particulière.

Avec une bonne stratégie d'anticipation, la plupart des problèmes liés à tempdb peuvent être évités. En revanche, si tempdb n'est pas correctement placée ou dimensionnée, vous pouvez rencontrer les problèmes suivants :

- tempdb se remplit fréquemment, ce qui génère des messages d'erreur et constraint les utilisateurs à soumettre à nouveau leurs requêtes après libération d'espace.
- Le tri s'opère lentement et les utilisateurs ne s'expliquent pas pourquoi leurs requêtes présentent des performances si irrégulières.
- Les requêtes des utilisateurs sont momentanément dans l'incapacité de créer des tables temporaires parce que des verrous sont posés sur les tables système.
- L'utilisation massive d'objets tempdb évacue les autres pages du cache de données.

Principales solutions pour améliorer les performances de tempdb

Les solutions ci-après peuvent facilement être mises en oeuvre :

- dimensionnez correctement tempdb pour toutes les activités d'Adaptive Server ;
- déterminez l'emplacement optimal de tempdb pour limiter les risques de conflit ;
- liez tempdb à son propre cache de données ;
- minilmisez le verrouillage des ressources dans tempdb.

Types et utilisations de tables temporaires

La manière dont sont utilisées les tables temporaires définies par l'utilisateur influe grandement sur les performances globales d'Adaptive Server et de vos applications.

Les tables temporaires sont très utiles car elles réduisent souvent le volume de travail du serveur. Cependant, elles peuvent également accroître les besoins en taille de tempdb. Certaines tables sont véritablement temporaires et d'autres se révèlent être permanentes.

tempdb est utilisée pour trois types de tables :

- les tables véritablement temporaires,
- les tables utilisateur normales,
- les tables de travail.

Tables véritablement temporaires

Pour créer une table véritablement temporaire, choisissez « # » comme premier caractère de son nom :

```
create table #temptable (...)
```

ou :

```
select select_list
      into #temptable ...
```

Les tables temporaires :

- existent uniquement pour la durée de la session utilisateur ou pour la durée d'exécution de la procédure qui les a créées ;
- ne peuvent pas être partagées entre plusieurs connexions utilisateur ;
- sont automatiquement supprimées à la fin de la session ou de la procédure (elles peuvent également être supprimées manuellement).

Lorsque vous créez des index sur des tables temporaires, ceux-ci sont stockés dans tempdb :

```
create index tempix on #temptable(col1)
```

Tables utilisateur normales

Pour créer des tables utilisateur normales dans tempdb, indiquez le nom de la base de données dans la commande créant la table :

```
create table tempdb..temptable (...)
```

ou :

```
select select_list
      into tempdb..temptable
```

Les tables utilisateur régulières dans tempdb

- peuvent être conservées d'une session à l'autre ;
- peuvent être utilisées par des opérations bulkcopy ;
- peuvent être partagées si des autorisations leur sont associées ;
- doivent être explicitement supprimées par le propriétaire (sinon, elles le sont lorsqu'Adaptive Server est redémarré).

Vous pouvez créer des index dans tempdb sur des tables temporaires persistentes :

```
create index tempix on tempdb..temptable(col1)
```

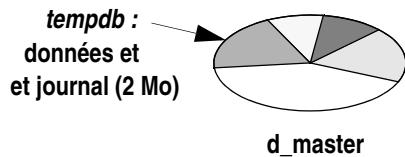
Tables de travail

Les tables de travail sont automatiquement créées dans tempdb par Adaptive Server pour les jointures par fusion, les tris et autres processus serveur internes. Ces tables :

- ne sont jamais partagées,
- disparaissent en fin d'exécution de la commande.

Allocation initiale de tempdb

Lors de l'installation d'Adaptive Server, tempdb a une taille de 2 Mo et est intégralement située sur le device master, comme le montre la [figure 12-1](#). Il s'agit généralement de la première base de données qu'un administrateur système doit modifier. Plus le nombre d'utilisateurs sur le serveur est élevé, plus la taille de tempdb doit être importante. Vous pouvez la modifier sur le device master ou sur d'autres devices. Selon vos besoins, vous pouvez répartir (striping) tempdb sur plusieurs devices.

Figure 12-1 : Allocation par défaut de tempdb

Utilisez `sp_helpdb` pour afficher la taille et l'état de tempdb. L'exemple suivant montre les valeurs par défaut de tempdb à l'installation :

sp_helpdb tempdb					
name	db_size	owner	dbid	created	status
tempdb	2.0 MB	sa	2	May 22, 1999	select into/bulkcopy

device_frag	size	usage		free	kbytes
master	2.0 MB	data and log		1248	

Taille de tempdb

tempdb doit avoir une taille suffisante pour gérer les processus ci-après pour chaque utilisateur concurrent d'Adaptive Server :

- tables de travail pour les jointures par fusion,
- tables de travail créées pour distinct, group by et order by, pour le reformatage, pour la stratégie OR et pour la matérialisation de certaines vues ou sous-requêtes,
- tables temporaires (créées avec « # » en tant que premier caractère du nom),
- index associés aux tables temporaires,
- tables utilisateur régulières dans tempdb,
- procédures constituées par SQL dynamique.

Certaines applications produisent de meilleures performances si vous utilisez des tables temporaires pour fractionner des jointures multitable. Cette stratégie est souvent mise en oeuvre :

- lorsque l'optimiseur ne choisit pas un plan d'exécution de requête approprié pour une requête joignant plus de 4 tables ;
- lorsque les requêtes qui joignent un grand nombre de tables ;
- lorsque les requêtes particulièrement complexes ;
- lorsque les applications qui doivent filtrer les données lors d'une étape intermédiaire.

Vous pouvez également utiliser tempdb pour :

- dénormaliser plusieurs tables en un petit nombre de tables temporaires ;
- normaliser une table dénormalisée, de façon à effectuer un traitement par agrégat.

Pour la plupart des applications, définissez la taille de tempdb sur 20 à 25 % de la taille de vos bases utilisateur afin de disposer de suffisamment d'espace pour les utiliser.

Positionnement de *tempdb*

Conservez tempdb sur des disques physiques distincts des bases de données de vos applications stratégiques. Utilisez les disques les plus rapides possibles. Si votre plate-forme gère les mémoires auto-alimentées et que l'utilisation de votre tempdb constitue un goulet d'étranglement pour vos applications, utilisez ce type de mémoire. Après avoir réparti tempdb sur des devices supplémentaires, vous pouvez libérer les segments system, default et logsegment du device master.

Bien qu'il soit possible de placer tempdb sur le même device que la base de données master, il est en général préférable d'utiliser des devices distincts. N'oubliez pas que les devices logiques sont mis en miroir (selon le mécanisme proposé par Adaptive Server), contrairement aux bases de données. Si vous mettez le device master en miroir, vous créez un miroir de toutes les parties des bases de données résidant sur le device master. Si le miroir utilise des écritures serial, les performances risquent de diminuer, notamment si la base de données tempdb est très utilisée.

Libération du device master de segments de tempdb

Par défaut, l'allocation de 2 Mo aux segments system, default et logsegment pour tempdb est définie sur le device master. Lorsque vous allouez de nouveaux devices à tempdb, ils font automatiquement partie intégrante de ces trois segments. Si vous allouez un second device à tempdb, vous pouvez libérer les segments default et logsegment du device master. De cette façon, vous êtes certain que les tables que travail et autres tables temporaires dans tempdb ne créeront pas de conflit sur le device master.

Pour libérer les segments du device master :

- 1 Si ce n'est déjà fait, transférez tempdb sur un autre device.
Par exemple :

```
alter database tempdb on tune3 = 20
```

- 2 Exécutez une commande `use tempdb`, puis libérez les segments du device master :

```
sp_dropsegment "default", tempdb, master
sp_dropsegment system, tempdb, master
sp_dropsegment logsegment, tempdb, master
```

- 3 Pour vérifier que l'allocation du segment default n'est plus définie sur le device master, exécutez la commande suivante :

```
select dbid, name, segmap
from sysusages, sysdevices
where sysdevices.low <= sysusages.size + vstart
and sysdevices.high >= sysusages.size + vstart -1
and dbid = 2
and status & 2 = 2
```

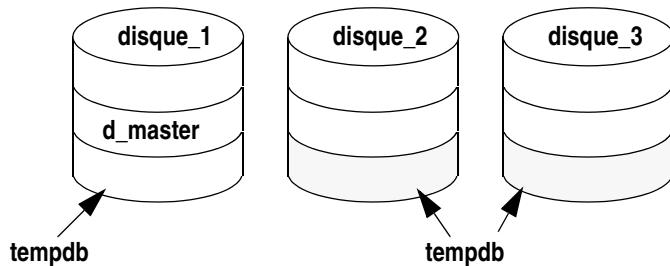
La colonne segmap doit indiquer « 1 » pour toute allocation sur le device master, ce qui signifie que seul le segment system utilise encore le device :

dbid	name	segmap
2	master	1
2	tune3	7

Utilisation de disques pour améliorer les performances des requêtes parallèles

Si tempdb s'étend sur plusieurs devices, comme illustré dans la figure 12-2, vous pouvez tirer parti des performances de la requête parallèle pour certaines tables ou tables de travail temporaires.

Figure 12-2 : Extension de tempdb sur des disques



Liaison de tempdb à son propre cache de données

Dans les conditions normales d'utilisation d'Adaptive Server, tempdb se sert très souvent du cache de données pour les opérations de création, d'alimentation et de suppression des tables temporaires.

L'affectation de tempdb à son propre cache de données permet :

- d'éviter que l'activité sur les objets temporaires n'évacue les autres objets du cache de données par défaut ;
- de répartir les E/S sur plusieurs caches.

Pour de plus amples informations, reportez-vous à la section « [Etude des besoins en cache pour tempdb](#) », page 253.

Commandes pour lier le cache

A l'aide des commandes `sp_cacheconfig` et `sp_poolconfig`, vous pouvez créer des caches de données nommés et configurer des zones d'une taille donnée pour les E/S étendues. Seul un administrateur système peut configurer des caches et des zones.

Remarque Les références aux E/S étendues sont sur un serveur de pages logiques dont la taille est de 2 Ko. Si la taille des pages de votre serveur est de 8 Ko, l'unité de base de l'E/S sera de 8 Ko. Si elle est de 16 Ko, l'unité de base de l'E/S sera de 16 Ko.

Pour toute instruction sur la configuration des zones et caches nommés, reportez-vous à la section *Guide d'administration système*.

Une fois les caches configurés et le serveur redémarré, vous pouvez lier tempdb au nouveau cache :

```
sp_bindcache "tempdb_cache", tempdb
```

Tables temporaires et verrouillage

La création ou la suppression de tables temporaires et de leurs index peut entraîner des conflits de verrous sur les tables système de la base tempdb. Lorsque des utilisateurs créent des tables dans tempdb, les informations les concernant doivent être stockées dans des tables système telles que sysobjects, syscolumns et sysindexes. Si plusieurs processus utilisateur créent et suppriment des tables dans tempdb, de nombreux conflits peuvent se produire sur les tables système. Les tables de travail créées en interne ne stockent pas d'informations dans les tables système.

Si les conflits au niveau des tables système de tempdb constituent un problème pour les applications qui doivent fréquemment créer et supprimer le même jeu de tables temporaires, créez ces tables au démarrage de l'application. Remplissez-les avec `insert...select` et supprimez toutes les lignes de données avec `truncate table`. Même si `insert...select` requiert une journalisation et est plus lente que `select into`, elle peut apporter une solution au problème de verrouillage.

Réduction de l'activité de journalisation dans tempdb

Même si l'option de base de données trunc log on checkpoint est activée sur tempdb, les modifications apportées à tempdb restent écrites dans le journal des transactions. Vous pouvez réduire l'activité de journal dans tempdb :

- en utilisant select into au lieu de create table et insert ;
- en sélectionnant uniquement les colonnes dont vous avez besoin dans les tables temporaires.

Utilisation de select into

Lorsque vous créez et remplissez des tables temporaires dans tempdb, utilisez, chaque fois que possible, la commande select into de préférence à create table et insert...select. Dans tempdb, l'option de base de données select into/bulkcopy est activée par défaut pour permettre ce mode de fonctionnement.

Les opérations select into sont plus rapides dans la mesure où elles ne font l'objet que d'une journalisation minimale. Seule l'allocation des pages de données est analysée, et non les modifications effectivement apportées à chaque ligne de données. En revanche, chaque insertion de données dans une requête insert...select est intégralement consignée, d'où une augmentation de l'overhead.

Utilisation de lignes plus courtes

Si l'application créant des tables dans tempdb n'utilise qu'un petit nombre de colonnes d'une table, vous pouvez réduire le nombre et la taille des enregistrements du journal de transactions en :

- sélectionnant uniquement les colonnes requises pour l'application, au lieu d'utiliser select * dans des requêtes insérant des données dans les tables ;
- limitant les lignes sélectionnées strictement au nombre de lignes requis par les applications.

En outre, ces deux options permettent également de limiter la taille des tables elles-mêmes.

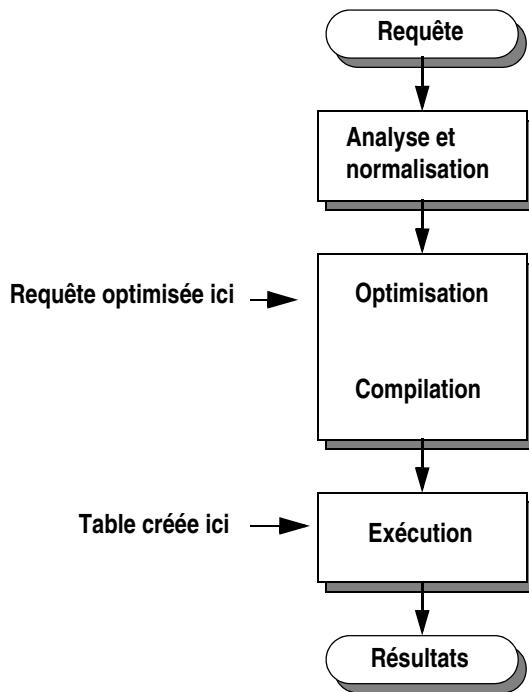
Optimisation des tables temporaires

La plupart des utilisations de tables temporaires sont simples et rapides et ne nécessitent qu'une faible optimisation. Toutefois, si vos applications requièrent des accès multiples aux tables dans tempdb, il est conseillé de les analyser en détail pour élaborer, le cas échéant, une stratégie d'optimisation. Généralement, cela implique de séparer les processus de création et d'indexation de l'accès à la table, à l'aide de plusieurs procédures ou batchs. Les conditions suivantes sont requises :

- La table doit contenir des données lorsque l'index est créé. En effet, si vous créez la table temporaire et construisez l'index sur une table vide, Adaptive Server ne crée pas de statistiques de colonne telles que les histogrammes ou les densités. Si vous insérez des lignes de données après avoir créé l'index, l'optimiseur dispose de statistiques incomplètes.
- L'optimiseur peut être amené à retenir une solution non optimale si des lignes ont été ajoutées ou supprimées après la création de l'index ou l'exécution de update statistics.

Lorsque vous créez une table dans la procédure stockée ou le batch dans lequel elle est utilisée, l'optimiseur de requêtes ne peut pas déterminer sa taille puisqu'elle n'est pas encore créée au moment de l'optimisation de la requête, comme illustré dans la [figure 12-3](#). Cela s'applique aux tables temporaires et aux tables utilisateur normales.

Figure 12-3 : Optimisation et création de tables temporaires



L'optimiseur part donc du principe que ces tables comptent 10 pages de données et 100 lignes. Or, si la table est vraiment grande, ce postulat peut induire l'optimiseur à retenir un plan non optimal d'exécution de requête.

Les deux techniques suivantes permettent d'améliorer l'optimisation des tables temporaires :

- création d'index sur les tables temporaires,
- fractionnement des utilisations complexes des tables temporaires en plusieurs batchs ou procédures, de façon à fournir des informations à l'optimiseur.

Création d'index sur les tables temporaires

Vous pouvez définir des index sur des tables temporaires. Le plus souvent, ceux-ci permettent d'améliorer les performances des requêtes utilisant tempdb. L'optimiseur les utilise comme des index sur les tables utilisateur ordinaires. Les conditions suivantes sont requises :

- La table doit contenir des données lorsque l'index est créé. En effet, si vous créez la table temporaire et construisez l'index sur une table vide, Adaptive Server ne crée pas de statistiques de colonne, telles que les histogrammes ou les densités. Si vous insérez des lignes de données après avoir créé l'index, l'optimiseur dispose de statistiques incomplètes.
- L'index doit déjà exister au moment où la requête qui l'utilise est optimisée. En effet, vous ne pouvez pas créer un index pour l'utiliser ensuite dans une requête du même batch ou de la même procédure.
- L'optimiseur peut être amené à retenir une solution non optimale si des lignes ont été ajoutées ou supprimées après la création de l'index ou l'exécution de update statistics.

Le fait de fournir un index à l'optimiseur peut contribuer à améliorer les performances de manière très importante, notamment dans le cas de procédures complexes qui créent des tables temporaires et exécutent de nombreuses opérations sur ces tables.

Création de procédures imbriquées avec des tables temporaires

Pour créer les procédures décrites ci-dessus, une étape supplémentaire est encore nécessaire. En effet, vous ne pouvez pas créer `base_proc` tant que `select_proc` n'existe pas et vous ne pouvez pas créer `select_proc` tant que la table temporaire n'existe pas. Suivez donc la procédure suivante :

- 1 Créez la table temporaire en dehors de la procédure. Elle peut être vide ; il suffit en effet qu'elle existe et qu'elle comporte des colonnes compatibles avec `select_proc` :

```
select * into #huge_result from ... where 1 = 2
```

- 2 Créez la procédure `select_proc`, comme indiqué ci-dessus.
- 3 Supprimez `#huge_result`.
- 4 Créez la procédure `base_proc`.

Fractionnement des utilisations de tempdb en plusieurs procédures

La requête suivante, par exemple, pose des problèmes d'optimisation avec `#huge_result` :

```
create proc base_proc
as
    select *
        into #huge_result
        from ...
    select *
        from tab,
        #huge_result where ...
```

Vous pouvez améliorer les performances en utilisant deux procédures. Lorsque la procédure `base_proc` appelle la procédure `select_proc`, l'optimiseur peut déterminer la taille de la table :

```
create proc select_proc
as
    select *
        from tab, #huge_result where ...
create proc base_proc
as
    select *
        into #huge_result
        from ...
    exec select_proc
```

Si le traitement de `#huge_result` requiert plusieurs accès, jointures ou autres processus tels qu'une itération avec `while`, le fait de créer un index sur `#huge_result` peut améliorer les performances. Créez l'index dans `base_proc`, afin qu'il soit accessible lors de l'optimisation de `select_proc`.

Curseurs et performances

Ce chapitre traite des problèmes de performances liés aux curseurs. Les curseurs sont des dispositifs permettant d'accéder aux résultats d'une instruction select SQL ligne par ligne (ou plusieurs lignes à la fois si vous utilisez set cursors rows). Etant donné que les curseurs utilisent un modèle différent du SQL à orientation ensembliste ordinaire, la manière dont les curseurs utilisent la mémoire et maintiennent les verrous influence les performances de vos applications. Les problèmes de performances liés aux curseurs concernent le verrouillage au niveau page et table, les ressources réseau et l'overhead des instructions de traitement.

Sujet	Page
Définition	321
Ressources nécessaires à chaque étape	324
Modes de curseur	327
Utilisation d'index et conditions requises pour les curseurs	328
Comparaison des performances réalisées avec et sans curseur	330
Verrouillage à l'aide de curseurs en lecture seule	334
Niveaux d'isolement et curseurs	335
Tables sans index partitionnées et curseurs	336
Conseils d'optimisation des curseurs	336

Définition

Un curseur est un nom symbolique associé à une instruction select. Il permet d'accéder ligne par ligne aux résultats d'une instruction select. La [figure 13-1](#) représente un curseur accédant à la table authors.

Figure 13-1 : Exemple de curseur

Curseur avec select * from authors where state = 'KY'	Groupe de résultats			
	A978606525	Marcello	Duncan	KY
	A937406538	Carton	Nita	KY
	A1525070956	Porczyk	Howard	KY
	A913907285	Bier	Lane	KY

La programmation peut :

- analyser une ligne
- exécuter une action avec les valeurs de ligne

Un curseur peut être considéré comme un « pointeur » sur le jeu de résultats d'une instruction select. Il vous permet d'examiner et parfois de manipuler une ligne à la fois.

Programmation à orientation ensembliste ou séquentielle

SQL a été conçu comme un langage à orientation ensembliste. Adaptive Server est extrêmement efficace lorsqu'il fonctionne en mode à orientation ensembliste. Les curseurs sont requis par les normes SQL ANSI et sont très efficaces dans les cas où ils sont employés. Cependant, ils peuvent diminuer les performances.

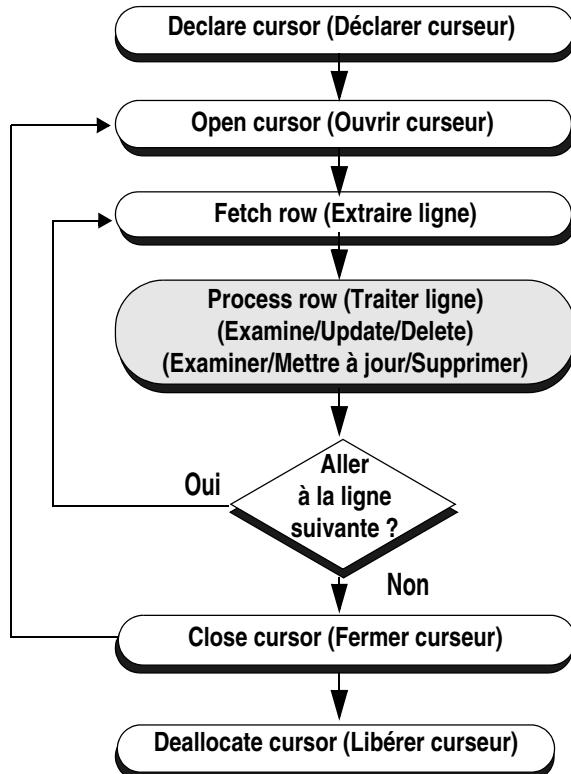
Par exemple, cette requête effectue une action identique sur toutes les lignes remplissant la condition définie dans la clause where :

```
update titles
    set contract = 1
  where type = 'business'
```

L'optimiseur effectue la mise à jour de la manière la plus performante possible. En revanche, un curseur examine chaque ligne et effectue des mises à jour qui ne portent que sur une ligne si les conditions sont remplies. L'application déclare un curseur pour une instruction select, l'ouvre, extrait une ligne, la traite, passe à la ligne suivante, et ainsi de suite. Il se peut que l'application effectue des opérations différentes en fonction des valeurs de la ligne courante et l'utilisation globale des ressources du serveur pour l'application curseur risque d'être moins efficace que les opérations au niveau ensemble du serveur. Cependant les curseurs offrent une plus grande flexibilité que la programmation à orientation ensembliste.

La [figure 13-2](#) illustre les étapes suivies lors de l'utilisation des curseurs. L'intérêt des curseurs est d'aboutir à l'étape centrale, où l'utilisateur ou un code d'application étudie une ligne et décide de l'action à effectuer en fonction de ses valeurs.

Figure 13-2 : Organigramme du curseur



Exemple

Voici un exemple simple de curseur avec l'étape de traitement de la ligne décrite ci-dessus en pseudo-code :

```

declare biz_book cursor
for select * from titles
  where type = 'business'
go
open biz_book
  
```

```
go
fetch biz_book
go
/* Look at each row in turn and perform
** various tasks based on values,
** and repeat fetches, until
** there are no more rows
*/
close biz_book
go
deallocate cursor biz_book
go
```

En fonction du contenu de la ligne, l'utilisateur peut supprimer la ligne courante :

```
delete titles where current of biz_book
```

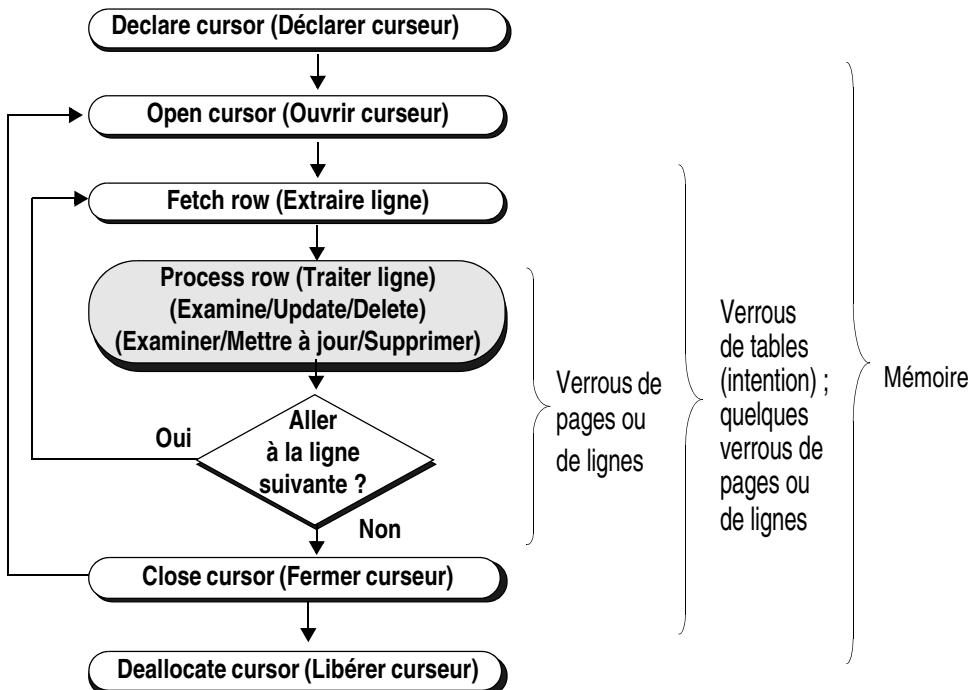
ou la mettre à jour :

```
update titles set title="The Rich
Executive's Database Guide"
where current of biz_book
```

Ressources nécessaires à chaque étape

Ces curseurs utilisent de la mémoire et requièrent des verrous sur les tables, les pages de données et les pages d'index. Lorsque vous ouvrez un curseur, vous lui allouez de la mémoire afin de stocker le plan d'exécution de requête qui est généré. Pendant que le curseur est ouvert, Adaptive Server pose des verrous d'intention sur la table et parfois sur des lignes ou sur des pages. La [figure 13-3](#) indique la durée de ces verrous lors des opérations de curseur.

Figure 13-3 : Utilisation des ressources par instruction du curseur



Les descriptions des ressources mémoire dans la [figure 13-3](#) et le [tableau 13-1](#) font référence à des curseurs adaptés aux requêtes envoyées par le biais d'isql ou de Client-Library™. Pour les autres types de curseur, les verrous sont identiques mais l'allocation de mémoire et sa libération diffèrent en fonction du type de curseur, comme indiqué à la section « [Utilisation de la mémoire et exécution des curseurs](#) », page 327.

Tableau 13-1 : Verrous et utilisation de la mémoire pour les curseurs client de Client-Library et isql

Commande du curseur	Utilisation des ressources
declare cursor	Lorsque vous déclarez un curseur, Adaptive Server n'utilise que la quantité de mémoire nécessaire pour stocker le texte de la requête.
open	Lorsque vous ouvrez un curseur, Adaptive Server lui alloue de la mémoire afin de stocker le plan d'exécution de requête qui est généré. Le serveur optimise alors la requête, parcourt les index et définit les variables mémoire. Le serveur n'accède pas encore aux lignes, sauf s'il a besoin de construire des tables de travail. Cependant, il configure les verrous nécessaires au niveau de la table (verrous d'intention). Le fonctionnement des verrous de lignes et de pages dépend du niveau d'isolement, de la configuration du serveur et du type de requête. Pour de plus amples informations, reportez-vous à la section <i>Guide d'administration système</i> .
fetch	Lorsque vous exécutez une instruction fetch, Adaptive Server extrait la ou les lignes requises, lit les valeurs spécifiées dans les variables du curseur ou envoie la ligne au client. Si le curseur a besoin de poser un verrou sur des lignes ou des pages, ce verrou est maintenu jusqu'à ce qu'une instruction fetch déplace le curseur ou jusqu'à ce que le curseur soit fermé. Selon le mode d'écriture du curseur, il peut s'agir d'un verrou partagé ou de mise à jour.
close	Lorsque vous fermez un curseur, Adaptive Server libère les verrous et une partie de la mémoire allouée. Vous pouvez, le cas échéant, ouvrir à nouveau le curseur.
deallocate cursor	Lorsque vous annulez l'allocation d'un curseur, Adaptive Server libère les ressources mémoire restantes utilisées par le curseur. Pour réutiliser le curseur, vous devez le déclarer à nouveau.

Utilisation de la mémoire et exécution des curseurs

Les descriptions de declare cursor et deallocate cursor dans le tableau 13-1 font référence à des curseurs ad hoc, envoyés par isql ou Client-Library. D'autres curseurs allouent de la mémoire d'une manière différente :

- Pour les curseurs déclarés *sur* les procédures stockées, seule une petite quantité de mémoire est allouée à l'exécution de declare cursor. Ces curseurs sont envoyés par le biais de Client-Library ou du pré-compilateur et sont appelés curseurs d'exécution.
- Pour les curseurs déclarés *au sein* d'une procédure stockée, la mémoire est déjà disponible pour cette procédure et l'instruction declare n'en requiert pas davantage.

Modes de curseur

Il existe deux modes de curseur : en lecture seule et de mise à jour. Comme l'indiquent leurs noms, les curseurs en lecture seule affichent uniquement les données d'une instruction select alors que les curseurs de mise à jour peuvent servir à effectuer des mises à jour et des suppressions positionnées.

Le mode lecture seule utilise des verrous partagés de pages ou de lignes. Si read committed with lock a la valeur 0 et que la requête s'exécute au niveau d'isolement 1, ce mode utilise des verrous momentanés et ne pose pas de verrous de pages ou de lignes jusqu'à l'extraction suivante.

Le mode lecture seule est activé lorsque vous spécifiez for read only ou lorsque l'instruction de sélection du curseur utilise distinct, group by, union ou des fonctions d'agrégat et, dans certains cas, la clause order by.

Le mode mise à jour utilise des verrous de pages ou de lignes qui permettent des mises à jour. Il est actif lorsque :

- vous spécifiez for update ;
- l'instruction select n'inclut pas distinct, group by, union, une sous-requête, des fonctions d'agrégat ou la clause at isolation read uncommitted ;
- vous spécifiez shared.

Si *liste_noms_colonnes* est spécifiée, seules ces colonnes peuvent être mises à jour.

Pour plus d'informations sur le verrouillage durant le traitement du curseur, reportez-vous au *Guide d'administration système*.

Lorsque vous déclarez le curseur, indiquez son mode. Si l'instruction `select` inclut certaines options, le curseur ne peut pas être mis à jour, même si vous le déclarez à cet effet.

Utilisation d'index et conditions requises pour les curseurs

Lorsqu'une requête est utilisée dans un curseur, elle est susceptible de choisir d'autres index que ceux employés pour la même requête hors du curseur.

Tables verrouillées au niveau de toutes les pages

Pour les curseurs en lecture seule, les requêtes au niveau d'isolement 0 (lectures de données modifiées) requièrent un index unique. Les curseurs en lecture seule au niveau d'isolement 1 ou 3 doivent générer le même plan d'exécution de requête qu'une instruction `select` employée hors d'un curseur.

Les critères d'indexation pour les curseurs modifiables signifient que ces derniers peuvent utiliser d'autres plans de requête que les curseurs en lecture seule. Les curseurs de mise à jour doivent répondre aux critères d'indexation suivant :

- Si le curseur n'est pas déclaré en vue d'une mise à jour, il est préférable d'utiliser un index unique plutôt qu'un balayage de la table ou un index non unique.
- Si le curseur est déclaré en vue d'une mise à jour *sans* liste `for update of`, un index unique est nécessaire pour les tables APL. En l'absence d'index unique, une erreur se produit.

- Si le curseur est déclaré en vue d'une mise à jour avec la liste `for update of`, vous ne pouvez choisir qu'un index unique *sans* aucune colonne de la liste pour une table APL. Une erreur se produit si aucun index unique ne correspond à la requête.

Lorsque des curseurs sont utilisés, tout index contenant une colonne `IDENTITY` est considéré unique, même si l'index a été déclaré différemment. Dans certains cas, il convient d'ajouter des colonnes `IDENTITY` aux index pour qu'ils soient uniques, sinon l'optimiseur devra choisir un plan d'exécution de requête moins performant pour une requête avec curseur.

Tables verrouillées au niveau des pages de données seulement

Les tables DOL utilisent des ID de ligne fixe pour positionner les balayages de curseur évitant ainsi de recourir aux index uniques pour les lectures de données modifiées ou les curseurs modifiables. La seule raison qui explique l'existence de plans d'exécution de requête différents dans des curseurs modifiables est qu'Adaptive Serveur utilise des balayages de tables lorsque des colonnes d'index exploitables figurent dans la liste `for update of`.

Balayages de tables pour éviter le problème Halloween

Le problème Halloween est une anomalie de mise à jour qui se produit lorsqu'un client utilisant un curseur met à jour une colonne de la ligne du jeu de résultats et que cette colonne définit l'ordre dans lequel les lignes sont renvoyées. Par exemple, si un curseur doit utiliser un index sur `last_name, first_name`, et mettre à jour une de ces colonnes, la ligne peut apparaître une seconde fois dans le jeu de résultats.

Pour éviter ce problème dans les tables DOL, Adaptive Server choisit un balayage de table lorsque la liste d'une clause `for update` contient des colonnes renvoyées par un index qui, en d'autres circonstances, serait exploitable.

Les curseurs implicitement modifiables déclarés sans clause `for update` et les curseurs dont la liste de colonnes spécifiée dans `for update` est vide risquent de rencontrer le problème Halloween en cas de mise à jour d'une colonne dans l'index qu'ils utilisent.

Comparaison des performances réalisées avec et sans curseur

Cette section compare les performances d'une procédure stockée écrite des deux manières suivantes :

- Sans curseur : cette procédure balaye la table trois fois en modifiant le prix de chaque livre.
- Avec un curseur : elle n'effectue qu'un seul passage dans la table.

Dans les deux exemples, il existe un index unique sur titles(title_id).

Exemple de procédure stockée sans curseur

Voici un exemple de procédure stockée sans curseur :

```
/* Increase the prices of books in the
** titles table as follows:
**
** If current price is <= $30, increase it by 20%
** If current price is > $30 and <= $60, increase
** it by 10%
** If current price is > $60, increase it by 5%
**
** All price changes must take effect, so this is
** done in a single transaction.
*/
create procedure increase_price
as

    /* start the transaction */
    begin transaction
    /* first update prices > $60 */
    update titles
        set price = price * 1.05
        where price > $60

    /* next, prices between $30 and $60 */
    update titles
        set price = price * 1.10
        where price > $30 and price <= $60
```

```

    /* and finally prices <= $30 */
    update titles
    set price = price * 1.20
    where price <= $30

    /* commit the transaction */
    commit transaction

    return

```

Exemple de procédure stockée avec curseur

Cette procédure effectue les mêmes modifications sur les tables sous-jacentes que la procédure écrite sans curseur mais elle utilise des curseurs au lieu de la programmation à orientation ensembliste. Au fur et à mesure de l'extraction, de l'analyse et de la mise à jour de chaque ligne, un verrou est posé sur la page de données appropriée. De plus, comme l'indiquent les commentaires, chaque mise à jour est validée en l'état, puisqu'il n'existe pas de transaction explicite.

```

/* Same as previous example, this time using a
** cursor. Each update commits as it is made.
*/
create procedure increase_price_cursor
as
declare @price money

/* declare a cursor for the select from titles */
declare curs cursor for
    select price
    from titles
    for update of price

/* open the cursor */
open curs

/* fetch the first row */
fetch curs into @price

/* now loop, processing all the rows
** @@sqlstatus = 0 means successful fetch
** @@sqlstatus = 1 means error on previous fetch
** @@sqlstatus = 2 means end of result set reached
*/

```

```
while (@@sqlstatus != 2)
begin
    /* check for errors */
    if (@@sqlstatus = 1)
        begin
            print "Error in increase_price"
            return
        end

    /* next adjust the price according to the
     ** criteria
    */
    if @price > $60
        select @price = @price * 1.05
    else
        if @price > $30 and @price <= $60
            select @price = @price * 1.10
        else
            if @price <= $30
                select @price = @price * 1.20

    /* now, update the row */
    update titles
    set price = @price
    where current of curs

    /* fetch the next row */
    fetch curs into @price
end

/* close the cursor and return */
close curs
return
```

A votre avis, quelle procédure sera la plus performante, celle qui exécute trois balayages de tables ou celle qui en effectue un seul par l'intermédiaire d'un curseur ?

Comparaison des performances avec curseur et sans curseur

Le tableau 13-2 montre les statistiques d'une table de 5 000 lignes. Le code curseur est quatre fois plus long, bien qu'il n'effectue qu'un seul balayage de la table.

Tableau 13-2 : Temps d'exécution type pour une table de 5 000 lignes

Procédure	Méthode d'accès	Durée
increase_price	Avec 3 balayages de tables	28 secondes
increase_price_cursor	Avec un curseur et un seul balayage de table	125 secondes

Les résultats de ce genre de test peuvent varier fortement. Ils sont assez contrastés sur les systèmes ayant des flux réseau importants, un grand nombre d'utilisateurs de base de données actifs et plusieurs utilisateurs accédant à la même table.

En plus du verrouillage, les curseurs engendrent une activité réseau plus importante que les opérations liées à l'option set et provoquent un overhead en termes d'instructions de traitement. Le programme d'application doit communiquer toutes les lignes de résultat de la requête à Adaptive Server. C'est la raison pour laquelle l'exécution du code du curseur est plus longue que celle du code qui effectue trois balayages de la table.

Les problèmes de performance liés aux curseurs concernent :

- le verrouillage au niveau page et table,
- l'utilisation des ressources réseau,
- l'overhead des instructions de traitement.

Il est préférable d'utiliser une programmation à orientation ensembliste même si elle comporte plusieurs balayages de tables.

Verrouillage à l'aide de curseurs en lecture seule

Vous pouvez utiliser le code de curseur ci-dessous pour afficher les verrous qui sont installés à différents stades de la vie d'un curseur. L'exemple suivant s'applique à une table APL. Exécutez le code dans la figure 13-4, effectuez des pauses au niveau des flèches pour exécuter sp_lock et examinez les types de verrous en place.

Figure 13-4 : Curseurs en lecture seule et essai de verrouillage

```

declare curs1 cursor for
select au_id, au_lname, au_fname
  from authors
 where au_id like '15%'
   for read only
go                                ←
open curs1                         ←
go                                ←
fetch curs1                        ←
go                                ←
fetch curs1                        ←
go 100                            ←
close curs1                        ←
go                                ←
deallocate cursor curs1           ←
go

```

Le tableau 13-3 présente les résultats.

Tableau 13-3 : Effets des commandes de curseur sur le verrouillage de pages de données et d'index

Événement	Page de données
Après declare	Pas de verrou associé aux curseurs.
Après open	Verrou d'intention partagé sur authors.
Après la première occurrence de fetch	Verrou d'intention partagé sur authors et verrou de page partagé sur une page d'authors.
Après 100 occurrences de fetch	Verrou d'intention partagé sur authors et verrou de page partagé sur une autre page d'authors.
Après close	Pas de verrou associé aux curseurs.

Si vous émettez une autre commande fetch après l'extraction de la dernière ligne du jeu de résultats, les verrous se trouvant sur la dernière page sont libérés. Il n'y aura donc pas de verrous associés au curseur.

Avec une table DOL :

- Si la requête s'exécute au niveau d'isolement 1 et que `read committed with lock` est défini à 0, vous ne voyez aucun verrou de page ou de ligne. Les valeurs sont copiées depuis la page ou la ligne et le verrou est immédiatement libéré.
- Si `read committed with lock` a la valeur 1 ou si la requête s'exécute au niveau d'isolement 2 ou 3, vous constatez la présence de verrous de pages ou de lignes partagés au niveau indiqué par le [tableau 13-3](#). Si la table utilise un verrouillage des lignes de données, les résultats de `sp_lock` mentionnent l'ID de la ligne extraite.

Niveaux d'isolement et curseurs

Le plan d'exécution de requête d'un curseur est compilé et optimisé au moment de son ouverture. Vous ne pouvez pas ouvrir un curseur, puis utiliser `set transaction isolation level` pour modifier le niveau d'isolement auquel le curseur opère.

Etant donné que la compilation des curseurs diffère selon qu'ils utilisent un niveau d'isolement 0 ou d'autres niveaux, vous ne pouvez pas ouvrir un curseur au niveau d'isolement 0, puis l'ouvrir ou y effectuer des extractions au niveau 1 ou 3. De même, vous ne pouvez pas ouvrir un curseur au niveau 1 ou 3 et y effectuer des extractions au niveau 0. Si vous essayez d'ouvrir un curseur ou d'en extraire des données à partir d'un niveau incompatible, vous obtiendrez un message d'erreur.

Une fois que le curseur a été ouvert à un niveau d'isolement spécifique, vous devez le libérer avant de changer de niveau d'isolement. La modification des niveaux d'isolement tant que le curseur est ouvert entraîne les effets suivants :

- La tentative de fermeture et de réouverture du curseur à un autre niveau d'isolement échoue et un message d'erreur est généré.
- La tentative de modification des niveaux d'isolement sans fermeture et réouverture du curseur n'a pas d'influence sur le niveau en cours et ne génère pas de message d'erreur.

Vous pouvez inclure une clause `at isolation` dans le curseur pour spécifier un niveau d'isolement. Le curseur repris dans l'exemple ci-dessous peut être déclaré au niveau 1 et extrait au niveau 0 car le plan d'exécution de requête et le niveau d'isolement sont compatibles :

```
declare cprice cursor for
    select title_id, price
        from titles
            where type = "business"
                at isolation read uncommitted
```

Tables sans index partitionnées et curseurs

En effectuant un balayage de curseur sur une table non partitionnée sans index, vous pouvez lire toutes les données jusqu'à la dernière insertion effectuée dans cette table, même si des insertions ont eu lieu après le lancement du balayage de curseur.

Si une table sans index est partitionnée, les données peuvent être insérées dans un des nombreux chaînages de pages. Le point d'insertion physique peut se trouver avant ou après la position courante d'un balayage de curseur. En d'autres termes, un balayage de curseur sur une table partitionnée ne garantit *pas* le balayage des dernières insertions effectuées dans la table.

Remarque S'il est nécessaire, pour vos opérations de curseur, que toutes les insertions soient effectuées à la fin d'un seul chaînage de type page, ne partitionnez *pas* la table utilisée dans le balayage de curseur.

Conseils d'optimisation des curseurs

Voici quelques conseils pour l'optimisation des curseurs :

- Optimisez les sélections de curseur à l'aide du curseur et non d'une requête ad hoc.
- Utilisez `union` ou `union all` au lieu des clauses `or` ou des listes `in`.
- Déclarez l'intention du curseur.

- Spécifiez les noms de colonne dans la clause for update.
- Extrayez plusieurs lignes si vous renvoyez des lignes au client.
- Laissez les curseurs ouverts lors des validations et des annulations.
- Ouvrez plusieurs curseurs sur une seule connexion.

Optimisation des sélections de curseur à l'aide d'un curseur

L'optimisation d'une instruction select autonome peut être très différente selon que l'instruction se trouve dans un curseur explicitement ou implicitement modifiable. Lorsque vous développez des applications utilisant des curseurs, vérifiez toujours vos plans d'exécution de requête et les statistiques d'E/S en vous servant du curseur plutôt que d'une instruction select autonome. En particulier, les restrictions d'index de curseurs modifiables requièrent des méthodes d'accès très différentes.

Utilisation d'*union* au lieu des clauses *or* ou des listes *in*

Les curseurs ne peuvent pas utiliser les index dynamiques des ID de ligne générés par la stratégie OR. Les requêtes utilisant la stratégie OR dans des instructions select autonomes effectuent généralement des balayages de tables à l'aide de curseurs en lecture seule. Les curseurs modifiables peuvent requérir l'emploi d'un index unique et devoir malgré tout accéder à chaque ligne de données, séquentiellement, afin d'évaluer les clauses de la requête.

Pour de plus amples informations, reportez-vous à la section « [Méthodes d'accès et évaluation du coût des clauses or et in](#) », page 94.

Un curseur en lecture seule utilisant union crée une table de travail lorsqu'il est déclaré et effectue une opération de tri pour supprimer les valeurs en double. Les extractions sont effectuées sur la table de travail. Un curseur utilisant union all peut retourner des valeurs en double et ne requiert pas de table de travail.

Déclaration de l'intention du curseur

Déclarez toujours l'état d'un curseur : en lecture seule ou modifiable. Cela vous permet de mieux contrôler les conséquences au niveau de la concurrence d'accès. Si vous ne le spécifiez pas, Adaptive Server choisit un état, et la plupart du temps, il choisit les curseurs modifiables. Ces derniers utilisent des verrous de mise à jour, empêchant la pose d'autres verrous de mise à jour ou de verrous exclusifs. Si la mise à jour modifie une colonne d'index, l'optimiseur peut devoir choisir un balayage de table pour la requête, ce qui risque de provoquer des problèmes de concurrence d'accès complexes. Etudiez avec soin les plans d'exécution des requêtes qui utilisent des curseurs modifiables.

Spécification des noms de colonne dans la clause *for update*

Adaptive Server pose des verrous de mise à jour sur toutes les tables disposant de colonnes listées dans la clause *for update* de l'instruction *select* du curseur. Si la clause *for update* n'est pas incluse dans la déclaration du curseur, toutes les tables référencées dans la clause *from* posent des verrous de mise à jour.

La requête ci-après inclut le nom de la colonne dans la clause *for update* mais pose des verrous de mise à jour sur la table *titles* uniquement car *price* est mentionné dans la clause *for update*. La table utilise le verrouillage des pages complètes. Les verrous sur *authors* et *titleauthor* sont des verrous de page partagés :

```
declare curs3 cursor
for
  select au_lname, au_fname, price
    from titles t, authors a,
         titleauthor ta
   where advance <= $1000
     and t.title_id = ta.title_id
     and a.au_id = ta.au_id
  for update of price
```

Le [tableau 13-4](#) répertorie les conséquences des erreurs suivantes :

- omission complète de la clause *for update* ; pas de clause *shared* ;
- omission du nom de colonne dans la clause *for update* ;

- insertion du nom de la colonne à mettre à jour dans la clause for update ;
- ajout de shared après le nom de la table titles tout en utilisant for update of price.

Dans le tableau ci-dessous, les verrous supplémentaires ou les verrous plus restrictifs pour les deux versions de la clause for update sont mis en évidence.

Tableau 13-4 : Conséquences de la clause for update et shared sur le verrouillage du curseur

Clause	titles	authors	titleauthor
Aucune	sh_page sur l'index sh_page sur les données	sh_page sur l'index sh_page sur les données	sh_page sur les données
for update	updpage sur l'index	updpage sur l'index	updpage sur les données
	updpage sur les données	updpage sur les données	
for update of price	updpage sur l'index updpage sur les données	sh_page sur l'index sh_page sur les données	sh_page sur les données
for update of price + shared	sh_page sur les données	sh_page sur l'index sh_page sur les données	sh_page sur les données

Utilisation de set cursor rows

La norme SQL spécifie une extraction d'une ligne pour les curseurs, qui consomme des ressources réseau. Vous pouvez améliorer les performances à l'aide de l'option de requête set cursor rows et la mise en buffer transparente des extractions avec Open Client :

```
ct_cursor (CT_CURSOR_ROWS)
```

Choisissez avec soin le nombre de lignes renvoyées pour les applications fréquemment exécutées à l'aide de curseurs ; adaptez-les au réseau.

Pour plus d'informations sur ce processus, reportez-vous à la section « [Modification de la taille d'un paquet réseau](#) », page 28.

Curseurs ouverts lors des validations et des annulations

ANSI ferme les curseurs à la fin de chaque transaction. Transact-SQL fournit l'option `set close on endtran` pour les applications qui doivent se conformer aux normes ANSI. Par défaut, cependant, cette option est désactivée. A moins d'être obligé de vous conformer aux normes ANSI, laissez cette option la désactivée afin de maintenir la concurrence d'accès et la capacité de traitement.

Si vous devez respecter les normes ANSI, vous devez choisir comment en gérer son impact sur Adaptive Server. Devez-vous effectuer de nombreuses mises à jour ou suppressions en une seule transaction ? Ou devez-vous effectuer des transactions courtes ?

Si vous optez pour les transactions courtes, la fermeture et l'ouverture du curseur peuvent réduire la capacité de traitement car Adaptive Server doit rematérialiser le jeu de résultats à chaque ouverture du curseur. Si vous choisissez d'effectuer davantage d'opérations dans chaque transaction par transaction, vous risquez de vous confronter à des problèmes de concurrence d'accès, puisque la requête maintient les verrous.

Ouverture de plusieurs curseurs sur une seule connexion

Certains développeurs simulent les curseurs en utilisant deux connexions ou plus depuis DB-Library™. Une connexion effectue une opération `select` et l'autre effectue des mises à jour ou des suppressions sur les mêmes tables. Cependant, cette situation risque fort de créer des interblocages au niveau des applications. Par exemple :

- La connexion A maintient un verrou partagé sur une page. Tant qu'il existe des lignes en attente d'Adaptive Server, un verrou partagé est maintenu sur la page courante.
- La connexion B requiert un verrou exclusif sur les mêmes pages, puis attend.
- L'application attend que la connexion B aboutisse avant d'appeler le programme nécessaire pour supprimer le verrou partagé. Or, cette connexion ne se produit jamais.

Du fait que la connexion A ne demande jamais le verrou posé par la connexion B, il s'agit bien d'un interblocage au niveau applicatif et non pas associé au serveur.

Présentation des plans abstraits

Ce chapitre est une présentation générale des plans abstraits.

Sujet	Page
Définition	341
Gestion des plans abstraits	343
Relation entre le texte des requêtes et les plans d'exécution de requête	343
Plans complets et plans partiels	344
Groupes de plans abstraits	346
Association des plans abstraits avec les requêtes	347

Définition

Adaptive Server peut générer un plan abstrait pour une requête et sauvegarder le texte avec son plan abstrait dans la table système sysqueryplans. Grâce à une méthode rapide de hachage, les requêtes SQL entrantes sont comparées au texte des requêtes enregistrées et, en cas de correspondance, le plan abstrait correspondant enregistré sert à leur exécution.

Un plan abstrait décrit le plan d'exécution d'une requête en utilisant un langage créé à cette fin. Ce langage contient des opérateurs qui spécifient les choix et les actions que l'optimiseur peut générer. Par exemple, pour spécifier un balayage d'index sur la table titles avec l'index title_id_ix, le plan abstrait indique :

```
( i_scan title_id_ix titles)
```

Les plans abstraits offrent aux administrateurs système et à tous les développeurs soucieux d'optimisation un moyen de préserver les performances globales d'un serveur en cas de modification des plans d'exécution de requête. Ces modifications sont essentiellement dues à :

- des mises à jour logicielles d'Adaptive Server, qui se répercutent sur les choix de l'optimiseur et les plans d'exécution de requêtes ;
- l'introduction de nouvelles fonctionnalités d'Adaptive Server qui changent les plans d'exécution de requête ;
- la modification d'options d'optimisation telles que le degré de parallélisation, le partitionnement des tables ou l'indexation.

Le principal objectif des plans abstraits est de permettre la capture des plans de requête avant et après d'importants changements au niveau système. Il est possible de comparer les plans avant et après les modifications afin d'en déterminer les conséquences sur vos requêtes. Les plans abstraits peuvent également servir à :

- rechercher des types spécifiques de plans, par exemple des balayages de tables ou des reformatages,
- rechercher des plans qui utilisent des index particuliers,
- spécifier des plans, complets ou partiels, pour des requêtes peu performantes,
- enregistrer des plans pour des requêtes longues à optimiser.

Vous pouvez recourir aux plans abstraits au lieu d'employer des options qui doivent figurer dans le batch ou dans la requête pour orienter les décisions de l'optimiseur. Avec des plans abstraits, vous pouvez jouer sur l'optimisation d'une instruction SQL sans avoir à modifier sa syntaxe. Bien que la mise en correspondance du texte d'une requête avec le texte stocké exige un certain overhead de traitement, le recours à un plan sauvegardé réduit l'overhead d'optimisation.

Gestion des plans abstraits

Toute une série de procédures système permettent à l'administrateur système et au propriétaire de base de données de contrôler des plans et des groupes de plans. Les utilisateurs peuvent afficher, supprimer et copier les plans des requêtes qu'ils ont exécutées.

Pour de plus amples informations, reportez-vous au [Chapitre 17, « Gestion des plans abstraits avec des procédures système »](#).

Relation entre le texte des requêtes et les plans d'exécution de requête

Pour la plupart des requêtes SQL, les plans d'exécution possibles sont multiples. SQL décrit le jeu de résultats souhaité mais non la manière dont il doit être extrait de la base de données. Prenons une requête qui joint trois tables, comme suit :

```
select t1.c11, t2.c21
  from t1, t2, t3
 where t1.c11 = t2.c21
   and t1.c11 = t3.c31
```

Il existe de nombreux ordres de jointure possibles et, selon les index existant sur les tables, de nombreuses méthodes d'accès, telles que les balayages de tables ou d'index ou la stratégie de reformatage. Chaque jointure peut utiliser une fusion ou une boucle imbriquée. Ces choix sont déterminés par l'optimiseur sur la base d'algorithmes d'évaluation du coût des requêtes ; ils ne figurent pas dans la requête elle-même.

Lorsque vous capturez le plan abstrait, la requête est optimisée selon le processus normal, mais en outre, l'optimiseur génère un plan abstrait qu'il sauvegarde, avec le texte de la requête, dans la table sysqueryplans.

Limites des options pour influer sur les plans d'exécution de requête

Adaptive Server offre d'autres options pour influencer les choix de l'optimiseur :

- des options au niveau de la session, telles que `set forceplan` pour imposer un ordre de jointure ou `set parallel_degree` pour préciser le nombre maximum de processus de travail à utiliser sur la requête ;
- des options que vous pouvez inclure dans le texte de la requête afin d'influencer sur le choix d'un index, la stratégie de mise en cache ou le degré de parallélisation.

Toutefois, l'utilisation des commandes `set` ou l'ajout d'indications dans le texte des requêtes sont soumis à certaines limites :

- il n'est pas possible d'influencer sur toutes les étapes d'un plan d'exécution de requête, par exemple l'association de sous-requêtes ;
- certains outils de génération de requêtes ne supportent pas les options intégrées au texte ou encore ils ne sont opérationnels que si toutes les requêtes sont indépendantes d'un fournisseur.

Plans complets et plans partiels

Les plans abstraits peuvent être soit des plans complets, qui décrivent toutes les étapes et options de traitement d'une requête, soit des plans partiels. Un plan partiel peut demander qu'un index soit utilisé pour balayer une table spécifique, sans indiquer le nom de cet index ou l'ordre de jointure de la requête. Par exemple :

```
select t1.c11, t2.c21
  from t1, t2, t3
 where t1.c11 = t2.c21
   and t1.c11 = t3.c31
```

Le plan abstrait complet mentionne :

- Le type de jointure, soit `nl_g_join` pour les jointures à boucle imbriquée, soit `m_g_join` pour les jointures par fusion. Le plan de la requête ci-dessus spécifie une jointure à boucle imbriquée.
- L'ordre de jointure, inclus dans la clause `nl_g_join`.

- Le type de balayage : `t_scan` pour un balayage de table ou `i_scan` pour un balayage d'index.
- Le nom de l'index choisi pour les tables dont l'accès s'effectue par un balayage d'index.
- Les propriétés de balayage : degré de parallélisation, taille des E/S et stratégie de mise en cache pour chaque table de la requête.

Le plan abstrait de la requête ci-dessus spécifie l'ordre de jointure, la méthode d'accès à chaque table de la requête et les propriétés de balayage des tables :

```
( nl_g_join
  ( t_scan t2 )
  ( i_scan t1_c11_ix t1 )
  ( i_scan t3_c31_ix t3 )
)
( prop t3
  ( parallel 1 )
  ( prefetch 16 )
  ( lru )
)
( prop t1
  ( parallel 1 )
  ( prefetch 16 )
  ( lru )
)
( prop t2
  ( parallel 1 )
  ( prefetch 16 )
  ( lru )
)
```

Le [Chapitre 18, « Référence de langage des plans abstraits »](#), décrit le langage et la syntaxe des plans abstraits.

Création d'un plan partiel

Lorsque les plans abstraits sont capturés, les plans complets sont générés et enregistrés. Vous pouvez rédiger des plans partiels qui n'incluent qu'une partie des choix de l'optimiseur. Si la requête précédente n'a pas utilisé l'index sur t3 mais que toutes les autres parties du plan d'exécution sont performantes, vous pouvez créer un plan partiel en utilisant la commande `create plan`. Le plan ci-dessous ne spécifie que le choix d'index pour t3 :

```
create plan
"select t1.c11, t2.c21
from t1, t2, t3
where t1.c11 = t2.c21
and t1.c11 = t3.c31"
"( i_scan t3_c31_ix t3 )"
```

Vous pouvez aussi créer des plans abstraits avec la clause `plan` pour des commandes `select`, `delete`, `update` ou d'autres commandes également optimisables.

Reportez-vous à la section « [Création de plans avec SQL](#) », page 389.

Groupes de plans abstraits

Lorsque vous installez Adaptive Server pour la première fois, il existe deux groupes de plans abstraits :

- `ap_stdout`, utilisé par défaut pour la capture des plans,
- `ap_stdin`, utilisé par défaut pour l'association des plans.

L'administrateur système peut activer, au niveau du serveur, la capture de plans dans `ap_stdout`, ce qui permet la capture de tous les plans de toutes les requêtes. La fonction d'association de plans au niveau du serveur utilise des requêtes et des plans du groupe `ap_stdin`. Si des requêtes exigent des plans d'optimisation spécifiques, il est possible de les rendre accessibles au niveau du serveur.

L'administrateur système ou le propriétaire de la base de données peut créer d'autres groupes de plans, copier des plans d'un groupe vers un autre et comparer les plans de deux groupes différents.

La capture des plans abstraits et l’association de plans abstraits à des requêtes se déroulent toujours au sein du groupe de plans actif. Au niveau de la session, les utilisateurs disposent des commandes `SET` pour activer la capture et l’association des plans.

Les diverses utilisations de groupes de plans abstraits sont les suivantes :

- Un optimiseur de requêtes peut créer des plans abstraits dans un groupe à des fins de test sans perturber les plans des autres utilisateurs du système.
- En définissant des groupes, il est possible de définir des ensembles de plans « avant » et « après » afin de déterminer l’impact d’une modification ou d’une mise à niveau du système sur l’optimisation des requêtes.

Pour plus d’informations sur l’activation des fonctions de capture et d’association de plans, reportez-vous au [Chapitre 16, « Création et utilisation des plans abstraits »](#).

Association des plans abstraits avec les requêtes

Lorsqu’un plan abstrait est sauvegardé, tous les espaces blancs (retours à la ligne, tabulations, espaces multiples) dans une requête sont tronqués à un seul espace et une valeur de clé de hachage est calculée pour l’instruction SQL tronquée correspondant à l’espace blanc. L’instruction SQL tronquée et la clé de hachage sont stockées dans la table `sysqueryplans` en même temps que le plan abstrait, un ID de plan unique, l’ID d’utilisateur et l’ID du groupe de plans abstraits en cours.

Lorsque l’association de plans abstraits est activée, le système calcule la clé de hachage pour les instructions SQL entrantes, puis s’en sert pour rechercher la requête et le plan abstrait correspondants dans le groupe d’association actif, avec l’ID d’utilisateur correspondant. La **clé d’association** complète pour un plan abstrait se compose des éléments suivants :

- l’ID de l’utilisateur courant,
- l’ID du groupe d’association courant,
- le texte complet de la requête.

Dès qu'Adaptive Server trouve une clé de hachage pertinente, il compare le texte complet de la requête sauvegardée avec celui de la requête à exécuter et, le cas échéant, l'utilise.

La combinaison des clés d'association d'ID utilisateur, d'ID de groupe et de texte de requête signifie que pour un utilisateur donné, qu'un même groupe de plans abstraits ne peut comporter deux requêtes ayant le même texte, mais des plans de requête différents.

Guide des plans abstraits pour les requêtes

Ce chapitre présente quelques instructions utiles pour la création de plans abstraits.

Sujet	Page
Introduction	349
Conseils pour écrire des plans abstraits	373
Comparaison des plans « avant » et « après »	374
Plans abstraits pour des procédures stockées	377
Requêtes ad hoc et plans abstraits	378

Introduction

Les plans abstraits permettent de spécifier le plan d'exécution que vous souhaitez adopter pour une requête. Vous pouvez recourir aux plans abstraits au lieu d'employer des options de niveau session et requête qui imposent un ordre de jointure, définissent l'index ou la taille des E/S ainsi que d'autres éléments d'exécution des requêtes. Les options de niveau session et requête sont décrites au [Chapitre 16, « Création et utilisation des plans abstraits »](#).

Il existe plusieurs choix d'optimisation qu'il est impossible de spécifier avec des commandes `set` ou des clauses incluses dans le texte de la requête. Voici quelques exemples :

- association de sous-requête,
- ordre de jointure pour les sous-requêtes mises à plat,
- redéfinition d'index.

Dans la plupart des cas, l'insertion de commandes `set` ou la modification du texte de la requête n'est pas toujours possible ni même souhaitable.

Les plans abstraits offrent alors d'autres moyens plus exhaustifs d'influencer les décisions de l'optimiseur.

Les plans abstraits sont des expressions algébriques relationnelles qui ne sont pas incluses dans le texte de la requête. Ils sont stockés dans un catalogue système et associés aux requêtes entrantes sur la base du texte de ces dernières.

Les tables citées dans cette section sont les mêmes que dans le [Chapitre 18, « Référence de langage des plans abstraits »](#). Pour plus d'informations sur les instructions `create table` et `create index`, reportez-vous à la section [« Schéma des exemples », page 410](#).

Langage des plans abstraits

Le langage des plans abstraits est un ensemble d'expressions algébriques relationnelles qui utilisent les opérateurs suivants :

- `g_join`, la jointure générique, un opérateur de jointure logique de haut niveau. Il décrit les jointures internes, externes et de contrôle de présence en utilisant soit des jointures à boucles imbriquées, soit des jointures tri-fusion ;
- `nl_g_join`, qui spécifie une jointure à boucles imbriquées, comprenant toutes les jointures internes, externes et de contrôle de présence ;
- `m_g_join`, qui spécifie une jointure par fusion, comprenant les jointures internes et externes ;
- `union`, opérateur logique d'union. Il décrit à la fois les constructions SQL `union` et `union all` ;
- `scan`, opérateur logique qui transforme une table stockée en un flux de lignes, appelé table sous-jacente de plan abstrait. Il permet de définir des plans partiels qui ne restreignent pas la méthode d'accès.
- `i_scan`, opérateur physique qui met en oeuvre l'opérateur `scan`. Il oriente le choix de l'optimiseur vers un balayage d'index sur la table désignée.
- `t_scan`, opérateur physique qui met en oeuvre l'opérateur `scan`. Il oriente le choix de l'optimiseur vers un balayage complet de table sur la désignée.
- `store`, opérateur logique qui décrit la matérialisation d'une table sous-jacente de plan abstrait en table de travail stockée.
- `nested`, filtre décrivant l'emplacement et la structure des sous-requêtes imbriquées.

Pour plus d'informations sur les commandes `create table` et `create index` utilisées dans les exemples, reportez-vous à la section « [Schéma des exemples](#) », page 410.

D'autres mots-clés propres aux plans abstraits sont utilisés pour le regroupement et l'identification, notamment :

- `plan` regroupe les éléments lorsqu'un plan nécessite plusieurs étapes ;
- `hints` regroupe un ensemble d'indications pour un plan partiel ;
- `prop` introduit un ensemble de propriétés de balayage pour une table : `prefetch`, `lru|mru` et `parallel` ;
- `table` identifie une table lorsqu'un alias est utilisé ou qu'elle figure dans une sous-requête ou une vue ;
- `work_t` identifie une table de travail ;
- `in`, utilisé avec `table`, sert à identifier les tables qui se trouvent dans une sous-requête (`subq`) ou une vue (`view`) ;
- `subq` est également utilisé avec l'opérateur `nested` pour indiquer le point d'association d'une sous-requête imbriquée et pour introduire le plan abstrait de la sous-requête.

Requêtes, méthodes d'accès et plans abstraits

Sur une table donnée, il existe plusieurs méthodes d'accès possibles pour une requête : par exemple, balayages d'index avec différents index, balayages de tables, stratégie OR et redéfinition d'index.

Ainsi, pour la requête simple ci-après, les méthodes d'accès possibles sont nombreuses :

```
select * from t1
where c11 > 1000 and c12 < 0
```

Les plans abstraits suivants spécifient trois méthodes d'accès différentes :

- Utilisation de l'index `i_c11` :
`(i_scan i_c11 t1)`
- Utilisation de l'index `i_c12` :
`(i_scan i_c12 t1)`
- Balayage complet de table :
`(t_scan t1)`

Les plans abstraits sont soit des plans complets précisant tous les choix de l'optimiseur pour une requête, soit des plans partiels ne contenant qu'un sous-ensemble de choix, par exemple l'index à utiliser pour une seule des tables de la requête, sans qu'il soit fait mention de l'ordre de jointure des tables. Par exemple, avec un plan partiel, vous pouvez demander qu'un index soit utilisé, en laissant l'optimiseur choisir entre i_c11 et i_c12 et en spécifiant par ailleurs que vous ne voulez pas de balayage complet de table. Les parenthèses vides remplacent le nom d'index :

```
(i_scan () t1)
```

En outre, la requête peut utiliser des E/S de 2 ou 16 Ko et s'exécuter en série ou en parallèle.

Tables sous-jacentes

Une table sous-jacente est définie par l'évaluation de l'expression d'une requête et diffère d'une table normale en ce qu'elle n'est ni décrite dans les catalogues système ni stockée sur le disque. Dans Adaptive Server, il existe deux types de tables sous-jacentes : table sous-jacente SQL et table sous-jacente de plan abstrait.

- *Une table sous-jacente SQL* est définie par une ou plusieurs tables via l'évaluation de l'expression d'une requête. Cette table est utilisée dans l'expression de requête dans laquelle elle est définie et n'existe que pendant la durée de la requête. Pour plus d'informations sur les tables sous-jacentes SQL, reportez-vous au *Guide de l'utilisateur Transact-SQL*.
- *Une table sous-jacente de plan abstrait* est une table sous-jacente utilisée pour le traitement, l'optimisation et l'exécution des requêtes. Elle diffère de la table sous-jacente SQL dans le sens où elle existe en tant que partie d'un plan abstrait et est invisible aux yeux de l'utilisateur final.

Identification des tables

Les plans abstraits doivent nommer toutes les tables d'une requête de manière non ambiguë, afin qu'il soit possible de lier une table nommée dans un plan à son occurrence dans la requête SQL. Le plus souvent, le nom de la table suffit. Si la requête qualifie le nom de la table avec le nom de la base de données et du propriétaire, ces informations sont également requises dans le plan abstrait pour identifier complètement une table.

Ainsi, la requête ci-dessous utilise le nom de table non qualifié :

```
select * from t1
```

Par conséquent, le plan abstrait utilisera lui aussi le nom non qualifié :

```
(t_scan t1)
```

Si un nom de base de données et/ou un nom de propriétaire figurent dans la requête :

```
select * from pubs2.dbo.t1
```

Le plan abstrait contiendra également ces qualifications :

```
(t_scan pubs2.dbo.t1)
```

Cependant, la même table peut apparaître plusieurs fois dans la même requête, comme ici :

```
select * from t1 a, t1 b
```

Dans l'exemple ci-dessus, les alias **a** et **b** identifient les deux tables dans SQL. Dans un plan abstrait, l'opérateur **table** associe chaque alias à l'occurrence de la table :

```
( g_join
    ( t_scan ( table ( a t1 ) ) )
    ( t_scan ( table ( b t1 ) ) )
)
```

Les noms de table peuvent également être ambigus dans les vues et les sous-requêtes, de sorte que dans ces objets, l'opérateur **table** est employé pour désigner les tables.

En ce qui concerne les sous-requêtes, les opérateurs **in** et **subq** qualifient le nom de la table avec la sous-requête, qui est son contexte syntaxique. Dans l'exemple suivant, la même table est utilisée dans la requête externe et dans la sous-requête :

```
select *
from t1
where c11 in (select c12 from t1 where c11 > 100)
```

Le plan abstrait identifie les deux occurrences sans aucune ambiguïté :

```
( g_join
  ( t_scan t1 )
  ( i_scan i_c11_c12 ( table t1 ( in ( subq 1 ) ) ) )
)
```

En ce qui concerne les vues, les opérateurs `in` et `view` servent à l'identification. Ici, la requête référence une table utilisée dans la vue :

```
create view v1
as
select * from t1 where c12 > 100
select t1.c11 from t1, v1
  where t1.c12 = v1.c11
```

Voici le plan abstrait correspondant :

```
( g_join
  ( t_scan t1 )
  ( i_scan i_c12 ( table t1 ( in ( view v1 ) ) ) )
)
```

Identification des index

L'opérateur `i_scan` a besoin de deux opérandes : le nom d'index et le nom de table, comme indiqué ci-dessous :

```
( i_scan i_c12 t1 )
```

Pour spécifier la nécessité d'utiliser un index sans préciser lequel, remplacez le nom de l'index par des parenthèses vides :

```
( i_scan ( ) t1 )
```

Spécification de l'ordre de jointure

Adaptive Server effectue des jointures de trois tables ou plus en joignant deux d'entre elles, puis en joignant la « table sous-jacente de plan abstrait » obtenue à la table suivante dans l'ordre de jointure. Cette table sous-jacente de plan abstrait est un flux de lignes, comme en renverrait une jointure par boucles imbriquées exécutée antérieurement dans la requête.

La requête ci-dessous joint trois tables :

```
select *
from t1, t2, t3
where c11 = c21
    and c12 = c31
    and c22 = 0
    and c32 = 100
```

Cet exemple illustre la nature binaire de l'algorithme de jointure, avec les opérateurs `g_join`. Le plan spécifie l'ordre de jointure `t2, t1, t3` :

```
(g_join
  (g_join
    (scan t2)
    (scan t1)
  )
  (scan t3)
)
```

Les résultats de la jointure `t2-t1` sont ensuite joints à `t3`. L'opérateur `scan` laisse à l'optimiseur le choix entre un balayage de table ou un balayage d'index.

Notation raccourcie des jointures

En général, une jointure multiple à N éléments, avec l'ordre $t1, t2, t3\dots, tN-1, tN$ se présente comme suit :

```
(g_join
  (g_join
    ...
    (g_join
      (g_join
        (scan t1)
        (scan t2)
      )
      (scan t3)
    )
    ...
    (scan tN-1)
  )
  (scan tN)
)
```

Cette notation est une forme abrégée de l'opérateur g_join :

```
(g_join
  (scan t1)
  (scan t2)
  (scan t3)
  ...
  (scan tN-1)
  (scan tN)
)
```

Elle est également utilisable pour g_join, nl_g_join et m_g_join.

Exemples d'ordre de jointure

Pour cette requête à trois éléments, l'optimiseur peut choisir parmi plusieurs plans :

```
select *
  from t1, t2, t3
 where c11 = c21
       and c12 = c31
       and c22 = 0
       and c32 = 100
```

Voici quelques exemples :

- Utilisation de c22 comme argument de recherche sur t2, jointure avec t1 sur c11, puis avec t3 sur c31 :

```
(g_join
  (i_scan i_c22 t2)
  (i_scan i_c11 t1)
  (i_scan i_c31 t3)
)
```

- Utilisation de l'argument de recherche sur t3 et de l'ordre de jointure t3, t1, t2 :

```
(g_join
  (i_scan i_c32 t3)
  (i_scan i_c12 t1)
  (i_scan i_c21 t2)
)
```

- Balayage complet de table de t2 si celle-ci est de petite taille et tient dans le cache, toujours avec l'ordre de jointure t3, t1, t2 :

```
(g_join
  (i_scan i_c32 t3)
  (i_scan i_c12 t1)
  (t_scan t2)
)
```

- Si la table t1 est volumineuse et que t2 et t3 qualifient chacune une grande partie de t1 correspondant toutefois à une très petite partie par rapport à l'ensemble, le plan ci-dessous spécifie une jointure en étoile :

```
(g_join
  (i_scan i_c22 t2)
  (i_scan i_c32 t3)
  (i_scan i_c11_c12 t1)
)
```

Tous ces plans imposent le choix d'un ordre de jointure, en laissant à l'optimiseur le soin de choisir le type de la jointure.

L'opérateur générique g_join met en œuvre des jointures externes, internes et de contrôle de présence. Pour obtenir des exemples de sous-requêtes mises à plat qui réalisent des jointures de contrôle de présence, reportez-vous à la section « [Sous-requêtes mises à plat](#) », page 363.

Correspondance entre les méthodes d'exécution et les plans abstraits

Il existe certaines limites aux ordres et aux types de jointures, selon le type de la requête. C'est par exemple le cas des jointures externes, comme celle-ci :

```
select * from t1, t2
where c11 *= c21
```

Durant le traitement de la jointure par Adaptive Server, il faut que le membre extérieur de la jointure externe soit la table externe dans l'ordre de jointure. Par conséquent, le plan abstrait suivant est incorrect :

```
(g_join
  (scan t2)
  (scan t1)
)
```

Si vous tentez d'utiliser ce plan, un message d'erreur est généré et la requête n'est pas compilée.

Spécification de l'ordre de jointure pour les requêtes utilisant des vues

Vous pouvez utiliser les plans abstraits pour imposer l'ordre de jointure pour les vues fusionnées. L'exemple ci-après crée une vue. La vue ci-après effectue une jointure entre t2 et t3 :

```
create view v2
as
select *
from t2, t3
where c22 = c32
```

Cette requête effectue une jointure avec la table t2 dans la vue :

```
select * from t1, v2
where c11 = c21
and c22 = 0
```

Ce plan abstrait spécifie l'ordre de jointure t2, t1, t3 :

```
(g_join
  (scan (table t2 (in (view v2))))
  (scan t1)
  (scan (table t3 (in (view v2))))
)
```

L'exemple suivant joint t3 dans la vue :

```
select * from t1, v2
where c11 = c31
and c32 = 100
```

Ce plan spécifie l'ordre de jointure t3, t1, t2 :

```
(g_join
  (scan (table t3 (in (view v2))))
  (scan t1)
  (scan (table t2 (in (view v2))))
)
```

Il s'agit d'un exemple où il est possible d'utiliser des plans abstraits, si nécessaire, pour imposer l'ordre de jointure d'une requête, lorsque la commande `set forceplan` ne le permet pas.

Spécification du type de jointure

Adaptive Server peut effectuer des jointures à boucles imbriquées ou des jointures par fusion. L'opérateur `g_join` laisse l'optimiseur libre de choisir l'algorithme de jointure le mieux adapté en fonction du coût. Pour spécifier une jointure à boucles imbriquées, utilisez l'opérateur `nl_g_join` ; pour une jointure par tri-fusion, utilisez `m_g_join`. Les plans abstraits capturés par Adaptive Server incluent toujours l'opérateur désigné par l'algorithme et non l'opérateur `g_join`.

La lettre « `g` » figurant dans les opérateurs signifiant « générique », ceux-ci s'appliquent aussi bien à des jointures internes qu'externes ; `g_join` et `nl_g_join` sont également applicables à des jointures de contrôle de présence.

La requête ci-après spécifie une jointure entre `t1` et `t2` :

```
select * from t1, t2
      where c12 = c21 and c11 = 0
```

Le plan abstrait suivant spécifie une jointure à boucles imbriquées :

```
(nl_g_join
  (i_scan i_c11 t1)
  (i_scan i_c21 t2)
)
```

Le plan à boucles imbriquées utilise l'index `i_c11` pour restreindre le balayage en se servant de la clause de recherche, puis il effectue la jointure avec `t2`, en utilisant l'index sur la colonne de jointure.

Ce plan de jointure par fusion utilise des index différents :

```
(m_g_join
  (i_scan i_c12 t1)
  (i_scan i_c21 t2)
)
```

Pour les clés de fusion, la jointure par fusion utilise les index sur les colonnes de jointure `i_c12` et `i_c21`. Cette requête effectue une jointure par fusion complète et aucun tri n'est nécessaire.

Une jointure par fusion peut aussi utiliser l'index sur `i_c11` afin de sélectionner les lignes de `t1` et de les insérer dans une table de travail ; la fusion utilise l'index sur `i_c21` :

```
(m_g_join
  (i_scan i11 t1)
  (i_scan i21 t2)
)
```

La phase de création de la table de travail n'est pas précisée dans le plan ; l'optimiseur détecte lui-même s'il convient d'employer une table de travail et une opération de tri pour l'agencement des clés de jointure.

Spécification des plans partiels et des indications

Il existe des cas où le plan complet n'est pas nécessaire. Par exemple, si pour un plan d'exécution de requête, l'optimiseur choisit un balayage de table au lieu d'un index non clusterisé, vous pouvez, par le biais du plan abstrait, imposer le recours à l'index tout en laissant à l'optimiseur le soin de prendre les autres décisions.

Imaginons que celui-ci choisisse le balayage de table de t3 au lieu d'utiliser i_c31 dans la requête suivante :

```
select *
  from t1, t2, t3
 where c11 = c21
       and c12 < c31
       and c22 = 0
       and c32 = 100
```

Le plan suivant, tel que l'optimiseur le génère, spécifie l'ordre de jointure t2, t1, t3. Cependant, il spécifie également un balayage de la table t3 :

```
(g_join
  (i_scan i_c22 t2)
  (i_scan i_c11 t1)
  (t_scan t3)
)
```

Dans ce cas, vous pouvez modifier le plan complet afin d'imposer le recours à l'index i_c31 :

```
(g_join
  (i_scan i_c22 t2)
  (i_scan i_c11 t1)
  (i_scan i_c31 t3)
)
```

Toutefois, l'utilisation d'un plan abstrait partiel laisse à l'optimiseur une grande marge de manœuvre. A mesure que les données contenues dans les autres tables de la requête sont mises à jour, l'ordre de jointure optimal change aussi. Le plan partiel peut ainsi ne concerner qu'un élément.

Pour le balayage d'index de t3, ce plan partiel est très simple :

```
(i_scan i_c31 t3)
```

L'optimiseur choisit l'ordre de jointure et les méthodes d'accès à t1 et t2.

Regroupement d'indications multiples

Souvent, plusieurs fragments de plan s'avèrent nécessaires. Par exemple, vous souhaitez qu'un index soit utilisé pour chaque table de la requête, mais vous laissez l'optimiseur choisir l'ordre de jointure. Si plusieurs indications sont nécessaires, il convient de les regrouper avec l'opérateur hints :

```
(hints
  (i_scan () t1)
  (i_scan () t2)
  (i_scan () t3)
)
```

Dans ce cas, le rôle de l'opérateur hints est purement syntaxique, il n'a aucune incidence sur l'ordre des balayages.

Les indications fournies ne font l'objet d'aucune limitation particulière. Vous pouvez mélanger des ordres de jointure partiels et des méthodes d'accès partielles. Cet opérateur indique que t2 est externe à t1 dans l'ordre de jointure et conseille le recours à un index pour le balayage de t3 ; l'optimiseur peut néanmoins choisir l'index pour t3, les méthodes d'accès à t1 et t2 ainsi que la position de t3 dans l'ordre de jointure :

```
(hints
  (g_join
    (scan t2)
    (scan t1)
  )
  (i_scan () t3)
)
```

Plans incorrects et incohérents utilisant des indications

Il est possible qu'un plan contenant des indications soit incohérent, comme celui-ci qui stipule des ordres de jointure contradictoires :

```
(hints
  (g_join
    (scan t2)
    (scan t1)
  )
  (g_join
    (scan t1)
    (scan t2)
  )
)
```

Lorsque la requête associée à ce plan est exécutée, sa compilation est impossible et une erreur est générée.

D'autres indications incohérentes ne génèrent pas nécessairement une exception mais elles peuvent utiliser toutes les méthodes d'accès spécifiées, que le choix soit pertinent ou non. Ainsi, le plan ci-dessous spécifie à la fois un balayage d'index et un balayage de table pour la même table :

```
(hints
    (t_scan t3)
    (i_scan () t3)
)
```

Dans ce cas, l'une ou l'autre des méthodes est appliquée et le résultat est incertain.

Création de plans abstraits pour des sous-requêtes

Dans Adaptive Server, les sous-requêtes sont résolues de plusieurs façons et les plans abstraits reflètent les étapes de l'exécution de la requête :

- Matérialisation : la sous-requête est exécutée et les résultats sont stockés dans une table de travail ou une variable interne. Reportez-vous à la section « [Sous-requêtes matérialisées](#) », page 363.
- Mise à plat : la requête est mise à plat et transformée en jointure avec les tables de la requête principale. Reportez-vous à la section « [Sous-requêtes mises à plat](#) », page 363.
- Imbrication : la sous-requête est exécutée une fois pour chaque ligne de requête externe. Reportez-vous à la section « [Sous-requêtes imbriquées](#) », page 365.

Les plans abstraits ne permettent pas de choisir la méthode de résolution d'une sous-requête élémentaire. Cette décision s'appuie sur des règles et elle n'est pas modifiable pendant l'optimisation de la requête. Toutefois, ils peuvent influencer les plans relatifs aux requêtes internes et externes. Dans des sous-requêtes imbriquées, les plans abstraits sont également utilisables pour choisir le point d'imbrication dans la requête externe.

Sous-requêtes matérialisées

La requête ci-après inclut une sous-requête non liée qui peut être matérialisée :

```
select *
from t1
where c11 = (select count(*) from t2)
```

La première étape du plan abstrait consiste à matérialiser l'agrégat scalaire de la sous-requête. La seconde étape utilise le résultat pour balayer t1 :

```
( plan
  ( i_scan i_c21 ( table t2 ( in (subq 1 ) ) ) )
  ( i_scan i_c11 t1 )
)
```

La requête ci-après inclut un agrégat vectoriel dans la sous-requête :

```
select *
from t1
where c11 in (select max(c21)
               from t2
               group by c22)
```

Dans la première étape, le plan abstrait matérialise la sous-requête puis, dans une seconde étape, il la joint à la requête externe :

```
( plan
  ( store Worktab1
    ( t_scan ( table t2 ( in (subq 1 ) ) ) )
  )
  ( nl_g_join
    ( t_scan t1 )
    ( t_scan ( work_t Worktab1 ) )
  )
)
```

Sous-requêtes mises à plat

Certaines sous-requêtes peuvent être transformées en jointure. Les opérateurs g_join et nl_g_join laissent à l'optimiseur le soin de détecter si une jointure de contrôle de présence est nécessaire.

Par exemple, cette requête inclut une sous-requête introduite par exists :

```
select * from t1
where c12 > 0
      and exists (select * from t2
                   where t1.c11 = c21
                     and c22 < 100)
```

La sémantique de la requête nécessite une jointure de contrôle de présence entre t1 et t2. L'ordre de jointure t1, t2 est interprété par l'optimiseur comme une jointure de contrôle de présence, le balayage de t2 s'arrêtant sur la première ligne correspondante de t2 pour chaque ligne qualifiée dans t1 :

```
(g_join
  (scan t1)
  (scan (table t2 (in (subq 1) ) )))
)
```

L'ordre de jointure t2, t1 nécessite l'application d'autres techniques pour garantir l'élimination des valeurs dupliquées :

```
(g_join
  (scan (table t2 (in (subq 1) ) ) )
  (scan t1)
)
```

Avec ce plan abstrait, l'optimiseur peut décider d'utiliser :

- un index unique sur t2.c21, s'il en existe un, avec une jointure standard ;
- la stratégie de reformatage unique, s'il n'existe pas déjà d'index unique. Dans ce cas, la requête utilisera probablement l'index sur c22 pour sélectionner les lignes dans une table de travail ;
- la stratégie d'optimisation du tri avec élimination des valeurs dupliquées, en effectuant une jointure standard et en insérant les résultats dans une table de travail puis en triant le contenu de cette dernière.

Le plan abstrait n'a pas à spécifier les phases de création et de balayage des tables de travail intervenant dans les deux dernières options.

Pour plus d'informations sur la mise à plat des sous-requêtes, reportez-vous à la section « [Sous-requêtes mises à plat](#) », page 363.

Exemple : modification de l'ordre de jointure dans une sous-requête fusionnée

La requête peut être transformée en jointure de contrôle de présence :

```
select *
from t1, t2
where c11 = c21
  and c21 > 100
  and exists (select * from t3
  where c31 != t1.c11)
```

La corrélation « != » peut rendre le balayage de t3 assez onéreux.

Si l'ordre de jointure est t1, t2, le meilleur emplacement de t3 dans cet ordre est fonction de si la jointure de t1 et t2 augmente ou diminue le nombre de lignes et, par conséquent, le nombre de balayages de tables onéreux. Si l'optimiseur ne trouve pas le bon ordre de jointure pour t3, il peut utiliser le plan abstrait suivant, si la jointure réduit le nombre de balayages de t3 :

```
(g_join
  (scan t1)
  (scan t2)
  (scan (table t3 (in (subq 1) ) ) )
)
```

Si la jointure accroît le nombre de balayages nécessaires de t3, le plan abstrait ci-dessous effectue les balayages avant la jointure :

```
(g_join
  (scan t1)
  (scan (table t3 (in (subq 1) ) ) )
  (scan t2)
)
```

Sous-requêtes imbriquées

Les sous-requêtes imbriquées peuvent être explicitement décrites dans des plans abstraits :

- le plan abstrait de la sous-requête est indiqué,
- l'emplacement au niveau duquel la sous-requête est liée à la requête principale est spécifié.

Avec les plans abstraits, vous pouvez influer sur le plan d'exécution relatif à la sous-requête et modifier le point d'attache de cette dernière dans la requête externe.

L'opérateur `nested` spécifie la position de la sous-requête dans la requête externe. Les sous-requêtes sont « imbriquées par rapport » à une table sous-jacente de plan abstrait spécifique. L'optimiseur choisit un point au niveau duquel toutes les colonnes de corrélation de la requête externe sont disponibles et où il estime que le nombre d'exécutions de la sous-requête doit être réduit au minimum possible.

L'instruction SQL suivante contient une sous-requête d'expression reliée :

```
select *
  from t1, t2
 where c11 = c21
       and c21 > 100
       and c12 = (select c31 from t3
                  where c32 = t1.c11)
```

Le plan abstrait présente la sous-requête imbriquée par rapport au balayage de t1 :

```
( g_join
  ( nested
    ( i_scan i_c12 t1 )
    ( subq 1
      (t_scan ( table t3 ( in ( subq 1 ) ) ) )
    )
  )
  ( i_scan i_c21 t2 )
)
```

Identification et point d'attache des sous-requêtes

Les sous-requêtes sont identifiées par des numéros, dans l'ordre de leur parenthèse ouvrante « (».

Voici deux sous-requêtes, dont une dans la liste de sélection :

```
select
  (select c11 from t1 where c12 = t3.c32), c31
  from t3
  where c32 > (select c22 from t2 where c21 = t3.c31)
```

Dans le plan abstrait, la sous-requête contenant t1 est appelée « 1 » et celle contenant t2 « 2 ». Les deux sous-requêtes 1 et 2 sont imbriquées par rapport au balayage de t3 :

```
( nested
  ( nested
    ( t_scan t3 )
    ( subq 1
      ( i_scan i_c11_c12 ( table t1 (in ( subq 1 ) ) ) )
    )
  )
  ( subq 2
    ( i_scan i_c21 ( table t2 ( in ( subq 2 ) ) ) )
  )
)
```

Dans la requête ci-dessous, la seconde sous-requête est imbriquée dans la première :

```
select * from t3
where c32 > all
  (select c11 from t1 where c12 > all
    (select c22 from t2 where c21 = t3.c31))
```

Ici, la sous-requête contenant t1 est également désignée par « 1 » et celle qui contient t2 par « 2 ». Dans le plan correspondant ci-après, la sous-requête 2 est imbriquée par rapport au balayage de t1, lequel est exécuté dans la sous-requête 1 ; celle-ci est imbriquée par rapport au balayage de t3 dans la requête principale :

```
( nested
  ( t_scan t3 )
  ( subq 1
    ( nested
      ( i_scan i_c11_c12 ( table t1 ( in ( subq 1 ) ) ) )
      ( subq 2
        ( i_scan i_c21 ( table t2 ( in ( subq 2 ) ) ) )
      )
    )
  )
)
```

Autres exemples de sous-requête : ordre de lecture et points d'attache

L'opérateur nested contient comme premier opérande la table sous-jacente de plan abstrait et comme second opérande la sous-requête imbriquée. Ceci permet une lecture verticale facile de l'ordre de jointure et de l'emplacement de la sous-requête :

```
select *
from t1, t2, t3
where c12 = 0
  and c11 = c21
  and c22 = c32
  and 0 < (select c21 from t2 where c22 = t1.c11)
```

Dans le plan, l'ordre de jointure est t1, t2, t3, la sous-requête étant imbriquée par rapport au balayage de t1 :

```
( g_join
  ( nested
    ( i_scan i_c11 t1 )
    ( subq 1
      ( t_scan ( table t2 ( in (subq 1 ) ) )
    )
  )
  ( i_scan i_c21 t2 )
  ( i_scan i_c32 t3 )
)
```

Modification de l'imbrication des sous-requêtes

Si vous modifiez le point d'attache d'une sous-requête, vous devez choisir un point au niveau duquel toutes les colonnes reliées sont disponibles. La requête ci-après est en corrélation avec les deux tables de la requête externe :

```
select *
from t1, t2, t3
where c12 = 0
  and c11 = c21
  and c22 = c32
  and 0 < (select c31 from t3 where c31 = t1.c11
            and c32 = t2.c22)
```

Ce plan utilise l'ordre de jointure t1, t2, t3, avec la sous-requête qui est imbriquée par rapport à la jointure t1-t2 :

```
( g_join
  ( nested
    ( g_join
      ( i_scan i_c11_c12 t1 )
      ( i_scan i_c22 t2 )
    )
    ( subq 1
      ( t_scan ( table t3 ( in (subq 1 ) ) ) )
    )
  )
  ( i_scan i_c32 t3 )
)
```

Il serait incorrect d'imbriquer la sous-requête avec le balayage de t1 ou t2, puisqu'elle a besoin de colonnes des deux tables externes ; ce type d'erreur est corrigé pendant l'optimisation, à votre insu.

Plans abstraits pour des vues matérialisées

Cette vue est matérialisée durant le traitement de la requête :

```
create view v3
as
select distinct *
from t3
```

La requête suivante effectue une jointure sur la vue matérialisée :

```
select *
from t1, v3
where c11 = c31
```

Dans un premier temps, la vue v3 est matérialisée dans une table de travail.

Dans un second temps, une jointure avec la table de la requête principale t1 est exécutée :

```
( plan
  ( store Worktab1
    ( t_scan ( table t3 ( in (view v3) ) ) )
  )
  ( g_join
    ( t_scan t1 )
    ( t_scan ( work_t Worktab1 ) )
  )
)
```

Plans abstraits de requêtes contenant des agrégats

La requête ci-dessous renvoie un agrégat scalaire :

```
select max(c11) from t1
```

La première étape consiste à calculer l'agrégat scalaire et à le stocker dans une variable interne. La seconde étape est vide, puisqu'elle ne concerne que le renvoi de la variable, sans rien à optimiser :

```
( plan
  ( t_scan t1 )
  ( )
)
```

Pour les agrégats vectoriels, les requêtes comportent également deux étapes :

```
select max(c11)
from t1
group by c12
```

La première étape effectue le traitement des agrégats dans une table de travail, la seconde étape consistant à balayer cette table :

```
( plan
  ( store Worktab1
    ( t_scan t1 )
  )
  ( t_scan ( work_t Worktab1 ) )
)
```

Les agrégats imbriqués sont une extension de Transact-SQL :

```
select max(count(*))
from t1
group by c11
```

Une première étape traite l'agrégat vectoriel de façon à fournir une table de travail ; la seconde étape consiste à balayer cette table pour traiter l'agrégat scalaire imbriqué et le copier dans une variable interne ; dans la troisième étape, la valeur est renvoyée.

```
( plan
  ( store Worktab1
    ( i_scan i_c12 t1 )
  )
  ( t_scan ( work_t Worktab1 ) )
  ( )
)
```

Les colonnes étendues dans les requêtes avec agrégats sont une extension de Transact-SQL :

```
select max(c11), c11
from t1
group by c12
```

Dans la première étape, l'agrégat vectoriel est traité ; dans la seconde, il est joint de nouveau à la table sous-jacente afin de permettre le traitement des colonnes étendues :

```
( plan
  ( store Worktab1
    ( t_scan t1 )
  )
  ( g_join
    ( t_scan t1 )
    ( i_scan i_c11 ( work_t Worktab1 ) )
  )
)
```

L'exemple suivant contient un agrégat dans une vue fusionnée :

```
create view v4
as
select max(c11) as c41, c12 as c42
from t1
group by c12
select * from t2, v4
where c21 = 0
  and c22 > c41
```

La première étape traite l'agrégat vectoriel, la seconde le joint à la table de la requête principale :

```
( plan
  ( store Worktab1
    ( t_scan ( table t1 ( in (view v4) ) ) )
  )
  ( g_join
    ( i_scan i_c22 t2 )
    ( t_scan ( work_t Worktab1 ) )
  )
)
```

L'exemple ci-dessous inclut un agrégat qui est dans une vue matérialisée :

```
create view v5
as
select distinct max(c11) as c51, c12 as c52
from t1
group by c12
select * from t2, v5
where c21 = 0
  and c22 > c51
```

La première étape consiste à traiter l'agrégat vectoriel pour fournir une table de travail. La seconde étape crée une seconde table de travail pour traiter la vue matérialisée. La troisième étape réalise une jointure de la seconde table de travail dans la requête principale :

```
( plan
  ( store Worktab1
    ( t_scan ( table t1 ( in (view v5) ) ) )
  )
  ( store Worktab2
    ( t_scan ( work_t Worktab1 ) )
  )
  ( g_join
    ( i_scan i_c22 t2 )
    ( t_scan ( work_t Worktab2 ) )
  )
)
```

Spécification de la stratégie de reformatage

Dans la requête suivante, t2 est de très grande taille et ne contient pas d'index :

```
select *
from t1, t2
where c11 > 0
  and c12 = c21
  and c22 = 0
```

Le plan abstrait qui détermine la stratégie de reformatage sur t2 est le suivant :

```
( g_join
  (t_scan t1
  (scan
    (store Worktab1
      (t_scan t2)
    )
  )
)
```

Dans le cas d'un reformatage, l'opérateur `store` est un opérande de `scan`. C'est le seul cas où `store` n'est pas l'opérande d'un opérateur `plan`.

Limites de la stratégie OR

La stratégie OR ne dispose d'aucun plan abstrait correspondant décrivant le balayage RID requis pour l'opération finale. Tous les plans abstraits générés par Adaptive Server pour la stratégie OR ne spécifient que l'opérateur scan. Vous ne pouvez pas utiliser de plans abstraits pour influencer le choix de l'optimiseur dans des requêtes nécessitant l'application de la stratégie OR en vue d'éliminer les valeurs dupliquées.

Cas où l'opérateur `store` n'est pas spécifié

Pour certaines requêtes à plusieurs étapes nécessitant des tables de travail, il n'est pas indispensable d'établir des plans contenant eux-mêmes plusieurs étapes, dont une pour créer la table de travail en utilisant l'opérateur `store`. C'est notamment le cas pour :

- l'opération de tri des requêtes utilisant la clause `distinct`,
- les tables de travail nécessaires aux jointures par fusion,
- les tables de travail nécessaires dans les requêtes `union`,
- l'opération de tri lorsqu'une sous-requête transformée (rationalisée) doit être triée pour supprimer les valeurs dupliquées.

Conseils pour écrire des plans abstraits

Les paragraphes qui suivent contiennent quelques conseils sur l'écriture et l'utilisation des plans abstraits :

- Examinez le plan de la requête existant ainsi que les plans qui utilisent les mêmes étapes d'exécution que celui que vous devez écrire. Il est souvent plus facile de modifier un plan existant que d'en créer un entièrement nouveau.
 - Capturez le plan de la requête.
 - Utilisez la procédure `sp_help_qplan` pour afficher le plan et le texte SQL.
 - Modifiez ses résultats afin de générer une commande `create plan` ou attachez un plan modifié à la requête SQL en utilisant la clause `plan`.

- Il est souvent préférable de spécifier des plans partiels pour l'optimisation des requêtes lorsque la plupart des choix opérés par l'optimiseur s'avèrent appropriés et que seul le choix d'index par exemple doit être revu.

Avec ces plans, l'optimiseur peut choisir d'autres chemins pour les autres tables dont les données sont modifiées.

- Une fois enregistrés, les plans abstraits sont statiques. Les volumes de données et leur répartition peuvent changer et dès lors, les plans abstraits enregistrés ne sont plus aussi efficaces.

Suite à des modifications d'optimisation par ajout d'index, partitionnement des tables ou création de nouvelles zones de buffers, certains plans abstraits enregistrés risquent de n'être pas aussi performants qu'ils le devraient. La plupart du temps, vous allez utiliser un petit nombre de plans abstraits destinés à résoudre des problèmes spécifiques.

Examinez donc ces plans régulièrement afin de vérifier s'ils continuent d'être mieux adaptés que les solutions de l'optimiseur.

Comparaison des plans « avant » et « après »

Les plans abstraits peuvent servir à évaluer l'impact d'une mise à jour d'Adaptive Server ou d'une optimisation du système sur vos plans d'exécution de requêtes. Vous devez enregistrer les plans avant que ces modifications n'interviennent, procéder à la mise à niveau ou à l'optimisation puis enregistrer à nouveau les plans afin de pouvoir les comparer aux précédents. Les opérations sont généralement les suivantes :

- 1 Activez le mode de capture au niveau du serveur en définissant le paramètre de configuration `abstract plan dump` sur 1. Tous les plans sont alors capturés dans le groupe par défaut `ap_stdout`.
- 2 Prévoyez un temps suffisant pour que les plans capturés soient représentatifs des requêtes exécutées sur le système. Vous pouvez vérifier si des plans supplémentaires sont générés en contrôlant le nombre de lignes dans le groupe `ap_stdout` dans `sysqueryplans` ; ce nombre doit être stable :

```
select count(*) from sysqueryplans where gid = 2
```

- 3 Copiez tous les plans de ap_stdout dans ap_stdin (ou un autre groupe si vous ne souhaitez pas utiliser le mode de chargement au niveau du serveur), à l'aide de la procédure sp_copy_all_qplans.
- 4 Supprimez tous les plans d'exécution des requêtes du groupe ap_stdout, au moyen de sp_drop_all_qplans.
- 5 Effectuez les modifications de mise à niveau ou d'optimisation.
- 6 Prévoyez un temps suffisant pour que les plans soient capturés dans ap_stdout.
- 7 Comparez les plans dans ap_stdout et ap_stdin, en utilisant le paramètre de mode diff de sp_cmp_all_qplans. Par exemple, la requête suivante compare tous les plans qui se trouvent dans ap_stdout et ap_stdin :

```
sp_cmp_all_qplans ap_stdout, ap_stdin, diff
```

Cette requête n'affiche que des informations sur les plans différents.

Conséquences de l'activation du mode de capture au niveau du serveur

Lorsque le mode de capture au niveau du serveur est activé, les plans de toutes les requêtes optimisables sont enregistrés dans toutes les bases de données du serveur. Les incidences éventuelles au niveau de l'administration du système sont notamment :

- Lorsque les plans sont capturés, ils sont enregistrés dans sysqueryplans et des enregistrements de fichier de trace sont générés. La quantité d'espace requis par les plans et les enregistrements de trace dépend de la taille et de la complexité des instructions SQL et des plans d'exécution de requêtes. Vérifiez l'espace libre dans chacune des bases auxquelles les utilisateurs se connecteront.

Vous devez effectuer des enregistrements fréquents du journal des transactions, notamment dans les premières phases de la capture au niveau du serveur, pendant lesquelles de nombreux plans sont générés.

- Si des utilisateurs exécutent des procédures système à partir de la base de données master et si `installmaster` a été chargée alors que la capture au niveau du serveur était activée, les plans relatifs aux instructions optimisables dans les procédures système sont enregistrés dans `master..sysqueryplans`.

Cette règle est également vraie pour les procédures définies par l'utilisateur et créées au moment où la capture des plans était activée. Si l'espace dans `master` est limité, vous pouvez définir une base de données par défaut qui sera proposée à tous les utilisateurs, y compris les administrateurs système, au moment de la connexion.

- La table `sysqueryplans` utilise un plan de verrouillage des lignes de données (datarows) afin de réduire les conflits de verrouillage. Cependant, en cas notamment d'enregistrement d'un grand nombre de plans nouveaux, vous risquez de constater une légère dégradation des performances.
- Lorsque le mode de capture au niveau du serveur est activé, l'utilisation de `bcp` entraîne l'enregistrement des plans d'exécution des requêtes dans la base de données `master`. Si vous exécutez la commande `bcp` avec un grand nombre de tables ou de vues, vérifiez la table `sysqueryplans` et le journal de transactions dans la base `master`.

Temps et espace requis pour copier des plans

Si le nombre de plans d'exécution des requêtes dans `ap_stdout` est élevé, vérifiez que l'espace libre sur le segment système est suffisant avant de lancer la copie. Utilisez la procédure `sp_spaceused` pour contrôler la taille de `sysqueryplans` et `sp_helpsegment` pour vérifier la taille du segment système.

La copie des plans exige aussi de l'espace dans le journal de transactions.

`sp_copy_all_qplans` appelle `sp_copy_qplan` pour chaque plan du groupe à copier. Si `sp_copy_all_qplans` échoue en raison d'un manque de place ou pour tout autre motif, les plans copiés jusque là restent dans le groupe de destination.

Plans abstraits pour des procédures stockées

Pour capturer les plans abstraits relatifs à des instructions SQL optimisables dans des procédures stockées :

- Les procédures doivent être créées pendant que le mode capture ou association de plans est activé. (Ceci permet d'enregistrer le texte de la procédure dans sysprocedures.)
- La procédure doit être exécutée avec le mode capture activé et elle doit être lue à partir du disque, non du cache de procédures.

La série d'opérations ci-dessous permet de capturer le texte de requête et les plans abstraits pour toutes les instructions optimisables dans la procédure :

```
set plan dump dev_plans on
go
create procedure myproc as ...
go
exec myproc
go
```

Si la procédure est dans le cache, de sorte que les plans qui la concernent ne sont pas capturés, vous pouvez exécuter la procédure with recompile. De même, une fois qu'une procédure stockée a été exécutée avec un plan de requête abstrait, le plan qui est dans le cache de la procédure est utilisé afin d'éviter toute association à moins de lire la procédure à partir du disque.

Procédures et appartenance des plans

Lorsque le mode capture de plans est activé, les plans abstraits concernant les instructions optimisables dans une procédure stockée sont enregistrés avec l'ID d'utilisateur qui est propriétaire de la procédure.

En mode association de plans, l'association pour les procédures stockées s'appuie sur l'ID utilisateur du propriétaire de la procédure et non sur l'ID de celui qui exécute la procédure. Par conséquent, une fois qu'un plan abstrait de requête a été créé pour une procédure, tous les utilisateurs qui possèdent l'autorisation d'exécuter celle-ci vont utiliser le même plan abstrait.

Optimisation et procédures avec des chemins d'exécution variables

L'exécution d'une procédure stockée a pour effet l'enregistrement des plans abstraits relatifs aux instructions optimisables, même si la procédure contient des instructions de contrôle du flux qui entraînent l'exécution d'autres d'instructions sur la base des paramètres transmis ou d'autres conditions. Si la requête s'exécute une seconde fois avec des paramètres différents qui spécifient un autre chemin d'accès à un code, les plans relatifs aux instructions optimisables ont de toute façon été enregistrés lors de l'exécution précédente et le plan abstrait correspondant est associé à la requête.

Toutefois, les plans abstraits de procédures ne résolvent pas les problèmes propres aux procédures dont les instructions sont optimisées différemment selon les paramètres ou les conditions. C'est le cas par exemple d'une procédure où les utilisateurs indiquent des valeurs minimale et maximale dans une clause `between`, avec une requête comme celle-ci :

```
select title_id  
from titles  
where price between @lo and @hi
```

Selon les paramètres, le plan le plus adapté peut être un accès par index ou un balayage de table. Pour ces procédures, le plan abstrait peut spécifier l'une ou l'autre des méthodes d'accès possibles en fonction des paramètres présents au moment de la première exécution de la procédure. Pour plus d'informations sur l'optimisation des procédures, reportez-vous aux premiers chapitres de ce guide.

Requêtes ad hoc et plans abstraits

La capture des plans abstraits enregistre le texte complet de l'instruction SQL et l'association des plans s'appuie sur le texte complet de la requête SQL. Si certains utilisateurs soumettent des instructions SQL ad-hoc au lieu des procédures stockées ou Embedded SQL, des plans abstraits sont enregistrés pour chaque combinaison de clauses de la requête. Il peut en résulter un nombre excessif de plans abstraits.

Ainsi, si des utilisateurs vérifient le prix d'un objet `title_id` avec des instructions `select`, un plan abstrait sera enregistré pour chaque instruction. Les deux requêtes suivantes génèrent chacune un plan abstrait :

```
select price from titles where title_id = "T19245"  
select price from titles where title_id = "T40007"
```

En outre, il existe un plan par utilisateur, de sorte que si plusieurs utilisateurs vérifient le title_id T40007, un plan est enregistré pour chacun d'eux.

En revanche, si ces requêtes figurent dans des procédures stockées, vous obtenez un double avantage :

- Un seul plan abstrait est enregistré, par exemple pour la requête ci-après :

```
select price from titles where title_id =  
@title_id
```

- Le plan est sauvegardé avec l'ID de l'utilisateur propriétaire de la procédure stockée et une association de plan abstrait est établie sur la base de cet ID de propriétaire.

Avec Embedded SQL, seul le plan abstrait est enregistré avec la variable hôte :

```
select price from titles  
where title_id = :host_var_id
```


Création et utilisation des plans abstraits

Ce chapitre présente les commandes utilisées pour capturer des plans abstraits et associer des requêtes SQL entrantes aux plans sauvegardés. Les utilisateurs peuvent exécuter des commandes pour capturer et charger des plans au niveau d'une session ; de son côté, l'administrateur système peut activer les modes capture et association de plans abstraits au niveau du serveur. Ce chapitre décrit aussi comment utiliser SQL pour spécifier les plans abstraits.

Sujet	Page
Utilisation des commandes set pour capturer et associer des plans	381
Option set plan exists check	386
Utilisation d'autres options set avec les plans abstraits	387
Modes capture et association de plans au niveau du serveur	389
Création de plans avec SQL	389

Utilisation des commandes set pour capturer et associer des plans

Au niveau d'une session, il est possible d'activer ou de désactiver la capture et l'utilisation des plans abstraits avec les commandes `set plan dump` et `set plan load`. La commande `set plan replace` permet d'indiquer si les plans modifiés doivent ou non remplacer ceux qui existent déjà.

L'activation et la désactivation des modes d'utilisation des plans abstraits prennent effet à la fin du batch qui contient la commande (comme avec `showplan`). Par conséquent, il est préférable de changer de mode dans un batch distinct, avant d'exécuter vos requêtes :

```
set plan dump on
go
/*queries to run*/
go
```

Les commandes set plan utilisées dans une procédure stockée n'ont aucune incidence sur la procédure dans laquelle elles se trouvent, mais elles continuent d'être actives même une fois la procédure terminée.

Activation du mode capture de plans avec **set plan dump**

La commande set plan dump active et désactive la capture des plans abstraits. Vous pouvez sauvegarder les plans dans le groupe par défaut, ap_stdout, en utilisant set plan dump sans nom de groupe :

```
set plan dump on
```

Pour démarrer la capture des plans dans un groupe spécifique, spécifiez le nom de ce dernier. L'exemple ci-dessous définit le groupe dev_plans comme groupe de capture :

```
set plan dump dev_plans on
```

Le groupe spécifié doit déjà exister au moment où la commande set est émise. La procédure système sp_add_qpgroup crée des groupes de plans abstraits ; seul l'administrateur système ou le propriétaire de la base de données a le droit de l'utiliser. Une fois que le groupe existe, les utilisateurs peuvent y sauvegarder des plans. Pour plus d'informations sur la création d'un groupe de plans, reportez-vous à la section « [Création d'un groupe](#) », page 394.

Pour désactiver le mode capture de plans, utilisez :

```
set plan dump off
```

Il n'est pas nécessaire de préciser le nom d'un groupe pour désactiver la capture. Un seul groupe à la fois peut être actif pour vos opérations de sauvegardes ou de comparaisons des plans. Par conséquent, si vous voulez changer de groupe, vous devez désactiver le mode dump pour le groupe actif, puis le réactiver pour le nouveau groupe, comme indiqué ci-après :

```
set plan dump on /*save to the default group*/
go
/*some queries to be captured */
go
set plan dump off
go
set plan dump dev_plans on
go
/*additional queries*/
go
```

L'utilisation de la commande `use database` pendant que `set plan dump` est active a pour effet de désactiver le mode sauvegarde des plans.

Association de requêtes à des plans enregistrés

La commande `set plan load` active et désactive la fonction d'association de requêtes à des plans abstraits enregistrés.

Pour démarrer le mode association sur le groupe par défaut `ap_stdin`, utilisez la commande :

```
set plan load on
```

Pour activer le mode association sur un autre groupe de plans abstraits, spécifiez le nom du groupe :

```
set plan load test_plans on
```

Un seul groupe à la fois peut être actif pour les opérations d'association. Ainsi, si vous voulez activer le mode association pour un autre groupe que celui en cours, vous devez désactiver ce dernier et spécifier l'association sur le nouveau groupe, comme suit :

```
set plan load test_plans on
go
/*some queries*/
go
set plan load off
go
set plan load dev_plans on
go
```

L'utilisation de la commande `use database` pendant que `set plan load` est active a pour effet de désactiver le mode chargement des plans.

Utilisation du mode remplacement durant la capture de plans

Pendant que le mode capture de plans est actif, vous pouvez demander que les plans relatifs à une même requête remplacent les plans existants, grâce à la commande `set plan replace`. Celle-ci active le mode remplacement des plans :

```
set plan replace on
```

Vous n'avez pas à spécifier un nom de groupe avec `set plan replace` ; cette commande s'applique au groupe de capture actif.

Pour désactiver le mode remplacement :

```
set plan replace off
```

L'utilisation de la commande `use database` en même temps que `set plan replace` désactive le mode remplacement des plans.

Quand utiliser le mode remplacement

Si, lors de la capture de plans, une requête contient le même texte qu'un plan déjà sauvegardé, le plan existant n'est remplacé que si le mode `replace` est activé. S'il existe déjà des plans abstraits pour certaines requêtes et que vous effectuez dans la base de données des modifications physiques influant sur les choix de l'optimiseur, vous devez remplacer les plans existants en fonction des changements introduits.

Les opérations qui peuvent exiger le remplacement des plans sont notamment les suivantes :

- ajout ou suppression d'index, ou modification des clés ou de l'ordre des clés dans les index,
- modification du partitionnement d'une table,
- ajout ou suppression de groupes de buffers,
- modification des paramètres de configuration pouvant se répercuter sur des plans d'exécution de requêtes.

Dans la plupart des cas de plans à remplacer, n'activez pas `plan load`. Lorsque l'association de plans est active, les spécifications de plans sont utilisées comme données d'entrée pour l'optimiseur. Prenons l'exemple d'un plan d'exécution complet qui inclut la propriété `prefetch` avec une taille d'E/S définie à 2 Ko : si vous avez créé une zone de 16 Ko et que vous voulez remplacer la spécification de prélecture dans le plan, n'activez pas le mode `plan load`.

Vous pouvez vérifier les plans d'exécution de requêtes et remplacer certains plans abstraits à mesure que la répartition des données change dans les tables, ou après la reconstruction d'index, la mise à jour de statistiques ou le changement du plan de verrouillage.

Utilisation simultanée des modes dump, load et replace

Les modes pan dump et plan load peuvent être actifs simultanément, avec le mode replace également actif ou non.

Utilisation de *dump* et *load* sur le même groupe

Si vous avez activé dump et load sur le même groupe, sans que le mode replace soit activé :

- S'il existe un plan d'exécution de requête adéquat, il est chargé et utilisé afin d'optimiser la requête.
- S'il existe un plan inadéquat (du fait de la suppression d'un index, par exemple), un nouveau plan est généré, qui sert à optimiser la requête, mais il n'est pas sauvegardé.
- Si le plan est partiel, un plan complet est généré mais le plan partiel existant n'est pas remplacé.
- S'il n'existe pas de plan pour la requête, un plan est généré et sauvegardé.

Lorsque le mode replace est activé :

- S'il existe un plan d'exécution de requête adéquat, il est chargé et utilisé afin d'optimiser la requête.
- Si le plan n'est pas adéquat, un nouveau plan est généré, qui sert à optimiser la requête et qui remplace l'ancien.
- Si le plan est partiel, un plan complet est généré et utilisé et il remplace le plan partiel existant. Les spécifications du plan partiel sont utilisées comme données d'entrée pour l'optimiseur.
- S'il n'existe pas de plan pour la requête, un plan est généré et sauvegardé.

Utilisation de *dump* et *load* sur des groupes différents

Si le mode dump est activé sur un groupe et load pour un autre, sans que le mode replace soit activé :

- S'il existe un plan adéquat pour la requête dans le groupe de chargement, il est chargé et utilisé. Le plan est sauvegardé dans le groupe de sauvegarde, à moins qu'il en existe déjà un pour la requête dans ce groupe.

- Si le plan dans le groupe de chargement n'est pas correct, un nouveau plan est généré. Ce dernier est sauvegardé dans le groupe de sauvegarde, à moins qu'il en existe déjà un pour la requête dans ce groupe.
- Si le plan dans le groupe de chargement est partiel, un plan complet est généré et sauvegardé dans le groupe de sauvegarde, à moins qu'il en existe déjà un. Les spécifications du plan partiel sont utilisées comme données d'entrée pour l'optimiseur.
- S'il n'existe pas de plan pour la requête dans le groupe de chargement, le plan est généré et sauvegardé dans le groupe de sauvegarde, à moins qu'il en existe déjà un dans ce groupe.

Lorsque le mode `replace` est activé :

- S'il existe un plan adéquat pour la requête dans le groupe de chargement, il est chargé et utilisé.
- Si le plan dans le groupe de chargement n'est pas correct, un nouveau plan est généré et utilisé pour optimiser la requête. Le plan est sauvegardé dans le groupe de sauvegarde.
- Si le plan dans le groupe de chargement est partiel, un plan complet est généré et sauvegardé dans le groupe de sauvegarde. Les spécifications du plan partiel sont utilisées comme données d'entrée pour l'optimiseur.
- S'il n'existe pas de plan pour la requête dans le groupe de chargement, un nouveau plan est généré. Le plan est sauvegardé dans le groupe de sauvegarde.

Option set plan exists check

Le mode `exists check` permet, durant l'association de plans d'exécution de requêtes, d'accélérer les performances lorsque les utilisateurs doivent charger des plans abstraits pour moins de 20 requêtes à partir d'un groupe donné. Lorsque les requêtes à optimiser sont peu nombreuses, l'activation du mode `exists check` accélère l'exécution de toutes les requêtes qui ne disposent d'aucun plan abstrait puisque, dans ce cas, l'étape de vérification dans la table `sysqueryplans` n'est pas nécessaire.

Lorsque les modes set plan load et set exists check sont tous les deux activés, les clés de hachage des requêtes figurant dans le groupe de chargement (20 au maximum) sont mises en cache pour l'utilisateur. Si ce groupe contient plus de 20 requêtes, le mode exists check est désactivé. Chaque requête entrante est soumise à un hachage ; si sa clé de hachage ne figure pas dans le cache des plans abstraits, c'est qu'il n'existe pas de plan pour la requête et aucune recherche n'est effectuée. Les requêtes sans plans sauvegardés présentent donc un temps de compilation plus court.

La syntaxe est la suivante :

```
set plan exists check { on | off }
```

Vous devez activer le mode chargement avant la mise en cache des clés de hachage des plans.

L'administrateur système peut configurer la mise en cache des clés de hachage au niveau du serveur avec le paramètre de configuration bstract plan cache. Pour activer la mise en cache des plans à l'échelle du serveur, utilisez :

```
sp_configure "abstract plan cache", 1
```

Utilisation d'autres options **set** avec les plans abstraits

Vous pouvez combiner d'autres options d'optimisation set avec set plan dump et set plan load.

Utilisation de **showplan**

Lorsque la commande showplan est activée et que le mode association de plans abstraits a été activé avec set plan load, showplan affiche l'ID du plan abstrait correspondant au début des résultats showplan concernant l'instruction :

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using an Abstract Plan (ID : 832005995).
```

Si vous exécutez des requêtes en ajoutant la clause plan dans une instruction SQL, showplan affiche :

```
Optimized using the Abstract Plan in the PLAN clause.
```

Utilisation de **noexec**

Le mode **noexec** permet de capturer des plans abstraits sans exécuter réellement les requêtes. Si le mode **noexec** est activé, les requêtes sont optimisées et les plans abstraits sont sauvegardés mais aucun résultat n'est renvoyé.

Pour utiliser le mode **noexec** tout en capturant des plans abstraits, exécutez les procédures nécessaires (telles que **sp_add_qpgroup**) et les autres options **set** (telles que **set plan dump**) avant d'activer le mode **noexec**. L'exemple suivant présente une série d'opérations courantes :

```
sp_add_qpgroup pubs_dev
go
set plan dump pubs_dev on
go
set noexec on
go
select type, sum(price) from titles group by type
go
```

Utilisation de **forceplan**

Lorsque l'option **set forceplan on** est activée et que l'association de requêtes l'est aussi pour la session, **forceplan** est ignorée si un plan abstrait est utilisé pour optimiser la requête. Si le plan ne précise que partiellement l'ordre de jointure :

- En premier lieu, les tables du plan abstrait sont ordonnées selon l'ordre spécifié.
- Ensuite, les tables restantes sont ordonnées comme spécifié dans la clause **from**.
- Pour finir, les deux listes de tables sont fusionnées.

Modes capture et association de plans au niveau du serveur

L'administrateur système peut activer les modes capture, association et remplacement de plans à l'échelle du serveur, en se servant des paramètres de configuration ci-dessous :

- abstract plan dump : active la sauvegarde dans le groupe de capture des plans abstraits par défaut, ap_stdout.
- abstract plan load : active le chargement à partir du groupe de chargement des plans abstraits par défaut, ap_stdin.
- abstract plan replace : lorsque le mode sauvegarde est également activé, ce paramètre permet le remplacement des plans.
- abstract plan cache : active la mise en cache des ID de hachage des plans abstraits ; abstract plan load doit également être activé. Pour de plus amples informations, reportez-vous à la section « [Option set plan exists check](#) », page 386.

Par défaut, ces paramètres de configuration sont définis à 0, de sorte que les modes capture et association sont désactivés. Pour activer un mode, définissez le paramètre correspondant à 1 :

```
sp_configure "abstract plan dump", 1
```

L'activation des modes relatifs aux plans abstraits à l'échelle du serveur étant dynamique, il n'est pas nécessaire de redémarrer celui-ci.

La capture et l'association à l'échelle du serveur permettent à l'administrateur système de capturer tous les plans pour tous les utilisateurs d'un serveur. Il est impossible d'annuler les modes à l'échelle du serveur au niveau de la session.

Création de plans avec SQL

Vous pouvez directement spécifier le plan abstrait d'une requête de l'une des façons suivantes :

- en utilisant la commande create plan
- en ajoutant la clause plan aux commandes select, insert...select, update, delete et return ainsi qu'aux clauses if et while

Pour plus d'informations sur la création de plans, reportez-vous au Chapitre 15, « [Guide des plans abstraits pour les requêtes](#) ».

Utilisation de *create plan*

La commande *create plan* spécifie le texte d'une requête ainsi que le plan abstrait à sauvegarder pour celle-ci.

L'exemple ci-dessous crée un plan abstrait :

```
create plan
    "select avg(price) from titles"
"( plan
    ( i_scan type_price_ix titles )
    ( )
) "
```

Le plan est sauvegardé dans le groupe actif courant. Vous pouvez mentionner aussi le nom du groupe :

```
create plan
    "select avg(price) from titles"
"( plan
    ( i_scan type_price_ix titles )
    ( )
) "
into dev_plans
```

Si, dans le groupe courant ou le groupe indiqué, il existe déjà un plan pour la requête en question, vous devez au préalable activer le mode *replace* afin de remplacer le plan existant.

Pour voir l'ID attribué au plan que vous créez, utilisez la commande *create plan* qui peut renvoyer cet ID sous la forme d'une variable. Mais auparavant, vous devez déclarer la variable. L'exemple ci-dessous renvoie l'ID de plan :

```
declare @id int
create plan
    "select avg(price) from titles"
"( plan
    ( i_scan type_price_ix titles )
    ( )
) "
into dev_plans
and set @id
select @id
```

Lorsque vous utilisez `create plan`, la requête contenue dans le plan n'est pas exécutée. Par conséquent :

- Le texte de la requête n'est pas analysé et aucun contrôle n'est effectué sur sa syntaxe SQL.
- Il n'y a pas de vérification des plans et donc pas de contrôle de la syntaxe.
- Il n'y a pas de vérification de la compatibilité des plans avec le texte SQL.

Pour éviter toute erreur, vous devez exécuter immédiatement la requête spécifiée en activant `showplan`.

Utilisation de la clause `plan`

En ajoutant la clause `plan` aux instructions SQL suivantes, vous pouvez préciser le plan à utiliser pour exécuter la requête :

- `select`
- `insert...select`
- `delete`
- `update`
- `if`
- `while`
- `return`

L'exemple ci-dessous spécifie le plan à utiliser pour la requête :

```
select avg(price) from titles
      plan
    " ( plan
      ( i_scan type_price_ix titles )
      ( )
    ) "
```

Lorsque vous désignez un plan abstrait pour une requête, celle-ci est exécutée conformément à ce plan. Si l'option `showplan` est activée, le message suivant s'affiche :

```
Optimized using the Abstract Plan in the PLAN clause.
```

Lorsque vous utilisez la clause `plan` avec une requête, les erreurs susceptibles de figurer dans le texte SQL et dans la syntaxe du plan, ainsi que toute autre incohérence entre le plan et le texte SQL, sont signalées comme erreurs. Par exemple, le plan ci-dessous omet les parenthèses vides qui correspondent à l'étape de renvoi de l'agrégat :

```
/* step missing! */
select avg(price) from titles
      plan
" ( plan
      ( i_scan type_price titles )
 ) "
```

Le message suivant est renvoyé :

```
Msg 1005, Level 16, State 1:
Server 'smj', Line 2:
Abstract Plan (AP) : The number of operands of the PLAN operator in the AP
differs from the number of steps needed to compute the query. The extra items
will be ignored. Check the AP syntax and its correspondence to the query.
```

Les plans spécifiés avec la clause `plan` sont sauvegardés dans `sysqueryplans` uniquement si le mode capture est activé. S'il existe déjà un plan dans le groupe capture actif, vous devez activer le mode `replace` afin que l'ancien plan soit remplacé.

Gestion des plans abstraits avec des procédures système

Ce chapitre présente les fonctionnalités fondamentales et l'utilisation des procédures système avec des plans abstraits. Pour plus d'informations sur chaque procédure, reportez-vous au Manuel de référence d'Adaptive Server.

Sujet	Page
Procédures système pour gérer les plans abstraits	393
Gestion d'un groupe de plans abstraits	394
Recherche de plans abstraits	398
Gestion des plans abstraits individuels	399
Gestion de tous les plans d'un groupe	403
Importation et exportation de groupes de plans	407

Procédures système pour gérer les plans abstraits

Les procédures système permettant de gérer les plans abstraits opèrent sur des plans individuels ou sur des groupes de plans abstraits.

- [Gestion d'un groupe de plans abstraits](#)
 - `sp_add_qpgroup`
 - `sp_drop_qpgroup`
 - `sp_help_qpgroup`
 - `sp_rename_qpgroup`
- [Recherche de plans abstraits](#)
 - `sp_find_qplan`

- Gestion des plans abstraits individuels
 - sp_help_qplan
 - sp_copy_qplan
 - sp_drop_qplan
 - sp_cmp_qplans
 - sp_set_qplan
- Gestion de tous les plans d'un groupe
 - sp_copy_all_qplans
 - sp_cmp_all_qplans
 - sp_drop_all_qplans
- Importation et exportation de groupes de plans
 - sp_export_qpgroup
 - sp_import_qpgroup

Gestion d'un groupe de plans abstraits

Vous pouvez utiliser des procédures système pour créer, supprimer, renommer un groupe de plans abstraits, ainsi que pour afficher des informations sur celui-ci.

Création d'un groupe

`sp_add_qpgroup` crée et nomme un groupe de plans abstraits. A moins que vous n'utilisiez le groupe de capture par défaut, `ap_stdout`, vous devez créer un groupe de plans avant de commencer la capture. Par exemple, pour permettre la sauvegarde des plans dans un groupe nommé `dev_plans`, créez d'abord le groupe, puis exécutez la commande `set plan dump` en spécifiant le nom du groupe :

```
sp_add_qpgroup dev_plans
set plan dump dev_plans on
/*SQL queries to capture*/
```

Seul l'administrateur système ou le propriétaire de la base de données peut ajouter des groupes de plans abstraits. Une fois qu'un groupe est créé, les utilisateurs peuvent y sauvegarder des plans ou charger ceux qui s'y trouvent.

Suppression d'un groupe

`sp_drop_qpgroup` supprime un groupe de plans abstraits.

Toutefois, les restrictions suivantes s'appliquent à la procédure `sp_drop_qpgroup` :

- Seul l'administrateur système ou le propriétaire de la base de données peut supprimer des groupes de plans abstraits.
- Vous ne pouvez pas supprimer un groupe contenant des plans. Pour supprimer tous les plans qui s'y trouvent, utilisez `sp_drop_all_qplans`, en précisant le nom du groupe.
- Vous ne pouvez pas supprimer les groupes de plans abstraits par défaut `ap_stdin` et `ap_stdout`.

Cette commande supprime le groupe de plans `dev_plans` :

```
sp_drop_qpgroup dev_plans
```

Informations relatives à un groupe

`sp_help_qpgroup` affiche des informations sur un groupe de plans abstraits ou sur tous ceux qui appartiennent à une base de données.

Lorsque vous exécutez la procédure `sp_help_qpgroup` sans nom de groupe, elle affiche les noms de tous les groupes de plans abstraits, les ID de groupe et le nombre de plans dans chacun d'eux :

```
sp_help_qpgroup
Query plan groups in database 'pubtune'
  Group          GID      Plans
  -----  -----
ap_stdin           1        0
ap_stdout          2        2
p_prod             4        0
priv_test          8        1
ptest              3       51
ptest2             7      189
```

Lorsque vous utilisez `sp_help_qpgroup` avec le nom d'un groupe, le rapport fournit des statistiques sur les plans qu'il contient. L'exemple ci-dessous affiche des informations sur le groupe `ptest2` :

```
sp_help_qpgroup ptest2
Query plans group 'ptest2', GID 7

Total Rows Total QueryPlans
-----
452           189
sysqueryplans rows consumption, number of query
plans per row count
Rows          Plans
-----
5             2
3             68
2             119
Query plans that use the most sysqueryplans rows
Rows          Plan
-----
5   1932533918
5   1964534032
Hashkeys
-----
123
There is no hash key collision in this group.
```

Dans un rapport relatif à un groupe individuel, `sp_help_qpgroup` indique :

- le nombre total de plans abstraits et le nombre total de lignes dans la table `sysqueryplans` ;
- le nombre de plans contenant plusieurs lignes dans `sysqueryplans`. Ces nombres sont placés par ordre décroissant à partir du plus grand nombre de lignes ;
- les informations sur le nombre de clés de hachage et les conflits entre clés. Les plans abstraits sont associés à des requêtes par le biais d'un algorithme de hachage appliqué à la requête entière.

Lorsque l'administrateur système ou le propriétaire de la base de données exécute la procédure `sp_help_qpgroup`, celle-ci établit un rapport sur tous les plans de la base de données ou du groupe spécifié. En revanche, lorsque d'autres utilisateurs exécutent cette procédure, elle n'analyse que les plans qu'ils possèdent.

`sp_help_qpgroup` propose plusieurs modes d'affichage des informations. Ces modes sont les suivants :

Mode	Informations renvoyées
full	Nombre de lignes et nombre de plans du groupe, nombre de plans utilisant deux lignes ou plus, nombre de lignes et d'ID de plan pour les plans les plus longs et nombre de clés de hachage et informations relatives aux conflits entre clés de hachage. Il s'agit du mode d'affichage par défaut.
stats	Toutes les informations issues du rapport « full », à l'exception des informations relatives aux clés de hachage.
hash	Le nombre de lignes et de plans abstraits dans le groupe, le nombre de clés de hachage, avec des informations sur les conflits entre ces clés.
list	Le nombre de lignes et de plans abstraits dans le groupe, ainsi que les éléments suivants pour chaque association requête/plan : clé de hachage, ID de plan, les premiers caractères de la requête et les premiers caractères du plan.
queries	Le nombre de lignes et de plans abstraits dans le groupe, ainsi que les éléments suivants pour chaque requête : clé de hachage, ID de plan et les premiers caractères de la requête.
plans	Le nombre de lignes et de plans abstraits dans le groupe, ainsi que les éléments suivants pour chaque plan : clé de hachage, ID de plan et les premiers caractères du plan.
counts	Le nombre de lignes et de plans abstraits dans le groupe, ainsi que les éléments suivants pour chaque plan : nombre de lignes, nombre de caractères, clé de hachage, ID de plan et les premiers caractères de la requête.

L'exemple ci-dessous reproduit les informations renvoyées en mode `counts` :

```
sp_help_qpgroup ptest1, counts
Query plans group 'ptest1', GID 3

Total Rows   Total QueryPlans
-----      -----
        48           19

Query plans in this group

Rows   Chars     hashkey      id      query
-----  -----  -----
    3       623  1801454852  876530156  select title from titles ...
    3       576  476063777  700529529  select au_lname, au_fname...
    3       513  444226348  652529358  select aul.au_lname, aul....
```

```
3      470    792078608    716529586 select au_lname, au_fname...
3      430    789259291    684529472 select au1.au_lname, au1....
3      425    1929666826   668529415 select au_lname, au_fname...
3      421    169283426    860530099 select title from titles ...
3      382    571605257    524528902 select pub_name from publ...
3      355    845230887    764529757 delete salesdetail where ...
3      347    846937663    796529871 delete salesdetail where ...
2      379    1400470361   732529643 update titles set price =...
```

Modification du nom d'un groupe

L'administrateur système ou le propriétaire de la base de données peut renommer un groupe de plans abstraits avec la procédure `sp_rename_qpgroup`. L'exemple ci-après remplace le nom du groupe `dev_plans` par `prod_plans` :

```
sp_rename_qpgroup dev_plans, prod_plans
```

Le nouveau nom ne peut pas être identique à celui d'un groupe existant.

Recherche de plans abstraits

`sp_find_qplan` explore à la fois le texte de la requête et celui du plan pour trouver les plans correspondant à une chaîne de recherche donnée.

Dans l'exemple ci-dessous, la procédure recherche tous les plans dans lesquels la requête inclut la chaîne « `from titles` » :

```
sp_find_qplan "%from titles%"
```

L'exemple suivant recherche tous les plans abstraits qui effectuent un balayage de table :

```
sp_find_qplan "%t_scan%"
```

Lorsque l'administrateur système ou le propriétaire de la base de données exécute la procédure `sp_find_qplan`, celle-ci examine les plans appartenant à tous les utilisateurs et affiche les informations correspondantes.

En revanche, lorsque d'autres utilisateurs exécutent la procédure, elle n'analyse que les plans qu'ils possèdent.

Pour effectuer une recherche dans un seul groupe de plans abstraits, spécifiez son nom à la suite de `sp_find_qplan`. Dans l'exemple suivant, la procédure explore uniquement le groupe `test_plans` à la recherche de tous les plans qui utilisent un index :

```
sp_find_qplan "%i_scan title_id_ix%", test_plans
```

Pour chaque plan correspondant, `sp_find_qplan` imprime l'ID du groupe, l'ID du plan, le texte de la requête et le texte du plan abstrait.

Gestion des plans abstraits individuels

Vous pouvez utiliser des procédures système pour afficher la requête et le texte de plans n'appartenant pas à un groupe, les copier, les supprimer ou les comparer, ou encore changer le plan associé à une requête.

Affichage d'un plan

`sp_help_qplan` affiche des données sur des plans abstraits individuels. Cette procédure offre trois types d'affichage : brief, full et list. Le mode brief ne présente que les 78 premiers caractères de la requête et du plan ; le mode full affiche le plan et la requête en entier et list affiche uniquement les 20 premiers caractères.

Cet exemple imprime le rapport en mode « brief » :

```
sp_help_qplan 588529130
gid          hashkey        id
-----      -----
8           1460604254   588529130
query
-----
select min(price) from titles
plan
-----
( plan
  ( i_scan type_price titles )
  ( )
)
( prop titles
  ( parallel ...
```

L'administrateur système ou le propriétaire de la base de données peut utiliser `sp_help_qplan` pour afficher des informations sur tous les plans de la base. Les autres utilisateurs ne peuvent afficher que les plans qu'ils possèdent.

`sp_help_qpgroup` donne des informations sur tous les plans d'un groupe. Pour de plus amples informations, reportez-vous à la section « [Informations relatives à un groupe](#) », page 395.

Copie d'un plan dans un autre groupe

`sp_copy_qplan` copie un plan abstrait d'un groupe à un autre. Dans cet exemple, le plan 316528161 est copié de son groupe actuel au groupe `prod_plans` :

```
sp_copy_qplan 316528161, prod_plans
```

`sp_copy_qplan` vérifie que la requête n'existe pas déjà dans le groupe de destination. En cas de conflit, il exécute `sp_cmp_qplans` pour vérifier les plans dans le groupe de destination. Outre le message imprimé par `sp_cmp_qplans`, `sp_copy_qplan` imprime des messages lorsque :

- la requête et le plan que vous cherchez à copier existent déjà dans le groupe de destination ;
- un autre plan du groupe possède le même ID utilisateur et la même clé de hachage ;
- un autre plan du groupe possède la même clé de hachage mais les requêtes sont différentes.

En cas de conflit entre clés de hachage, le plan est copié. Si le plan existe dans le groupe de destination ou si sa copie entraîne un conflit entre clés d'association, il n'est pas copié. Les messages générés par `sp_copy_qplan` contiennent l'ID du plan figurant dans le groupe de destination, de sorte que vous pouvez utiliser `sp_help_qplan` pour vérifier la requête et le plan concernés.

L'administrateur système ou le propriétaire de la base de données peut copier des plans abstraits, quels qu'ils soient. En revanche, les autres utilisateurs ne peuvent copier que les plans qu'ils possèdent. Le plan et le groupe d'origine ne sont pas modifiés par la procédure `p_copy_qplan`. Le plan copié reçoit un nouvel ID de plan, l'ID du groupe de destination et l'ID de l'utilisateur qui a exécuté la requête de création du plan.

Suppression d'un plan abstrait individuel

`sp_drop_qplan` supprime des plans abstraits individuels. Dans l'exemple suivant, le plan spécifié est supprimé :

```
sp_drop_qplan 588529130
```

L'administrateur système ou le propriétaire de la base de données peut supprimer des plans abstraits de la base de données, quels qu'ils soient. Les autres utilisateurs n'ont le droit de supprimer que les plans qu'ils possèdent.

Pour trouver des ID de plans abstraits, utilisez la procédure `sp_find_qplan` qui effectue une recherche des plans à partir d'un modèle de requête ou de plan, ou bien la procédure `sp_help_qpgroup` qui répertorie les plans d'un groupe.

Comparaison de deux plans abstraits

Lorsque vous avez deux ID de plan, `sp_cmp_qplans` compare les deux plans et les requêtes associées. Par exemple :

```
sp_cmp_qplans 588529130, 1932533918
```

`sp_cmp_qplans` affiche un message signalant la comparaison de la requête et un second message sur le plan, comme suit :

- Pour les deux requêtes, un des messages suivants :
 - The queries are the same (Les requêtes sont identiques).
 - The queries are different (Les requêtes sont différentes).
 - The queries are different but have the same hash key (Les deux requêtes sont différentes mais elles ont la même clé de hachage).
- Pour les plans :
 - The query plans are the same (Les plans d'exécution de requêtes sont les mêmes).
 - The query plans are different (Les plans d'exécution de requêtes sont différents).

L'exemple ci-dessous compare deux plans dont les requêtes et les plans d'exécution correspondent :

```
sp_cmp_qplans 411252620, 1383780087
The queries are the same.
The query plans are the same.
```

L'exemple ci-dessous compare deux plans dont les requêtes correspondent, mais pas les plans :

```
sp_cmp_qplans 2091258605, 647777465
The queries are the same.
The query plans are different.
```

`sp_cmp_qplans` renvoie une valeur d'état indiquant les résultats de la comparaison. Ces valeurs sont répertoriées dans le [tableau 17-1](#).

Tableau 17-1 : Valeurs d'état renvoyées pour `sp_cmp_qplans`

Valeur renournée	Signification
0	Le texte de la requête et le plan abstrait sont les mêmes.
+1	Les requêtes et les clés de hachage sont différentes.
+2	Les requêtes sont différentes mais les clés de hachage sont les mêmes.
+10	Les plans abstraits sont différents.
100	Un ID de plan ou les deux n'existent pas.

L'administrateur système ou le propriétaire de la base de données peut comparer deux plans abstraits de la base de données, quels qu'ils soient. Les autres utilisateurs n'ont le droit de supprimer que les plans qu'ils possèdent.

Changement d'un plan existant

`sp_set_qplan` change le plan abstrait associé à un ID de plan existant sans modifier l'ID ou le texte de la requête. Cette procédure n'est utilisable que lorsque le texte du plan ne dépasse pas 255 caractères.

```
sp_set_qplan 588529130, "( i_scan title_ix titles)"
```

L'administrateur système ou le propriétaire de la base de données peut changer le plan abstrait pour n'importe quelle requête enregistrée. En revanche, les autres utilisateurs ne peuvent changer que des plans qu'ils possèdent.

Lorsque vous exécutez `sp_set_qplan`, il n'y a pas de vérification du plan abstrait par rapport au texte de la requête afin de déterminer si le nouveau plan est valide ou s'il existe déjà des tables et des index. Pour tester la validité du plan, vous devez donc exécuter la requête associée.

Vous pouvez également utiliser une commande `create plan` et la clause `plan` afin de spécifier le plan abstrait destiné à une requête. Reportez-vous à la section « [Création de plans avec SQL](#) », page 389.

Gestion de tous les plans d'un groupe

Les procédures système suivantes permettent de gérer des groupes de plans :

- `sp_copy_all_qplans`
- `sp_cmp_all_qplans`
- `sp_drop_all_qplans`

Copie de tous les plans d'un groupe

`sp_copy_all_qplans` permet de copier tous les plans d'un groupe dans un autre. Dans l'exemple ci-après, tous les plans du groupe `test_plans` sont copiés dans le groupe `helpful_plans` :

```
sp_copy_all_qplans test_plans, helpful_plans
```

Le groupe `helpful_plans` doit déjà exister au moment où vous exécutez `sp_copy_all_qplans`. Il peut contenir d'autres plans.

`sp_copy_all_qplans` copie chaque plan du groupe en exécutant `sp_copy_qplan`, ce qui implique des risques d'échec de l'opération pour les mêmes raisons qu'avec `sp_copy_qplan`. Reportez-vous à la section « [Comparaison de deux plans abstraits](#) », page 401.

Chaque copie de plan fait l'objet d'une transaction distincte et l'échec d'une opération de copie d'un plan n'entraîne pas l'échec de toute la procédure `sp_copy_all_qplans`. Si `sp_copy_all_qplans` échoue pour une quelconque raison et si vous devez la redémarrer, des messages signalent les plans qui ont pu être copiés dans le groupe de destination.

Un nouvel ID est attribué à chaque plan copié. Les plans copiés conservent l'ID de l'utilisateur d'origine. Pour copier des plans abstraits et leur attribuer de nouveaux ID utilisateur, vous devez exécuter `sp_export_qpgroup` et `sp_import_qpgroup`. Reportez-vous à la section « Importation et exportation de groupes de plans », page 407.

L'administrateur système ou le propriétaire de la base de données peut copier tous les plans de la base de données, quels qu'ils soient. En revanche, les autres utilisateurs ne peuvent copier que les plans qu'ils possèdent.

Comparaison entre tous les plans d'un groupe

`sp_cmp_all_qplans` compare tous les plans abstraits contenus dans deux groupes et affiche les informations suivantes :

- le nombre de plans qui sont identiques dans les deux groupes ;
- le nombre de plans qui ont la même clé d'association mais différents plans abstraits ;
- le nombre de plans qui figurent dans un groupe et pas dans l'autre.

L'exemple ci-dessous compare les plans dans `ap_stdout` et `ap_stdin` :

```
sp_cmp_all_qplans ap_stdout, ap_stdin
If the two query plans groups are large, this might take some time.
```

```
Query plans that are the same
```

```
count
```

```
-----
```

```
338
```

```
Different query plans that have the same association key
```

```
count
```

```
-----
```

```
25
```

```
Query plans present only in group 'ap_stdout' :
```

```
count
```

```
-----
```

```
0
```

```
Query plans present only in group 'ap_stdin' :
```

```
count
```

```
-----
```

```
1
```

En spécifiant en outre un paramètre de mode d'affichage, vous pouvez obtenir avec `sp_cmp_all_qplans` des informations sur les ID, les requêtes et les plans abstraits des requêtes figurant dans les groupes. Ce paramètre vous permet d'obtenir des informations détaillées sur tous les plans ou uniquement sur des types de différences spécifiques. Le tableau 17-2 répertorie les modes d'affichage et le type d'informations présentées pour chacun d'eux.

Tableau 17-2 : Modes d'affichage pour `sp_cmp_all_qplans`

Mode	Informations affichées
counts	Décompte de : plans identiques, plans ayant la même clé d'association, mais des groupes différents, et plans existant dans un groupe, mais pas dans l'autre. Il s'agit du mode d'affichage par défaut.
brief	Les mêmes informations que dans le mode précédent, plus les ID des plans abstraits de chaque groupe lorsqu'ils diffèrent tout en ayant la même clé d'association, et les ID des plans qui figurent dans un groupe mais pas dans l'autre.
same	Les mêmes informations que dans le premier mode, plus les ID, les requêtes et les plans d'exécution pour tous les plans abstraits dont les plans et les requêtes correspondent.
diff	Les mêmes informations que dans le premier mode, plus les ID, les requêtes et les plans d'exécution pour tous les plans abstraits dont les plans et les requêtes diffèrent.
first	Les mêmes informations que dans le premier mode, plus les ID, les requêtes et les plans d'exécution pour tous les plans abstraits qui appartiennent au premier groupe mais pas au second.
second	Les mêmes informations que dans le premier mode, plus les ID, les requêtes et les plans d'exécution pour tous les plans abstraits qui appartiennent au second groupe mais pas au premier.
offending	Les mêmes informations que dans le premier mode, plus les ID, les requêtes et les plans d'exécution pour tous les plans abstraits qui ont des clés d'association différentes ou qui n'existent dans aucun des deux groupes. Il s'agit de la combinaison des modes diff, first et second.
full	Les mêmes informations que dans le premier mode, plus les ID, les requêtes et les plans d'exécution pour tous les plans abstraits. Il s'agit de la combinaison des modes same et offending.

Cet exemple imprime le rapport en mode brief :

```
sp_cmp_all_qplans ptest1, ptest2, brief
If the two query plans groups are large, this might take some
time.
Query plans that are the same
count
-----
39
Different query plans that have the same association key

count
-----
4

ptest1      ptest2

id1          id2
-----
764529757  1580532664
780529814  1596532721
796529871  1612532778
908530270  1724533177
Query plans present only in group 'ptest1':


count
-----
3

id
-----
524528902
1292531638
1308531695

Query plans present only in group 'ptest2':


count
-----
1

id
-----
2108534545
```

Suppression de tous les plans abstraits d'un groupe

`sp_drop_all_qplans` supprime tous les plans abstraits d'un groupe.

Dans l'exemple ci-dessous, tous les plans abstraits du groupe `dev_plans` sont supprimés :

```
sp_drop_all_qplans dev_plans
```

Lorsque l'administrateur système ou le propriétaire de la base de données exécute `sp_drop_all_qplans`, tous les plans appartenant à tous les utilisateurs sont supprimés du groupe spécifié. En revanche, lorsque d'autres types d'utilisateur exécutent cette procédure, elle n'opère que sur les plans qu'ils possèdent.

Importation et exportation de groupes de plans

`sp_export_qpgroup` et `sp_import_qpgroup` copient des groupes de plans entre `sysqueryplans` et une table utilisateur. Avec ces procédures, l'administrateur système ou le propriétaire de la base de données peuvent :

- copier des plans abstraits d'une base dans une autre sur le même serveur ;
- créer une table dans le serveur actuellement utilisé qu'il est possible de copier, avec l'utilitaire `bcp`, dans un autre serveur ;
- attribuer des ID utilisateur différents aux plans existants dans la même base de données.

Exportation de plans vers une table utilisateur

`sp_export_qpgroup` copie tous les plans d'un utilisateur depuis un groupe de plans abstraits dans une table utilisateur. Dans l'exemple ci-dessous, la procédure copie les plans du propriétaire de la base de données (`dbo`) depuis le groupe `fast_plans`, dans une table qu'elle crée à cet effet nommée `transfer` :

```
sp_export_qpgroup dbo, fast_plans, transfer
```

`sp_export_qpgroup` utilise `select...into` pour créer une table ayant le même nombre de colonnes et les mêmes types de données que `sysqueryplans`. Si l'option `select into/bulkcopy/plisort` n'est pas activée dans la base de données, vous pouvez préciser le nom d'une autre base. La commande ci-dessous crée la table d'exportation dans `tempdb` :

```
sp_export_qpgroup mary, ap_stdout, "tempdb..mplans"
```

Vous pouvez, avec `bcp`, copier la table vers un autre serveur. Il est également possible d'importer les plans vers `sysqueryplans` dans une autre base de données sur le même serveur ou vers `sysqueryplans` sur la même base mais avec un nom de groupe ou un ID utilisateur différent.

Importation de plans depuis une table utilisateur

`sp_import_qpgroup` copie des plans à partir de tables créées par `sp_export_qpgroup`, dans un groupe défini au sein de `sysqueryplans`. Dans l'exemple ci-dessous, la procédure copie les plans de la table `tempdb..mplans` dans le groupe `ap_stdin` en utilisant l'ID utilisateur du propriétaire de la base de données :

```
sp_import_qpgroup "tempdb..mplans", dbo, ap_stdin
```

Il est impossible de copier des plans dans un groupe qui contient déjà des plans pour l'utilisateur spécifié.

Référence de langage des plans abstraits

Ce chapitre décrit les opérateurs et autres éléments du langage des plans abstraits.

Sujet	Page
Mots-clés	409
Opérandes	409
Schéma des exemples	410

Mots-clés

Les mots-clés employés dans le langage des plans abstraits de requêtes sont énumérés ci-après. Ce ne sont pas des mots réservés et ils ne sont pas en conflit avec les noms de tables ou d'index utilisés dans les plans abstraits. Par exemple, vous pouvez nommer une table ou un index hints.

Opérandes

Les opérandes suivants sont utilisés dans la syntaxe des instructions de plans abstraits :

Tableau 18-1 : Identificateurs utilisés

Identificateur	Explication
<i>nom_table</i>	Nom d'une table sous-jacente, c'est-à-dire une table système ou utilisateur
<i>alias</i>	Alias ou nom lié spécifié pour une table dans une requête
<i>table_sous-jacente</i>	Table résultant du balayage d'une table stockée
<i>table_stockée</i>	Table sous-jacente ou table de travail

<i>nom_table_travail</i>	Nom d'une table de travail
<i>nom_vue</i>	Nom d'une vue
<i>nom_index</i>	Nom d'un index
<i>id_sous-requête</i>	Entier identifiant l'ordre des sous-requêtes dans la requête

nom_table et *nom_vue* peuvent être spécifiés par la notation *base_de_données.propriétaire.nom_objet*.

Tables sous-jacentes de plan abstrait

Une table sous-jacente de plan abstrait est le résultat de l'accès à une table stockée pendant l'exécution d'une requête. Il peut s'agir de :

- du jeu de résultats généré par la requête
- d'un résultat intermédiaire pendant l'exécution de la requête ; c'est-à-dire du résultat de la jointure des deux premières tables dans l'ordre de jointure avec une troisième table.

Les tables sous-jacentes de plan abstrait résultent de l'un des opérateurs de balayage spécifiant la méthode d'accès : *scan*, *i_scan* ou *t_scan*, par exemple, (*i_scan title_id_ix titles*).

Remarque Il convient de ne pas confondre les tables sous-jacentes de plan abstrait et les tables sous-jacentes SQL. Pour plus d'informations sur les tables sous-jacentes SQL, reportez-vous au *Guide de l'utilisateur Transact-SQL*.

Schéma des exemples

Pour faciliter la lecture des exemples de plans abstraits, vous trouverez ci-dessous la liste des tables utilisées dans cette section :

```
create table t1 (c11 int, c12 int)
create table t2 (c21 int, c22 int)
create table t3 (c31 int, c32 int)
```

Les index utilisés sont les suivants :

```
create index i_c11 on t1(c11)
create index i_c12 on t1(c12)
create index i_c11_c12 on t1(c11, c12)
create index i_c21 on t2(c21)
create index i_c22 on t2(c22)
create index i_c31 on t3(c31)
create index i_c32 on t3(c32)
```

g_join

Description	Spécifie la jointure de deux ou plusieurs tables sous-jacentes de plan abstrait sans préciser le type de jointure (boucle imbriquée ou fusion).
Syntaxe	(g_join <i>table_sous-jacente1</i> <i>table_sous-jacente2</i>) (g_join (<i>table_sous-jacente1</i>) (<i>table_sous-jacente2</i>) ... (<i>table_sous-jacenteN</i>))
Paramètres	<i>table_sous-jacente1</i> ... <i>table_sous-jacenteN</i> sont les tables sous-jacentes de plan abstrait à unir.
Valeur renournée	Une table sous-jacente de plan abstrait qui est la jointure des tables sous-jacentes du plan abstrait spécifié.

Exemples

Exemple 1

```
select *
from t1, t2
where c21 = 0
and c22 = c12

( g_join
  ( i_scan i_c21 t2 )
  ( i_scan i_c12 t1 )
)
```

Dans l'ordre de jointure, la table t2 est la table externe et t1 est la table interne.

Exemple 2

```
select *
  from t1, t2, t3
 where c21 = 0
   and c22 = c12
   and c11 = c31

( g_join
  ( i_scan i_c21 t2 )
  ( i_scan i_c12 t1 )
  ( i_scan i_c31 t3 )
)
```

La table t2 est jointe à t1 et la table sous-jacente de plan abstrait est jointe à t3.

Utilisation

- L'opérateur `g_join` est un opérateur logique générique qui décrit toutes les jointures binaires (jointure interne, externe, de contrôle de présence).
- L'opérateur `g_join` n'est jamais utilisé dans des plans générés ; les opérateurs `nl_g_join` et `m_g_join` indiquent le type de jointure utilisé.
- Lorsque l'opérateur `g_join` est utilisé, l'optimiseur choisit entre une jointure à boucle imbriquée et une jointure tri-fusion. Pour spécifier la jointure tri-fusion, utilisez `m_g_join`. Pour une jointure à boucle imbriquée, utilisez `nl_g_join`.
- La syntaxe propose une notation abrégée pour décrire une jointure englobant plusieurs tables. La syntaxe suivante :

```
( g_join
  ( scan t1)
  ( scan t2)
  ( scan t3)
  ...
  ( scan tN-1)
  ( scan tN)
)
```

est la forme abrégée de :

```
( g_join
  ( g_join
    ...
    ( g_join
      (g_join
        ( scan t1)
        ( scan t2)
      )
      ( scan t3)
    )
  ...
  ( scan tN-1)
)
( scan tN)
)
```

- Si `g_join` sert à spécifier l'ordre de jointure pour un certain nombre de tables d'une requête, l'optimiseur utilise l'ordre de jointure indiqué, mais il peut insérer d'autres tables entre les opérandes `g_join`.

Par exemple, dans la requête :

```
select *
  from t1, t2, t3
  where ...
```

le plan abstrait partiel ci-dessous décrit uniquement l'ordre de jointure de `t1` et `t2` :

```
( g_join
  ( scan t2)
  ( scan t1)
)
```

L'optimiseur a le choix entre les trois ordres suivants : `t3-t2-t1`, `t2-t3-t1` or `t2-t1-t3`.

- Les tables sont jointes dans l'ordre indiqué dans la clause `g_join`.
- Lorsque l'option `set forceplan` est activée et que l'association de requêtes l'est aussi pour la session, `forceplan` est ignorée si un plan abstrait complet sert à optimiser la requête. Si le plan ne précise que partiellement l'ordre de jointure :
 - En premier lieu, les tables du plan abstrait sont ordonnées selon l'ordre spécifié.

- Ensuite, les tables restantes sont ordonnées comme spécifié dans la clause `from`.
- Pour finir, les deux listes de tables sont fusionnées.

Voir aussi

[m_g_join](#), [nl_g_join](#)

hints

Description

Introduit et regroupe des éléments d'un plan abstrait partiel.

Syntaxe

(`hints` (*table_sous-jacente*)

...
)

Paramètres

table_sous-jacente

est une ou plusieurs expressions qui génèrent une table sous-jacente de plan abstrait.

Valeur renournée

Une table sous-jacente de plan abstrait.

Exemples

```
select *  
  from t1, t2  
 where c12 = c21  
       and c11 > 0  
       and c22 < 1000
```

```
( hints  
  ( g_join  
    ( t_scan t2 )  
    ( i_scan () t1 )  
  )  
)
```

Spécifie un plan partiel qui inclut un balayage de table sur `t2`, le recours à un index sur `t1` et l'ordre de jointure `t1-t2`. Le choix de l'index pour `t1` et du type de jointure (boucle imbriquée ou tri-fusion) est effectué par l'optimiseur.

Utilisation

- Les indications spécifiées sont utilisées lors de l'optimisation de la requête.
- L'opérateur `hints` constitue en quelque sorte la racine d'un plan abstrait partiel qui inclut plusieurs indications. Si un plan partiel ne contient qu'une indication, `hints` est facultatif.

- L'opérateur hints n'apparaît pas dans les plans générés par l'optimiseur, ceux-ci étant toujours complets.
- Il est possible d'associer des indications aux requêtes :
 - en changeant le plan d'une requête existante à l'aide de la procédure `sp_set_qplan` ;
 - en spécifiant le plan d'une requête avec la clause `plan`. Pour sauvegarder la requête et les indications, vous devez activer `set plan dump` ;
 - en utilisant la commande `create plan`.
- Lorsque des indications sont précisées dans la clause `plan` pour une instruction SQL, une vérification des plans est effectuée afin de garantir leur validité. Lorsque ces indications sont spécifiées avec `sp_set_qplan`, les plans ne sont pas vérifiés avant d'être sauvegardés.

i_scan

Description

Spécifie un balayage d'index d'une table stockée.

Syntaxe

(`i_scan nom_index table_sous-jacente`)

(`i_scan () table_sous-jacente`)

Paramètres

nom_index

spécifie le nom ou ID de l'index à utiliser pour un balayage d'index de la table stockée spécifiée. Utilisez des parenthèses vides pour indiquer qu'il faut utiliser un balayage d'index au lieu d'un balayage de table, tout en laissant à l'optimiseur le soin de choisir l'index.

table_sous-jacente

est le nom de la table sous-jacente à balayer.

Valeur renournée

Une table sous-jacente de plan abstrait générée par le balayage de la table sous-jacente.

Exemples

Exemple 1

```
select * from t1 where c11 = 0
```

```
( i_scan i_c11 t1 )
```

Spécifie l'index `i_c11` à utiliser pour un balayage de `t1`.

Exemple 2

```
select *
  from t1, t2
 where c11 = 0
   and c22 = 1000
   and c12 = c21

( g_join
  ( scan t2 )
  ( i_scan () t1 )
)
```

Spécifie un plan partiel qui indique l'ordre de jointure mais laisse à l'optimiseur le soin de choisir la méthode d'accès à t2 et l'index pour t1.

```
select * from t1 where c12 = 0
( i_scan 2 t1 )
```

L'index sur t1 est identifié par son ID au lieu du nom.

Utilisation

- L'index indiqué est utilisé pour balayer la table ou, si aucun index n'est spécifié, l'optimiseur utilise un index de son choix au lieu d'effectuer un balayage de table.
- L'utilisation de parenthèses vides après l'opérateur *i_scan* permet à l'optimiseur de choisir l'index ou d'effectuer un balayage de tables si aucun index n'existe sur la table.
- Lorsque l'opérateur *i_scan* est spécifié, l'optimiseur effectue toujours un balayage avec couverture d'index si toutes les colonnes sont comprises dans l'index. Aucune spécification de plan abstrait n'est nécessaire pour décrire un balayage avec couverture d'index.
- Lorsque l'opérateur *i_scan* est utilisé, l'optimiseur n'a pas la possibilité de choisir la stratégie de reformatage ni la stratégie OR, même si l'index spécifié n'existe pas. Il choisit un autre index exploitable et un message d'information s'affiche. Si aucun index n'est spécifié pour *i_scan* ou s'il n'en existe pas, un balayage de table est effectué et un message d'information s'affiche.
- Bien que la spécification d'un index par son ID soit correcte dans les plans abstraits de requêtes, il est conseillé de l'éviter. En effet, si des index sont supprimés, puis recréés dans un ordre différent, les plans deviennent incorrects ou peu efficaces.

Voir aussi[scan, t_scan](#)

in

Description	Identifie l'emplacement d'une table spécifiée dans une sous-requête ou dans une vue.
Syntaxe	(in ([subq <i>id_sous-requête</i> view <i>nom_vue</i>]))
Paramètres	<p>subq <i>id_sous-requête</i> <i>id_sous-requête</i> est un entier identifiant la sous-requête. Dans les plans abstraits, la numérotation des sous-requêtes repose sur l'ordre des parenthèses ouvrantes de début des sous-requêtes figurant dans la requête.</p> <p>view <i>nom_vue</i> <i>nom_vue</i> spécifie le nom d'une vue. L'indication du nom de la base de données et du propriétaire dans le plan abstrait doit correspondre aux valeurs utilisées dans la requête, sinon l'association de plans ne peut pas avoir lieu.</p>
Exemples	

Exemple 1

```
create view v1 as
select * from t1

select * from v1

( t_scan ( table t1 ( in ( view v1 ) ) ) )
```

Identifie la vue dans laquelle apparaît la table t1.

Exemple 2

```
select *
from t2
where c21
in (select c12 from t1)

( g_join
  ( t_scan t2 )
  ( t_scan ( table t1 ( in ( subq 1 ) ) ) )
)
```

Identifie le balayage de la table *t1* de la sous-requête 1.

Exemple 3

```

create view v9
as
select *
from t1
where c11 in (select c21 from t2)

create view v10
as
select * from v9
where c11 in (select c11 from v9)

select * from v10, t3
where c11 in
      (select c11 from v10 where c12 = t3.c31)

(
g_join
( t_scan t3 )
( i_scan i_c21 ( table t2 ( in ( subq 1 ) ( view v9 ) ( view v10 ))))
( i_scan i_c11 ( table t1 ( in ( view v9 ) ( view v10 ))))
( i_scan i_c11 ( table t1 ( in ( view v9 ) ( view v10 ) ( subq 1 ))))
( i_scan i_c11 ( table t1 ( in ( view v9 ) ( subq 1 ) ( view v10 ))))
( i_scan i_c21 ( table t2 ( in ( subq 1 ) ( view v9 ) ( subq
1 ) ( view v10 ))))
( i_scan i_c11 ( table t1 ( in ( view v9 ) ( subq 1 ) ( view v10 ) ( subq
1 ))))
( i_scan i_c21 ( table t2 ( in ( subq 1 ) ( view v9 ) ( view v10 ) ( subq
1 ))))
( i_scan i_c21 ( table t2 ( in ( subq 1 ) ( view v9 ) ( subq 1 ) ( view
v10 ) ( subq 1 ))))
)

```

Exemple de multiples imbriques de vues et de sous-requêtes.

Utilisation

- Identifie l'occurrence d'une table dans une vue ou une sous-requête de la requête SQL.
- Dans la liste `in`, les éléments les plus imbriqués sont situés à gauche, près du nom de table (qui est elle-même l'élément le plus imbriqué) et les éléments les moins imbriqués (ceux qui apparaissent dans la requête externe) sont à droite.

Par exemple, la qualification :

```
(table t2 (in (subq 1) (view v9) (subq 1)
               (view v10) (subq 1) ) )
```

peut se lire dans les deux sens :

- de gauche à droite, en commençant au niveau de la table : la table sous-jacente t2 de la première sous-requête de la vue v9, qui est dans la première sous-requête de la vue v10, celle-ci étant dans la première sous-requête de la requête principale.
- de droite à gauche, en commençant par la requête principale : la requête principale contient une première sous-requête, qui balaye la vue v10, laquelle à son tour contient une première sous-requête balayant la vue v9, qui dispose de la même fonction pour la table sous-jacente t2.

Voir aussi

[nested](#), [subq](#), [table](#), [view](#)

lru

Description Spécifie la stratégie de mise en cache LRU pour le balayage d'une table stockée.

Syntaxe

```
( prop nom_table
    ( lru )
)
```

Paramètres

nom_table
est la table à laquelle la propriété doit s'appliquer.

Exemples

```
select * from t1
```

```
( prop t1
    ( lru )
)
```

Spécifie l'utilisation de la stratégie de mise en cache LRU pour le balayage de t1.

Utilisation

- La stratégie LRU est utilisée dans le plan de requête résultant.
- Les plans partiels peuvent spécifier des propriétés de balayage sans mentionner d'autres parties du plan d'exécution de la requête.
- Les plans complets d'exécution de requêtes incluent toujours toutes les propriétés de balayage.

Voir aussi

[mru](#), [prop](#)

m_g_join

Description

Spécifie une jointure par fusion de deux tables sous-jacentes de plan abstrait.

Syntaxe

```
( m_g_join
    ( table_sous-jacente1 )
    ( table_sous-jacente2 )
)
```

Paramètres

table_sous-jacente1...*table_sous-jacenteN*

sont les tables sous-jacentes de plan abstrait à joindre. *table_sous-jacente1* est toujours la table externe et *table_sous-jacente2* est la table interne.

Valeur renournée

Une table sous-jacente de plan abstrait qui est la jointure des tables sous-jacentes du plan abstrait spécifié.

Exemples

Exemple 1

```
select t1.c11, t2.c21
      from t1, t2, t3
      where t1.c11 = t2.c21
            and t1.c11 = t3.c31
```

```
( nl_g_join
    ( m_g_join
        ( i_scan i_c31 t3 )
        ( i_scan i_c11 t1 )
    )
    ( t_scan t2 )
)
```

Spécifie une jointure par fusion des tables t1 et t3, suivie d'une jointure à boucle imbriquée avec t2.

Exemple 2

```
select * from t1, t2, t3
where t1.c11 = t2.c21 and t1.c11 = t3.c31
and t2.c22 =7

( nl_g_join
  ( m_g_join
    ( i_scan i_c21 t2 )
    ( i_scan i_c11 t1 )
  )
  ( i_scan i_c31 t3 )
)
```

Spécifie une jointure par fusion des tables t2 (externe) et t1 (interne), suivie dans l'ordre de jointure par une jointure à boucle imbriquée t3.

Exemple 3

```
select c11, c22, c32
from t1, t2, t3
where t1.c11 = t2.c21
and t2.c22 = t3.c32

( m_g_join
  (nl_g_join
    (i_scan i_c11 t1)
    (i_scan i_c12 t2)
  )
  (i_scan i_c32_ix t3)
)
```

Spécifie une jointure à boucle imbriquée de t1 et t2, suivie d'une jointure par fusion avec t3.

Utilisation

- Les tables sont jointes dans l'ordre indiqué dans la clause `m_g_join`.
- La phase de tri et la création d'une table de travail nécessaires pour effectuer les requêtes de jointure tri-fusion ne sont pas représentées dans le plan abstrait.
- Si l'opérateur `m_g_join` est utilisé pour spécifier une jointure qu'il est impossible de réaliser par fusion, la spécification est ignorée sans que vous en soyez informé.

Voir aussi

[g_join](#), [nl_g_join](#)

mru

Description Spécifie la stratégie de mise en cache MRU pour le balayage d'une table stockée.

Syntaxe (prop *nom_table*
 (mru)
)

Paramètres *nom_table*
est la table à laquelle la propriété doit s'appliquer.

Exemples

```
select * from t1
```

```
( prop t1  
      ( mru )  
    )
```

Spécifie l'utilisation de la stratégie de mise en cache MRU pour la table.

Utilisation

- La stratégie MRU est spécifiée dans le plan d'exécution de requête résultant.
- Les plans partiels peuvent spécifier des propriétés de balayage sans mentionner d'autres parties du plan d'exécution de la requête.
- Les plans d'exécution de requêtes générés incluent toujours toutes les propriétés de balayage.
- Si sp_cachestrategy a servi à désactiver le remplacement MRU pour une table ou un index et que le plan d'exécution de la requête spécifie MRU, la spécification dans le plan abstrait est ignorée sans que vous en soyez informé.

Voir aussi

[lru](#), [prop](#)

nested

Description Décrit l'imbrication des sous-requêtes sur une table sous-jacente de plan abstrait.

Syntaxe (nested
 (*table_sous-jacente*)
 (*spécification_sous-requête*)
)

Paramètres *table_sous-jacente*
table_sous-jacente est la table sous-jacente de plan abstrait sur laquelle la sous-requête spécifiée doit être imbriquée.

spécification_sous-requête
spécification_sous-requête est la sous-requête à imbriquer sur *table_sous-jacente*

Valeur retournée Une table sous-jacente du plan abstrait.

Exemples

Exemple 1

```
select c11 from t1
where c12 =
    (select c21 from t2 where c22 = t1.c11)
```

```
( nested
  ( t_scan t1 )
  ( subq 1
    ( t_scan ( table t2 ( in ( subq 1 ) ) ) )
  )
)
```

Sous-requête imbriquée simple.

Exemple 2

```
select c11 from t1
where c12 =
    (select c21 from t2 where c22 = t1.c11)
and c12 =
    (select c31 from t3 where c32 = t1.c11)
```

```
( nested
  ( nested
    ( t_scan t1 )
    ( subq 1
      ( t_scan ( table t2 ( in ( subq 1 ) ) ) )
    )
  )
  ( subq 2
    ( t_scan ( table t3 ( in ( subq 2 ) ) ) )
  )
)
```

Les deux sous-requêtes sont imbriquées dans la requête principale.

Exemple 3

```
select c11 from t1
where c12 =
  (select c21 from t2 where c22 =
    (select c31 from t3 where c32 = t1.c11))

( nested
  ( t_scan t1 )
  ( subq 1
    ( nested
      ( t_scan ( table t2 ( in ( subq 1 ) ) ) )
      ( subq 2
        ( t_scan ( table t3 ( in ( subq 2 ) ) ) )
      )
    )
  )
)
```

Une sous-requête de niveau 2 imbriquée dans une sous-requête de niveau 1, elle-même imbriquée dans la requête principale.

Utilisation

- La sous-requête est exécutée au niveau du point d'attache indiqué dans le plan d'exécution.
- Les sous-requêtes matérialisées et mises à plat n'apparaissent pas sous un opérateur nested. Pour obtenir des exemples, reportez-vous à la section relative à subq, [page 434](#)

Voir aussi

[in](#), [subq](#)

nl_g_join

Description

Spécifie une jointure par boucles imbriquées de deux ou plusieurs tables sous-jacentes de plan abstrait.

Syntaxe

```
( nl_g_join      ( table_sous-jacente1 )
  ( table_sous-jacente2 )
  ...
  ( table_sous-jacenteN )
)
```

Paramètres	<i>table_sous-jacente1</i> ... <i>table_sous-jacenteN</i>
	sont les tables sous-jacentes de plans abstraits à unir.
Valeur retournée	Une table sous-jacente de plan abstrait qui est la jointure des tables sous-jacentes du plan abstrait spécifié.

Exemples	Exemple 1
	<pre> select * from t1, t2 where c21 = 0 and c22 = c12 (nl_g_join (i_scan i_c21 t2) (i_scan i_c12 t1)) </pre>

Dans l'ordre de jointure, la table *t2* est la table externe et *t1* est la table interne.

Exemple 2

```

select *
from t1, t2, t3
where c21 = 0
and c22 = c12
and c11 = c31

( nl_g_join
  ( i_scan i_c21 t2 )
  ( i_scan i_c12 t1 )
  ( i_scan i_c31 t3 )
)

```

La table *t2* est jointe à *t1* et la table sous-jacente du plan abstrait est jointe à *t3*.

Utilisation	<ul style="list-style-type: none"> Les tables sont jointes dans l'ordre indiqué par la clause <i>nl_g_join</i>. L'opérateur <i>nl_g_join</i> est un opérateur logique générique qui décrit toutes les jointures binaires (jointure interne, externe, ou semi-jointure). Les jointures sont exécutées en utilisant la méthode d'exécution des requêtes à boucle imbriquée.
Voir aussi	g_join , m_g_join

parallel

Description Spécifie le degré de parallélisation pour le balayage d'une table stockée.

Syntaxe
(prop *nom_table*
 (parallel *degré*)
)

Paramètres
nom_table
est la table à laquelle la propriété doit s'appliquer.

degré
est le degré de parallélisation à utiliser pour le balayage.

Exemples
`select * from t1`

```
(prop t1  
      ( parallel 5 )  
)
```

Indique que 5 processus de travail doivent être utilisés pour balayer la table t1.

- Utilisation
- Le balayage se déroule avec le nombre indiqué de processus de travail, s'ils sont disponibles.
 - Les plans partiels peuvent spécifier des propriétés de balayage sans mentionner d'autres parties du plan d'exécution de la requête.
 - Si un plan sauvegardé spécifie l'utilisation d'un nombre de processus de travail mais que ces valeurs au niveau de la session ou serveur diffèrent au moment de l'exécution de la requête :
 - Si le plan spécifie plus de processus de travail que ne l'autorisent les paramètres courants, ceux-ci sont appliqués ou la requête est exécutée avec un plan en mode série.
 - Si le plan spécifie moins de processus de travail que ne l'autorisent les paramètres courants, les valeurs indiquées dans le plan sont utilisées.

Ces modifications au plan d'exécution de requête sont transparentes pour l'utilisateur. Par conséquent, aucun message d'avertissement n'est émis.

Voir aussi

[prop](#)

plan

Description	Offre un mécanisme de regroupement des étapes à exécuter dans des requêtes qui en comportent plusieurs, telles que des requêtes nécessitant la création de tables de travail ou qui calculent des valeurs d'agrégats.
Syntaxe	(plan étape_requête1 ... étape_requêteN)
Paramètres	étape_requête1...étape_requêteN – spécifient les plans abstraits de chaque étape de la requête.
Valeur retournée	Une table sous-jacente du plan abstrait.
Exemples	

Exemple 1

```
select max(c11) from t1
group by c12
```

```
( plan
  ( store Worktab1
    ( t_scan t1 )
  )
  ( t_scan ( work_t Worktab1 ) )
)
```

Renvoie un agrégat vectoriel. Le premier opérande de l'opérateur plan crée une table de travail Worktab1 et spécifie un balayage de la table sous-jacente. Le second opérande balaie la table de travail afin de renvoyer les résultats.

Exemple 2

```
select max(c11) from t1
```

```
( plan
  ( t_scan t1 )
  ( )
)
```

Renvoie un agrégat scalaire. La dernière table sous-jacente de plan abstrait est vide car les agrégats scalaires insèrent la valeur résultante dans une variable interne et non dans une table de travail.

Exemple 3

```
select *
from t1
where c11 = (select count(*) from t2)

( plan
  ( i_scan i_c21 (table t2 ( in_subq 1 ) ) )
  ( i_scan i_c11 t1 )
)
```

Spécifie l'exécution d'une sous-requête matérialisée.

Exemple 4

```
create view v3
as
select distinct * from t3
```

```
select * from t1, v3
where c11 = c31
```

```
( plan
  ( store Worktab1
    ( t_scan (table t3 (in_view v3 ) ) )
  )
  ( nl_g_join
    ( t_scan t1 )
    ( t_scan ( work_t Worktab1 ) )
  )
)
```

	Spécifie l'exécution d'une vue matérialisée.
Utilisation	<ul style="list-style-type: none"> • L'accès aux tables se fait dans l'ordre indiqué, avec les méthodes d'accès spécifiées. • L'opérateur <code>plan</code> est obligatoire dans les requêtes à multiples étapes, notamment : <ul style="list-style-type: none"> • les requêtes qui génèrent des tables de travail, telles que des requêtes qui effectuent des tris ou celles qui calculent des agrégats vectoriels, • les requêtes qui calculent des agrégats scalaires, • les requêtes qui incluent des sous-requêtes matérialisées. • Un plan abstrait pour une requête nécessitant l'exécution de multiples étapes doit inclure des opérandes pour chaque étape de la requête si elle commence par le mot clé <code>plan</code>. Utilisez l'opérateur <code>hints</code> pour introduire les plans partiels.
Voir aussi	hints

prefetch

Description	Spécifie la taille des E/S à utiliser pour le balayage d'une table stockée.
Syntaxe	<pre>(prop nom_table (prefetch taille))</pre>
Paramètres	<p><i>nom_table</i> <i>nom_table</i> est la table à laquelle la propriété doit s'appliquer.</p> <p><i>taille</i> <i>taille</i> est la taille E/S correcte : 2, 4, 8 ou 16.</p>
Exemples	<pre>select * from t1</pre> <pre>(prop t1 (prefetch 16))</pre>
	Une taille d'E/S de 16 Ko est utilisée pour le balayage de t1.

- Utilisation
- La taille d'E/S indiquée est utilisée dans le plan d'exécution de requête résultant s'il existe un groupe de buffers de cette taille dans le cache dont se sert la table.
 - Les plans partiels peuvent spécifier des propriétés de balayage sans mentionner d'autres parties du plan d'exécution de la requête.
 - Si des spécifications d'E/S étendues dans un plan sauvegardé ne correspondent pas à la configuration courante de la zone ou à d'autres options :
 - Si le plan spécifie des E/S de 16 Ko et qu'il n'existe pas de zone de 16 Ko, la plus grande taille d'E/S suivante disponible est utilisée ;
 - Si, en raison des options existantes au niveau de la session ou du serveur, des E/S ne sont pas disponibles pour la requête (set prefetch pour la session ou sp_cachestrategy pour la table), des E/S de 2 Ko sont utilisées.
 - Si vous sauvegardez des plans qui spécifient uniquement des E/S de 2 Ko pour les propriétés de balayage et que vous créez ensuite des zones d'E/S étendues, activez le mode replace afin de sauvegarder les nouveaux plans pour qu'ils utilisent des E/S étendues.

Voir aussi

[prop](#)

prop

Description

Spécifie les propriétés à utiliser pour le balayage d'une table stockée.

Syntaxe

```
( prop nom_table
      ( spécification_propriété ) ...
    )
```

spécification_propriété :

```
( prefetch taille )
  ( lru | mru )
  ( parallel degré )
```

Paramètres

nom_table

est la table à laquelle la propriété doit s'appliquer.

Exemples

```
select * from t1
```

```
( t_scan t1 )
( prop t1
  ( parallel 1 )
  ( prefetch 16 )
  ( lru )
)
```

Affiche les valeurs des propriétés utilisées par le balayage de t1.

Utilisation

- Les propriétés spécifiées sont utilisées pour le balayage de la table.
- Les plans partiels peuvent spécifier des propriétés de balayage sans mentionner d'autres parties du plan d'exécution de la requête.
- Les plans générés incluent les propriétés de parallélisation, de prélecture et de stratégie de mise en cache pour chaque table de la requête.

Voir aussi

[lru](#), [mru](#), [parallel](#), [prefetch](#)

scan

Description

Spécifie le balayage d'une table stockée sans préciser le type de balayage.

Syntaxe

(scan *table_stockée*)

Paramètres

table_stockée

spécifie le nom de la table stockée à balayer. Il peut s'agir d'une table sous-jacente ou d'une table de travail.

Valeur renournée

Une table sous-jacente de plan abstrait générée par le balayage de la table stockée.

Exemples

Exemple 1

```
select * from t1 where c11 > 10
```

```
( scan t1 )
```

Spécifie un balayage de t1 en laissant à l'optimiseur le soin de choisir entre un balayage de table ou un balayage d'index.

Exemple 2

```
select *
      from t1, t2
     where c11 = 0
       and c22 = 1000
       and c12 = c21

      ( nl_g_join
        ( scan t2 )
        ( i_scan i_c22 t1 )
      )
```

Spécifie un plan partiel qui indique l'ordre de jointure mais qui laisse l'optimiseur choisir la méthode d'accès pour t2.

Utilisation

- L'optimiseur choisit la méthode d'accès pour la table stockée.
- L'opérateur `scan` est utilisé lorsqu'il convient de laisser à l'optimiseur le soin de choisir le type de balayage à appliquer. La méthode d'accès appliquée peut être l'une des suivantes :
 - balayage complet de table,
 - balayage d'index avec accès aux pages de données,
 - balayage avec couverture d'index, sans accès aux pages de données,
 - balayage RID utilisé pour la stratégie OR.
- Pour obtenir un exemple de plan abstrait qui spécifie la stratégie de reformatage, reportez-vous à la section [store](#)

Voir aussi

[i_scan](#), [store](#), [t_scan](#)

store

Description	Stocke les résultats d'un balayage dans une table de travail.
Syntaxe	<pre>(store <i>nom_table_travail</i> ([scan i_scan t_scan] <i>nom_table</i>))</pre>
Paramètres	<p><i>nom_table_travail</i> spécifie le nom de la table de travail à créer.</p> <p><i>nom_table</i> spécifie le nom de la table sous-jacente à balayer.</p>
Valeur retournée	Une table de travail générée par le balayage.
Exemples	<pre>select c12, max(c11) from t1 group by c12</pre> <pre>(plan (store Worktab1 (t_scan t1)) (t_scan (work_t Worktab1)))</pre>
	Spécifie le processus en deux étapes qui consiste à sélectionner l'agrégat vectoriel pour le copier dans une table de travail, puis à sélectionner les résultats contenus dans la table de travail.
Utilisation	<ul style="list-style-type: none"> • La table spécifiée est balayée et le résultat est stocké dans une table de travail. • Dans un plan abstrait, l'opérateur <code>store</code> doit être placé aux endroits suivants : <ul style="list-style-type: none"> • sous un opérateur <code>plan</code> ou <code>union</code>, où l'opérateur <code>store</code> indique une étape de prétraitement générant une table de travail ; • sous un opérateur <code>scan</code> (mais pas sous un opérateur <code>i_scan</code> ou <code>t_scan</code>). • Dans le mode de capture de plan, les tables de travail sont identifiées par <code>table_travail1</code>, <code>table_travail2</code>, etc. Pour les plans saisis manuellement, n'importe quelle convention de dénomination peut être utilisée.

- L'utilisation de la stratégie de redéfinition d'index est décrite dans un plan abstrait à l'aide de la combinaison d'opérateurs scan (store ()). Par exemple, pour la table t2 de grande taille et dépourvue d'index, le plan abstrait ci-dessous indique que t2 doit être balayée une fois, via un balayage de table, et les résultats sont à stocker dans une table de travail :

```
select *
  from t1, t2
 where c11 > 0
       and c12 = c21
       and c22 between 0 and 10000
( nl_g_join
  ( i_scan i_c11 t1)
  ( scan (store (t_scan t2 )) )
)
```

Voir aussi

[scan](#)

subq

Description	Identifie une sous-requête.
Syntaxe	(subq <i>id_sous-requête</i>)
Paramètres	<i>id_sous-requête</i> est un entier identifiant la sous-requête. Dans les plans abstraits, la numérotation des sous-requêtes est fondée sur l'ordre de la première parenthèse ouvrante des sous-requêtes figurant dans la requête.
Exemples	

Exemple 1

```
select c11 from t1
  where c12 =
        (select c21 from t2 where c22 = t1.c11)

        ( nested
          ( t_scan t1 )
          ( subq 1
            ( t_scan ( table t2 ( in ( subq 1 ) ) ) )
          )
        )
```

Sous-requête imbriquée simple.

Exemple 2

```

select c11 from t1
where c12 =
    (select c21 from t2 where c22 = t1.c11)
and c12 =
    (select c31 from t3 where c32 = t1.c11)

(
    nested
        (
            nested
                (
                    t_scan t1
                    (
                        subq 1
                            (
                                t_scan ( table t2 ( in (
                                    subq 1 ) ) ) )
                            )
                    )
                (
                    subq 2
                        (
                            t_scan ( table t3 ( in ( subq 2 ) ) ) )
                )
            )
        )
)

```

Les deux sous-requêtes sont imbriquées dans la requête principale.

Exemple 3

```

select c11 from t1
where c12 =
    (select c21 from t2 where c22 =
        (select c31 from t3 where c32 = t1.c11))

(
    nested
        (
            t_scan t1
            (
                subq 1
                    (
                        nested
                            (
                                t_scan ( table t2 ( in ( subq 1 ) ) )
                                (
                                    subq 2
                                        (
                                            t_scan ( table t3 ( in ( subq
                                                2 ) ) ) )
                                        )
                                )
                            )
            )
        )
)

```

Une sous-requête de niveau 2 imbriquée dans une sous-requête de niveau 1, elle-même imbriquée dans la requête principale.

Utilisation

- Dans une expression de plan abstrait, l'opérateur **subq** a deux significations possibles :

- Sous un opérateur **nested**, il décrit le point d'attache d'une sous-requête imbriquée à une table.
- Sous un opérateur **in**, il décrit l'imbrication des tables sous-jacentes et des vues que contient la sous-requête.

- Pour spécifier l'association d'une sous-requête sans fournir de spécification de plan, utilisez une indication vide :

```
( nested
  ( t_scan t1)
  ( subq 1
    ()
  )
)
```

- Pour fournir une description de plan abstrait concernant une sous-requête, sans indiquer l'association, spécifiez une indication vide comme table sous-jacente de plan abstrait dans l'opérateur **nested** :

```
( nested
  ()
  ( subq 1
    (t_scan ( table t1 ( in ( subq 1 ) ) ) )
  )
)
```

- Lorsqu'une sous-requête est transformée en jointure, la seule référence à celle-ci dans le plan abstrait est l'identification de la table qui y est spécifiée :

```
select *
from t2
where c21 in (select c12 from t1)
( nl_g_join
  ( t_scan t1 )
  ( t_scan ( table t2 ( in ( subq 1 ) ) ) )
```

- Lorsqu'une sous-requête est matérialisée, elle apparaît dans l'opération store, en identifiant la table à balayer durant la matérialisation :

```

select *
from t1
where c11 in (select max(c22) from t2 group by
c21)
( plan
  ( store Worktab1
    ( t_scan ( table t2 ( in ( subq 1 ) ) ) )
  )
  ( nl_g_join
    ( t_scan t1 )
    ( t_scan ( work_t Worktab1 ) )
  )
)
)

```

Voir aussi

[in, nested, table](#)

t_scan

Description	Spécifie un balayage de table d'une table stockée.
Syntaxe	(t_scan <i>table_stockée</i>)
Paramètres	<i>table_stockée</i> spécifie le nom de la table stockée à balayer.
Valeur renournée	Une table sous-jacente de plan abstrait générée par le balayage de la table stockée.
Exemples	<pre> select * from t1 (t_scan t1) </pre> <p>Effectue un balayage de table sur <i>t1</i>.</p>
Utilisation	<ul style="list-style-type: none"> L'optimiseur effectue sur la table stockée un balayage de table. La présence de <i>t_scan</i> interdit l'utilisation d'un reformatage et de la stratégie OR.
Voir aussi	i_scan, scan, store

table

Description	Identifie une table sous-jacente qui apparaît dans une sous-requête ou une vue ou à laquelle un alias est attribué dans la clause from de la requête.
Syntaxe	(table <i>nom_table</i> [<i>qualification</i>]) (table (<i>alias nom_table</i>))
Paramètres	<i>nom_table</i> est une table sous-jacente. Si la requête utilise le nom de base de données et/ou le nom du propriétaire, le plan abstrait doit également les spécifier. <i>alias</i> est l'alias, si la requête en utilise un. <i>qualification</i> est soit in (subq <i>id_sous-requête</i>), soit in (view <i>nom_vue</i>).
Exemples	Exemple 1 <pre>select * from t1 table1, t2 table2 where table1.c11 = table2.c21</pre> <pre>(nl_g_join (t_scan (table (table1 t1))) (t_scan (table (table2 t2))))</pre> Les tables t1 et t2 sont identifiées par référence aux alias utilisés dans la requête.

Exemple 2

```
select c11 from t1
where c12 =
    (select c21 from t2 where c22 = t1.c11)

    ( nested
        ( t_scan t1 )
        ( subq 1
            ( t_scan ( table t2 ( in ( subq 1 ) ) ) )
        )
    )
```

La table t2 de la sous-requête est identifiée par référence à la sous-requête.

Exemple 3

```

create view v1
as
select * from t1 where c12 > 100

select t1.c11 from t1, v1
where t1.c12 = v1.c11

( nl_g_join
  ( t_scan t1 )
  ( i_scan 2 ( table t1 ( in ( view v1 ) ) ) )
)

```

La table t1 dans la vue est identifiée par référence à la vue.

Utilisation

- Les tables sous-jacentes de plan abstrait spécifiées sont mises en correspondance avec les tables correspondantes, en fonction de la position indiquée dans la requête.
- Dans un plan abstrait, l'opérateur `table` sert à lier les noms de table à la table correspondante dans une requête SQL, et ce pour les requêtes qui contiennent des vues, des sous-requêtes et des alias de tables.
- Lorsque des alias sont utilisés, toutes les références à la table, y compris celles qui figurent dans la section des propriétés de balayage, se présentent sous la forme :

```
( table ( alias nom_table ) )
```

L'opérateur `table` est utilisé pour toutes les références à la table, y compris les propriétés de balayage de la table sous l'opérateur `props`.

Voir aussi

[in](#), [subq](#), [view](#)

union

Description	Décrit l'union de deux ou plusieurs tables sous-jacentes de plan abstrait.
Syntaxe	(union <i>table_sous-jacente1</i> ... <i>table_sous-jacenteN</i>)
Paramètres	<i>table_sous-jacente1</i> ... <i>table_sous-jacenteN</i> sont les tables sous-jacentes de plan abstrait à unir.
Valeur renournée	Une table sous-jacente de plan abstrait qui est l'union des opérandes spécifiées.
Exemples	

Exemple 1

```
select * from t1
union
select * from t2
union
select * from t3
```

```
(union
    (t_scan t1)
    (t_scan t2)
    (t_scan t3)
)
```

Renvoie l'union des trois balayages complets de tables.

Exemple 2

```
select 1,2
union
select * from t2
```

```
(union
    ( )
    (tscan t2)
)
```

Le premier côté de l'union n'étant pas une requête optimisable, le premier opérande de l'union est vide.

- Utilisation**
- Les tables sous-jacentes de plan abstrait spécifiées sont mises en correspondance avec les tables correspondantes, en fonction de la position indiquée dans la requête.
 - L'opérateur union décrit le traitement pour :
 - union, qui supprime les valeurs dupliquées,
 - union all, qui conserve les valeurs dupliquées.
 - Dans un plan abstrait, l'opérateur union doit avoir le même nombre de membres de côtés d'union que la requête SQL et l'ordre des opérandes doit correspondre à l'ordre des tables de la requête.
 - L'étape de tri et la création d'une table de travail nécessaires pour traiter des requêtes union ne sont pas représentées dans les plans abstraits.
 - Si des requêtes d'union contiennent des éléments non optimisables, un opérande vide doit être entré. Une requête select sans clause from est présentée dans l'exemple.

Voir aussi

[i_scan](#), [scan](#), [t_scan](#)

view

- Description**
- Identifie une vue qui contient la table sous-jacente à balayer.
- Syntaxe**
- `view nom_vue`
- Paramètres**
- `nom_vue`
- spécifie le nom de la vue spécifiée dans la requête. Si la requête utilise le nom de base de données et/ou le nom du propriétaire, le plan abstrait doit également les spécifier.
- Exemples**
- ```
create view v1 as
 select * from t1
```

```
 select * from v1
```

```
 (t_scan (table t1 (in (view v))))
```

Identifie la vue dans laquelle apparaît la table t1.

Utilisation

- Lorsqu'une requête inclut une vue, les tables de la vue doivent être identifiées avec `table` (*nom\_table* (`in nom_vue`)).

Voir aussi

[in](#), [table](#)

## **work\_t**

Description

Décrit une table de travail stockée.

Syntaxe

```
(work_t [nom_table_travail
 | (alias nom_table_travail)]
)
```

Paramètres

*nom\_table\_travail*  
spécifie le nom d'une table de travail.

*alias*

est l'alias spécifié pour une table de travail, le cas échéant.

Valeur renournée

Une table stockée.

Exemples

```
select c12, max(c11) from t1
 group by c12
```

```
(plan
 (store Worktab1
 (t_scan t1)
)
 (t_scan (work_t Worktab1))
)
```

Spécifie le processus en deux étapes qui consiste à sélectionner les agrégats vectoriels pour les copier dans une table de travail, puis à sélectionner les résultats contenus dans la table de travail.

Utilisation

- Met en correspondance la table stockée avec une table de travail contenue dans le plan d'exécution de requête.
- L'opérateur `store` crée une table de travail ; l'opérateur `work_t` identifie une table de travail stockée en vue d'un accès ultérieur dans le plan abstrait.

- Dans le mode de capture de plan, les tables de travail sont identifiées par *table\_travail1*, *table\_travail2*, etc. Pour les plans saisis manuellement, n’importe quelle convention de dénomination peut être utilisée.
- Si le balayage de la table de travail n’est jamais spécifié explicitement par un opérateur *scan*, il n’est pas nécessaire de nommer la table de travail et l’opérateur *work\_t* peut être omis. Le plan suivant utilise un opérateur de balayage vide « () » à la place des spécifications *t\_scan* et *work\_t* utilisées dans l’exemple.

```
(plan
 (store
 (t_scan titles)
)
 ()
)
```

- Les alias pour les tables de travail ne sont nécessaires que pour des vues matérialisées ayant fait l’objet d’une autojointure, par exemple :

```
create view v
as
select distinct c11 from t1

select *
from v v1, v v2
where ...

(plan
 (store Worktab1
 (t_scan (table t1 (in (view v))))
)
 (g_join
 (t_scan (work_t (v1 Worktab1)))
 (t_scan (work_t (v2 Worktab1)))
)
)
```

Voir aussi

[store](#), [view](#)

*work\_t*

---

# Index

## Symboles

- > (supérieur à)
  - optimisation 16
- # (signe dièse)
  - préfixe identificateur de table temporaire 309
- () (parenthèses)
  - vides, avec **i\_scan** 416
  - vides, dans les requêtes **union** 441
  - vides, pour un balayage de table de travail 443
  - vides, sous-requête 436

## A

- accès
  - Voir également* méthodes d'accès
  - méthodes de l'optimiseur 192–206
- ajout
  - groupe de plans abstraits 394
  - ajustement des ressources à l'exécution 216
    - comment éviter 227
    - définition 174
    - identification 226
    - incidence 226
  - ajustement lors de l'exécution 224–228
- alias
  - de table 438
  - pour vue 443
- allocation d'espace
  - tempdb** 314
- ALS
  - cache de journaux utilisateur 55
  - scripteur de journaux 57
  - utilisation 56
- ALS, *voir* service de journaux asynchrones 54
- analyse de requête 69–113, 115–154
- annulation
  - requêtes avec plan ajusté 226
  - annulation et restauration, tempdbs 292

## anomalie de chaînage de pages

- définition 273
- index clusterisé 281
- index non clusterisé 281
- prélecture asynchrone 273, 280
- table sans index 281

## applications de systèmes d'aide à la décision (DSS)

- requête parallèles 157, 183

## argument de recherche

- compatibilité des types de données dans 28
- équivalent 11
- exemple 19
- fermeture transitive 12
- index 17
- indexable 18
- opérateur 18
- optimisation des requêtes parallèles 196
- statistiques 20
- syntaxe 18

## association de requêtes à des plans

- groupe de plan 347
- niveau session 383

## B

### balayage d'index

- opérateur **i\_scan** 415

### balayage d'index non clusterisé en parallèle

- avec hachage 200–201
- coût d'utilisation 201
- récapitulatif 205

### balayage d'index sans correspondance

- opérateur d'inégalité 19

### balayage de partitions d'index clusterisé

- en parallèle 196–198
- conditions requises 197
- définition 196
- récapitulatif 205

- balayage de partitions d'index clusterisés en parallèle
  - coût d'utilisation 198
- balayage de partitions en parallèle 193–195
  - conditions requises 195
  - coût d'utilisation 195
  - définition 194
  - exemple 217
  - récapitulatif 205
- balayage de table
  - coût du balayage des OAM 199
  - évaluation des coûts 72
  - forcé 45
  - prélecture asynchrone 267
  - spécification 437
  - vidage du cache 71
- balayage de table avec hachage
  - balayage de table 198–200
  - E/S 193
  - index non clusterisé 200–201, 206
  - indexage 206
  - processus de travail 193
  - table sans index 206
- balayage de table de travail
  - opérateur de balayage vide 443
- balayage de table en parallèle avec hachage 198–200
  - conditions requises 200
  - coût d'utilisation 200
  - définition 198, 199
  - récapitulatif 205
- balayage de table reposant sur un hachage
  - limite avec **set scan\_parallel\_degree** 172
  - prélecture asynchrone 278
- balayage décroissant
  - interblocage 93
- balayage reposant sur des intervalles
  - E/S 193
  - processus de travail 193
- balayage reposant sur les partitions 193–195, 196–198, 206
  - prélecture asynchrone 278
- balayage, table
  - coût 72
- base de données model, extension 295
- base de données *tempdb*
  - allocation d'espace 314
  - caches de données 314
- connexion 316
- performances 307–319
- positionnement 312
- segments 313
- striping 310
- bases de données temporaires multiples. *Voir tempdbs*
- bcp** (utilitaire bulkcopy)
  - tables temporaires 310
- buffer
  - pas disponible 50
  - tri 247–248
- buffer de tri
  - calcul de la valeur maximale autorisée 250
  - condition du tri parallèle 235
  - configuration 247–248
  - instruction 247
  - option **set sort\_resources** 257
- ## C
- cache de données
  - buffer de tri 249
  - cache de sous-requête 152
  - tempdb* lié à lui-même 314, 315
  - tri parallèle 249, 252
  - vidage pendant le balayage de table 71
- cache des journaux utilisateur, dans ALS 55
- cache, données
  - balayage de table 71
  - résultat de sous-requête 152
  - tempdb* lié à lui-même 315
  - tri 247–248
  - tri parallèle 246
- capture de plans
  - niveau session 382
- caractéristiques de mise en mémoire cache et tempdbs 295
- charge de travail globale comparée à l'optimisation
  - du temps de réponse 190
- chargement de tempdbs 299
- clause **order by**
  - optimisation 88
  - optimisation parallèle 223, 233
- clause **with consumers, create index** 243

clé d'association  
 association de plans 347  
 définition 347  
**sp\_cmp\_all\_qplans** 404  
**sp\_copy\_qplan** 400

clé d'index  
 opération de mise à jour sur 103  
 option d'agencement **desc** 88–89  
 option d'ordre **desc** 88–89  
 option de tri **asc** 88–89

colonne IDENTITY  
 curseur 329

colonne longueur fixe  
 index et modes de mise à jour 113

colonne NULL  
 optimisation des mises à jour 111

commande **alter database** et tempdbs 294

commande **alter table**  
 tri parallèle 244

commande **close**,  
 mémoire 326

commande **create index**  
 clause **with consumers** 244  
 considérations sur la journalisation 255  
 espace requis 254  
 paramètre **number of sort buffers** 235, 246–252

commande de configuration 275

commande **deallocate cursor**,  
 mémoire 326

commande **declare cursor**,  
 mémoire 326

commande **dump database**,  
 tri parallèle 255

commande **open**,  
 mémoire 326

commande **select**  
 spécification d'un index 45

commande **select \***,  
 consignation 316  
 optimisation parallèle 224

commande **select into**  
 dans des requêtes parallèles 224

commande **set**  
**forceplan** 41  
 interaction de **noexec** et **statistics io** 68  
**jtc** 59

**parallel degree** 172  
**plan dump** 381  
**plan exists** 386  
**plan load** 383  
**plan replace** 383  
**scan\_parallel\_degree** 172  
**sort\_merge** 58  
**sort\_resources** 256  
**statistics io** 67  
 statistiques de cache de sous-requête 152

commande **set plan dump** 382  
 commande **set plan load** 383  
 commande **set plan replace** 383

commande **select**  
 clause **parallel** 173

comparaison de plans abstraits 401

comportement du mode compagnon normal en  
 reprise haute disponibilité et tempdbs 299

conception d'applications  
 curseur 340  
 spécification d'index 46  
 tables temporaires dans 312

configuration  
 nombre de bases de données ouvertes  
 dans tempdbs 300  
 tempdbs pour des applications 303

conflict  
 table dans le système *tempdb* 315

connexion  
 curseur 340

constante xxiii

conventions  
 dans les manuels xxi

conversion  
 listes **in** en clauses **or** 94  
 sous-requête convertie en équijointure 148  
 type de données 36

copie  
 groupe de plans 403  
 plan 400, 403  
 plan abstrait 400

coût  
 balayage d'index non clusterisé en parallèle  
 avec hachage 201  
 balayage de partitions d'index clusterisés  
 en parallèle 198

- balayage de partitions en parallèle 195  
balayage de table en parallèle avec hachage 200  
opération de tri 86  
requête à intervalles utilisant un index clusterisé 79,  
  81, 82  
requête ponctuelle 78  
coût estimé  
  clause **or** 97  
  index 4  
  jointure 25  
  matérialisation 150  
  optimisation des sous-requêtes 153  
  reformatage 141  
  traitement rapide et lent de la requête 5  
couverture par index  
  opération de tri 92  
couverture par index non clusterisé  
  coût 81  
  coût d'une requête à intervalles 81  
  opérateur d'inégalité 19  
  prélecture asynchrone 267  
CPU  
  recommandations pour les requêtes parallèles 179  
  saturation 178, 180  
  utilisation 177, 183  
création  
  groupe de plans abstraits 394  
cursor  
  d'exécution 327  
  index 328  
  lecture seule 327  
  mode 327  
  modifiable 327  
  niveau d'isolement 335  
  optimisation de la stratégie **or** 98  
  plusieurs 340  
  problème d'Halloween 329  
  procédure stockée 327  
  verrouillage 324  
cursor de mise à jour 327  
cursor en lecture seule 327  
  index 328  
  verrouillage 334
- D**
- dbcc** (Database Consistency Checker)  
  configuration de prélecture asynchrone pour 280  
  prélecture asynchrone 268  
**dbcc addtempdb** et tempdbs 302  
**dbcc pravailabletempdbs** et tempdbs 301  
degré de parallélisation 167, 207–216  
  ajustement lors de l'exécution 216, 224–228  
  au niveau requête 173  
  définition 207  
  jointure 210, 212  
  limite supérieure à 207  
  niveau serveur 168  
  niveau session 172  
  optimisation 208  
  spécification 426  
  tri parallèle 243  
dénormalisation  
  table temporaire 312  
densité  
  cellule d'intervalle 22  
  index et jointure 116, 142  
  totale 22  
densité des cellules d'intervalle 22  
densité totale 22  
device  
  RAID 180  
device de base de données  
  requêtes parallèles et 180  
device disque  
  requêtes parallèles 175, 178  
  tri parallèle 253, 255  
  vitesse E/S 179  
device RAID  
  consommation 243  
  **create index** 243  
  tables partitionnées 180  
données à virgule flottante xxiii  
données entières  
  en SQL xxiii  
  optimisation des recherches sur 16  
duplication  
  mise à jour des effets sur la performance de 106

**E**

E/S

- Voir aussi* E/S étendues
- économie avec la redéfinition d'index 141
- mise à jour directe 102
- mot-clé **prefetch** 48
- opération de mise à jour 104
- prélecture asynchrone 263, 282
- répartition sur des caches 314
- requête sur un intervalle 47
- saturation 178
- spécification de la taille dans les requêtes 47

E/S étendues

- page niveau feuille d'index 47
- prélecture asynchrone 276
- échantillonnage pour le tri parallèle 238, 257
- ensemble de pages préalablement lues 265
  - balayage séquentiel 267
  - dbcc** 268
  - index non clusterisé 268
  - lors d'une reprise 266
- équijointure
  - sous-requête convertie en 148
- équivalent en argument de recherche 11
- espace
  - conditions requises pour le tri parallèle 254

étape

- mise à jour différée 105
- mise à jour directe 102
- table de répartition des valeurs de clé 21

évaluation d'une requête

- outil 65–68
- sp\_cachestrategy** 53

exécution, curseur

- utilisation de la mémoire 327
- exportation de groupes de plans 407

expression binaire xxiii

expression caractère xxiii

expression logique xxiii

expression numérique xxiii

extension

- base de données model 295
- tempdbs 295

extraction de curseur

- mémoire 326

**F**

- FALSE**, valeur renvoyée 144
- famille de processus de travail 158
- fermeture transitive
  - jointure 13
- fermeture transitive des SARGs 12
- fermeture transitive par jointure
  - activation 13
  - définition 13
- fonction d'agrégat
  - dénormalisation et table temporaire 312
  - optimisation 99, 101
  - optimisation parallèle 223
  - sous-requêtes 149

fonction d'agrégat **count col\_name**,

optimisation 100

fonction d'agrégat **count(\*)**,

optimisation 100

fonction d'agrégat **max**,**min** utilisé avec 101

optimisation 100, 101

fonction d'agrégat **min**,**min** utilisé avec 101

optimisation 100, 101

fonction **tempdb\_id()** 292

forceplan

plan abstrait 413

fragmentation, données

chaînage de pages 273

conséquence sur la prélecture asynchrone 273

fusion des résultats d'index 239

fusion, tri parallèle 239, 247

réduction 248

**G**

groupe de buffers

spécification de la taille E/S 429

groupe de plans

ajout 394

association de plans 347

capture de plans 347

copie 403

copie sur une table 407

création 394

## *Index*

exportation 407  
informations 395  
présentation de l'utilisation 346  
rapport 395  
suppression 395  
suppression de tous les plans 407  
groupe de plans abstraits  
ajout 394  
association de plans 347  
capture de plans 347  
création 394  
exportation 407  
importation 408  
informations 395  
présentation de l'utilisation 346  
procédure de gestion 393–408  
suppression 395  
groupe, processus de travail 158  
taille 174

## **H**

haute disponibilité  
configuration pour tempdbs 297  
montage de tempdbs 299  
tempdbs 297

## **I**

ID utilisateur  
changement avec **sp\_import\_qpgroup** 408  
identificateur  
liste 409  
imbrication  
table temporaire 319  
importation groupes de plans abstraits 408  
index  
balayage et prélecture asynchrone 267  
comment éviter les tris avec 86  
création 233  
création parallèle 233  
curseur à l'aide de 328  
dynamique 98  
E/S étendues pour 47  
modes de mise à jour 112

opération de mise à jour 103, 104  
spécification pour des requêtes 45  
table temporaire 310, 319  
index clusterisé  
conditions pour la commande **create index** 242  
coût d'une requête à intervalles 79  
coût d'une requête ponctuelle 78  
espace requis 254  
prélecture 48  
prélecture asynchrone et balayage 267  
index dynamique 98  
optimisation des requêtes **or** 95  
index non clusterisé  
balayage de table avec hachage 200–201  
conditions pour la commande **create index** 243  
coût d'une requête à intervalles 81, 82  
coût d'une requête ponctuelle 78  
prélecture asynchrone 268  
requête couverte et tri 92  
tri 93

## index unique

modes de mise à jour 112  
inégalité de répartition dans les tables  
partitionnées définition 195  
impact sur les plans d'exécution des requêtes 195

## insertion

consignation dans un journal 316

## intégrité référentielle

opération de mise à jour 103  
utilisation pour des mises à jour 106

## interblocage

balayage de table 62  
balayage décroissant 93  
paramètre du seuil d'optimisation  
de concurrence 62

## intervalle

tri de partitions 239

## **J**

## jointure

boucle imbriquée 121  
compatibilité de type de données dans 36  
densité d'index 116, 142  
indexation par optimiseur 25

nombre de tables prises en compte  
par l'optimiseur 43  
opération de mise à jour 106  
optimisation 115  
optimisation de l'opérateur **union** 154  
optimisation de la clause **or** 153  
optimisation parallèle 210–213, 219–222  
ordre des tables 41  
ordre des tables en parallèle 210–213, 219–222  
plusieurs tables 118, 119  
processus 25  
tables temporaires 312  
utilisation pour des mises à jour 103, 104,  
105, 106  
jointure à boucles imbriquées 121  
spécification 424  
jointure externe 123  
ordre de jointure 121  
permutation 121  
jointure par fusion  
plan abstrait pour 421  
journal de transactions  
opération de mise à jour et 102  
journalisation  
réduction dans *tempdb* 316  
tri parallèle 255

**L**

liaison  
*tempdb* 315  
liaison d'un sa à sa propre tempdb 289  
liaison de session et tempdbs 290  
liaison de tempdbs utilisateur à un cache  
de données 295  
liaison et tempdbs 285  
liaison souple et tempdbs 290  
liaison stricte et tempdbs 290  
lignes dupliquées  
extraction des tables de travail 97  
limite  
processus de travail 167, 172  
traitement parallèle des requêtes 167, 172  
tri parallèle 167  
limite d'utilisation des ressources 228  
liste globale des bases de données temporaires 287

**M**

matériel  
recommendations concernant le traitement  
parallèle des requêtes 179  
mémoire  
curseur 324  
message  
index supprimé 46  
messages d'erreur  
ajustements lors de l'exécution 226  
*process\_limit\_action* 226  
méthode d'accès 192  
avec hachage 193  
balayage de table avec hachage 193  
balayage reposant sur des intervalles 193  
parallèle 192–206  
sélection 206  
mise à jour différée 105  
mise à jour différée de l'index 107  
mise à jour directe 102  
courte 103  
en place 102  
jointure 106  
longue 104  
mise à jour directe courte 103  
mise à jour directe longue 104, 105  
mise à jour en place 102  
mode de mise à jour  
différé 105  
direct 106  
directe courte 103, 104  
directe longue 104, 105  
en place 102  
index différé 107  
indexage 112  
jointure 106  
optimisation 111  
triggers 110  
mode **exists check** 386  
mode reprise  
scénario et tempdbs 298  
modification de données  
mode de mise à jour 102  
modification de plans abstraits 402  
mot-clé  
*all*, **union**, optimisation 154

liste 409  
mot-clé **and**,  
  sous-requêtes contenant 151  
mot-clé **any**,  
  optimisation de sous-requête 142  
mot-clé **between**,  
  optimisation 11  
mot-clé **distinct**,  
  optimisation parallèle 234  
mot-clé **exists**  
  optimisation de sous-requête 142  
  optimisation parallèle 223  
mot-clé **in**  
  optimisation 94  
  optimisation de sous-requête 142  
mot-clé **or**,  
  coût estimé 97  
  optimisation 94  
  optimisation des clauses de jointure avec 153  
  sous-requête contenant 151  
  traitement 95  
mot-clé **parallel**, commande **select** 227  
mot-clé **prefetch**  
  taille d'E/S 48  
mot-clé **shared**  
  curseur 327  
moteurs  
  nombre 177

## N

niveau d'isolement  
  curseur 335  
nom  
  clause **index** 46  
  colonne, en arguments de recherche 18  
  prélecture d'index 49  
nombre (quantité)  
  cursor rows 339  
  moteurs 177  
  table prise en compte par l'optimiseur 43  
normalisation  
  table temporaire 312  
normes SQL  
  curseur 322

number of sort buffers 248  
numéro de ligne (RID)  
  opération de mise à jour 102

## O

**OAM.** Voir table d'allocation des objets (OAM)  
occupation de CPU  
  requêtes parallèles 183  
  requêtes utilisant les CPU de manière intensive 177  
OLTP, traitement transactionnel en ligne  
  requête parallèle 191  
opérande  
  liste 409  
opérateur  
  en arguments de recherche 18  
  inégalité, en arguments de recherche 19  
opérateur (plan abstrait) **in** 417–419  
opérateur **g\_join** 411–414  
opérateur **hints** 414–415  
opérateur **i\_scan** 415  
opérateur jointure  
  **g\_join** 411  
  jointure à boucles imbriquées 424  
  jointure par fusion 420  
  **m\_g\_join** 420  
  **nl\_g\_join** 424  
opérateur **m\_g\_join** 420–421  
opérateur **nested** 422–424  
opérateur **nl\_g\_join** 424–425  
opérateur **plan** 427–429  
opérateur **prop** 430–431  
opérateur **scan** 431–432  
opérateur **store** 433–434  
  sous-requête matérialisée 437  
opérateur **subq** 434–437  
opérateur **t\_scan** 437  
opérateur **table** 438–439  
opérateur **union** 440–441  
  curseur 337  
  numérotation des caches de sous-requête 153  
  optimisation de jointures avec 154  
  optimisation parallèle 223, 234  
opérateur **view** 441

opérateur **work\_t** 442–443  
 opération de mise à jour 102  
 opération de tri (**order by**)  
*Voir aussi* ordre de tri  
 index couvrant 92  
 index non clusterisé 93  
 plan de tri 256  
 problèmes de performances 308  
 sans index 86  
 opération sur les ensembles  
   et programmation à orientation séquentielle 322  
 optimisation  
*Voir aussi* optimisation des requêtes parallèles  
 balayage des OAM 199  
 curseur 326  
 mot-clé **in** 94  
 ordre de traitement des sous-requêtes 153  
 prélecture asynchrone 274  
 requête **order by** 88  
 requête parallèle 181, 189–231  
 requête sur un intervalle 45  
 table sous-jacente SQL 8  
 techniques avancées 39–63  
 traitement des requêtes parallèles 177–184  
 tri parallèle 245–255  
 optimisation de concurrence  
   pour une petite table 62  
 optimisation des requêtes 7  
   balayage des OAM 72  
 optimisation des requêtes parallèles 189–231  
   balayages effectués sur une seule table 217–218  
   clause **exists** 223  
   clause **order by** 223  
   condition 191  
   considérations de partitions 192  
   définition 190  
   degré de parallélisation 207–216  
   détection et résolution des problèmes 229  
   exemple 216–228  
   limite d'utilisation des ressources 231  
   objectif : rapidité 190  
   opérateur **union** 223  
   optimisation en série par rapport à 190  
   ordre de jointure 210–213, 219–222  
   overhead 190, 191–192  
   requête avec agrégat 223  
 requêtes select into 224  
 sous-requêtes 222  
 table système 191  
 optimisation **like** 11  
 optimiseur 4–38, 69–113, 115–154, 189–231  
*Voir aussi* optimisation des requêtes parallèles  
 agrégat 99, 223  
 annulation 39  
 clauses **or** 94  
 diagnostic des problèmes 7, 229  
 mises à jour 111  
 ordre de jointure 210–213  
 origine des problèmes 7  
 paramètre de procédure 24  
 requête parallèle 189–231  
 sous-requête 142  
 sous-requête d'expression 148  
 sous-requête de prédictat quantifié 142  
 stratégie de reformatage 141  
 table temporaire 317  
 option **close on endtran, set** 340  
 option **cursor rows, set** 339  
 option d'**index asc** 88–89  
 option d'**index desc** 88–89  
 option de base de données **select into/bulkcopy/plisort**  
   tri parallèle 234  
 option **for update, declare cursor**  
   optimisation 338  
 option **forceplan, set** 41  
   autres possibilités 43  
   risques 42  
 option **jtc, set** 59  
 option par défaut  
   nombre de tables optimisées 44  
   statistiques sur les index 26  
 option **sort\_merge, set** 58  
 option **sort\_resources, set** 256–260  
 option **statistics subquerycache, set** 152  
 option **table count, set** 43  
 ordre  
   jointure 210–213  
   table dans une jointure 41, 119  
 ordre de jointure  
   restrictions des jointures externes 121  
 ordre de tri  
   croissant 88, 91  
   décroissant 88, 91

ordre décroissant (mot-clé **desc**) 88, 91  
requête couverte 92  
origine des problèmes de performances 7  
outil de débogage  
  **set forceplan on** 42  
overhead  
   curseur 333  
   mises à jour différées 106  
   requête parallèle 191–192

## P

page de données  
   exécution de prefetch 49  
   prélecture 49  
paramètre de configuration **abstract plan cache** 389  
paramètre de configuration **abstract plan dump** 389  
paramètre de configuration **abstract plan load** 389  
paramètre de configuration **abstract plan replace** 389  
paramètre de configuration **disk i/o structures**  
   prélecture asynchrone 272  
paramètre de configuration **max async i/os per engine**  
   prélecture asynchrone 272  
paramètre de configuration **max async i/os per server**  
   prélecture asynchrone 272  
paramètre de configuration **max parallel degree** 168,  
   213, 214  
   tri 241  
paramètre de configuration **max scan parallel degree**  
   168, 208  
paramètre de configuration **number of sort buffers**  
   message de tri parallèle 257  
   tri parallèle 235, 246–252  
paramètre de configuration **number of worker**  
   **processes** 168  
paramètre, procédure  
   optimisation 24  
parenthèses vides  
   balayage de table 443  
   dans les requêtes **union** 441  
   opérateur **i\_scan** 416  
   sous-requête 436  
partition  
   device RAID 180  
   optimisation parallèle 192

recommandations pour la configuration 182  
taux de présence dans le cache 182  
performances  
   ajustement lors de l'exécution 226  
   diagnostic des requêtes lentes 229  
   évaluation du coût des requêtes pour  
     les tables DOL 72  
   nombre de tables prises en compte par  
     l'optimiseur 44  
   **order by** 88–89  
   **tempdb** 307–319  
plan  
   changement 402  
   comparaison 401  
   copie 400, 403  
   modification 402  
   recherche 398  
   suppression 401, 407  
plan abstrait  
   affichage avec **sp\_help\_plan** 399  
   comparaison 401  
   copie 400  
   correspondance de masque 398  
   informations 399  
   recherche 398  
plan d'exécution de requête  
   ajustement lors de l'exécution 225  
   curseur modifiable 338  
   en dessous du seuil optimal 45  
   optimiseur 4  
plan de répartition 238, 257  
plan partiel  
   opérateur **hints** 414  
   spécification avec **create table** 346  
plusieurs balayages d'index avec  
   correspondance 96, 99  
prélecture  
   activation 50  
   asynchrone 263  
   désactivation 50  
   page de données 49  
   requête 48  
   **sp\_cachestrategy** 53  
prélecture asynchrone 263, 275  
   anomalie de chaînage de pages 273, 280  
   balayage de table reposant sur un hachage 278

- balayage reposant sur les partitions 278
  - balayage séquentiel 267
  - contrôle des performances 282
  - dbcc** 268, 280
  - E/S étendues 276
  - ensemble de pages préalablement lues 265
  - fragmentation 273
  - fragmentation des chaînages de pages 272
  - index non clusterisé 268
  - limite des zones 271
  - lors d'une reprise 266
  - maintenance 280
  - optimisation des objectifs 274
  - reprise 279
  - stratégie de remplacement MRU 277
  - traitement des requêtes en parallèle 278
  - prise en charge des bases de données proxy
    - et tempdbs 298
  - problème d'Halloween
    - curseur 329
  - procédure stockée
    - curseur dans 331
    - optimisation 24
    - répartition 24
    - table temporaire 320
  - procédure stockée **sp\_tempdb** 288
    - yntaxe 288
  - procédure stockée, traitement pour tempdbs 296
  - procédure système modifiée 57
  - procédure système **sp\_add\_qpgroup** 394
  - procédure système **sp\_cachestrategy** 53
  - procédure système **sp\_chgattribute**,
    - concurrency\_opt\_threshold** 62
  - procédure système **sp\_cmp\_qplans** 401
  - procédure système **sp\_copy\_all\_qplans** 403
  - procédure système **sp\_copy\_qplan** 400
  - procédure système **sp\_drop\_all\_qplans** 407
  - procédure système **sp\_drop\_qpgroup** 395
  - procédure système **sp\_drop\_qplan** 401
  - procédure système **sp\_export\_qpgroup** 407
  - procédure système **sp\_find\_qplan** 398
  - procédure système **sp\_help\_qpgroup** 395
  - procédure système **sp\_help\_qplan** 399
  - procédure système **sp\_import\_qpgroup** 408
  - procédure système **sp\_set\_qplan** 402
- procédure système **sp\_sysmon**
    - tri 261
    - tri parallèle 261
  - procédure système, modification 57
  - processus de consommation 238, 257
  - processus de coordination 158, 239
  - processus de production 238, 257
  - processus de travail 158
    - ajustement lors de l'exécution 216, 224–228
    - conditions requises pour le tri parallèle 240
    - configuration 171
    - groupe 158
    - index clusterisé 242
    - index non clusterisé 243
    - jointure 210
    - limite d'utilisation des ressources avec 231
    - overhead 191
  - processus de consommation 238
  - processus de coordination 239
  - processus de production 238
  - spécification 426
  - taille de groupe 174
  - tri de table de travail 245
  - tri parallèle 243
  - propriété de balayage
    - spécification 430
  - propriété de balayage **lru** 419–420
  - propriété de balayage **mru** 422
  - propriété de balayage **parallèle** 426
  - propriété de balayage **prefetch** 429–430
  - propriété de mise en cache
    - spécification 419, 422
  - puissance de traitement 177

## R

- rapport
  - groupe de plans 395
  - stratégie de cache 53
- recherche de plans abstraits 398
- recompilation
  - sans ajustements lors de l'exécution 228
- reformatage 141
  - jointure 141
  - optimisation parallèle 234

## *Index*

- remarques concernant l'installation et les tempdbs 303
  - répartition
    - procédures pour l'optimisation 24
    - répartition (striping) de *tempdb* 310
    - performances du tri 254
  - réPLICATION
    - opération de mise à jour 103
  - reprise
    - configuration de prélecture asynchrone pour 279
    - prélecture asynchrone 266
  - requête
    - parallèle 189–231
    - spécification d'un index pour 45
    - spécification de la taille E/S 47
  - requête couverte
    - spécification d'une stratégie de caches pour 51
  - requête lente 7
  - requête parallèle
    - table de travail 223
  - requête parallèle et tempdbs 302
  - requête sur un intervalle
    - E/S étendues pour 47
  - RÉSEAU
    - activité du curseur 333
  - restauration
  - tri parallèle 255
- 
- ## S
- SARG. *Voir* argument de recherche
  - saturation
    - CPU 178
    - E/S 178
  - sauvegarde de tempdbs 299
  - segment
    - cible 241, 257
    - performances du tri parallèle 254
    - tempdb* 313
    - tri parallèle 241
  - segment cible 241, 257
  - sélectivité d'égalité
    - résultat de **dbcc traceon(302)** 26
    - statistiques 24
  - sélectivité de l'intervalle 24
  - sélectivité de l'opérateur **between**
    - statistics 24
  - sélectivité entre intervalles 24
  - serveur de configuration
    - traitement parallèle des requêtes 168
  - services
    - reposant sur les partitions 192
  - set forceplan on**
    - plan abstrait 413
  - set plan exists check** 386
  - set, option plan dump** 381
  - set, option plan load** 383
  - set, option plan replace** 383
  - seuil d'optimisation de concurrence
    - interblocage 62
  - sous-ensemble préfixé
    - définition 21
    - exemple 21
    - order by** 92
  - sous-requête
    - any**, optimisation 142
    - association 153
    - exemple d'imbrication 434
    - exists**, optimisation 142
    - expression, optimisation 148
    - identification dans les plans 434
    - imbrication 422
    - imbrication et vues 418
    - in**, optimisation 142
    - matérialisation 148, 437
    - mise à plat 142, 436
    - mise en mémoire cache des résultats 152, 222
    - optimisation 142, 222
    - optimisation parallèle 222
    - prédicat quantifié, optimisation 142
  - sous-requête d'expression
    - optimisation 148
  - sous-requête de prédicat quantifié
    - agrégats dans 149
    - optimisation 142
  - sous-requête matérialisée 148, 437
  - sous-requête mise à plat 142, 436
  - sp\_bindcache** et tempdbs 301
  - sp\_changedbowner** et tempdbs 301
  - sp\_dboption**
    - tempdbs 300, 301
  - sp\_defaultloc** et tempdbs 301
  - sp\_dropuser** et tempdbs 301

- sp\_helpdb** et tempdbs 300
- sp\_renamedb** et tempdbs 301
- statistics
  - sélectivité **between** 24
  - utilisation du cache de sous-requête 152
- statistiques
  - sélectivité d'égalité 24
- stratégie de reformatage
  - interdiction avec **i\_scan** 416
  - interdiction avec **t\_scan** 437
  - spécification dans des plans abstraits 434
- stratégie de remplacement LRU
  - spécification 52
- stratégie de remplacement MRU
  - désactivation 53
  - prélecture asynchrone 277
  - spécification 52
- stratégie OR 95
  - cursor 337
- stratégie OR spéciale 96, 99
- suppression
  - groupe de plans abstraits 395
  - index spécifié avec **index** 46
  - mode mise à jour dans les jointures 106
  - opération de mise à jour et jointures 106
  - plan 401, 407
  - tempdbs 293
- symbole
  - dans les instructions SQL xxi
- symptôme indiquant des problèmes
  - de performances 7
- sysattributes et tempdbs 291
- sysdatabases et tempdbs 291
- redéfinition d'index 141
- requête parallèle 204, 223
- tempdb** 310
  - tri parallèle 223, 244, 247
  - verrouillage 315
- table interne de jointures 123
- table partitionnée
  - create index** 242, 243, 255
  - inégalité de répartition dans la distribution des données 195
  - optimisation parallèle 192, 206
  - table de travail 204
- table sous-jacente
  - table sous-jacente de plan abstrait 352
  - table sous-jacente SQL 352
- table sous-jacente de plan abstrait 352
  - définition 410
- table sous-jacente SQL 352
  - optimisation 8
- table système
  - tempdbs 290
- table temporaire
  - dénormalisation 312
  - indexation 319
  - normalisation 312
  - optimisation 317
  - performances 308
  - permanente 309
  - procédure imbriquée 319
- table temporaire de session 286
- table temporaire partageable 287, 304
- table temporaire privée 286
- table temporaire procédurale 286
  - suppression 286
- tâche
  - ressources CPU et 177
- tâche de maintenance
  - contrôle de **forceplan** 42
  - index forcé 46
- taille
  - base de données **tempdb** 310
- taille de groupe
  - spécification 429
- taille de tempdbs pour des applications 303
- taille E/S
  - spécification 429

## T

- table
  - normale dans **tempdb** 309
- table d'allocation des objets (OAM)
  - évaluation des coûts 72
- table de travail
  - clauses **or** 97
  - espace requis 254
  - opérateur **store** 433
  - partitionnement 204

- taux de présences dans le cache  
partitionnement et 182
- tempdbs
- alter database** 294
  - annulation et restauration 292
  - bases de données temporaires utilisateur 284
  - caractéristiques de mise en mémoire cache 295
  - chargement 299
  - comportement du mode compagnon normal
    - en reprise haute disponibilité 299
  - configuration du nombre de bases de données ouvertes 300
  - configuration haute disponibilité 297
  - dbcc addtempdb** 302
  - dbcc pavailabletempdb** 301
  - description 283
  - extension 295
  - fonction **tempdb\_id()** 292
  - haute disponibilité 297
  - liaison 285
  - liaison d'un sa à sa propre tempdb 289
  - liaison de session 290
  - liaison de tempdbs utilisateur à un cache de données 295
  - liaison explicite d'objets 284
  - liaison souple 290
  - liaison stricte 290
  - liste globale 287
  - mise à jour des procédures stockées créées par l'utilisateur 304
  - mode reprise, scénario 298
  - modification de la liaison du cache d'une base de données 296
  - modification de table système 290
  - montage pendant la reprise 299
  - prise en charge des bases de données proxy 298
  - procédure stockée, traitement 296
  - remarques concernant l'installation 303
  - requête parallèle 302
  - retour à une version antérieure d'Adaptive Server 305
  - sauvegarde 299
  - sp\_bindcache** 301
  - sp\_changedbowner** 301
  - sp\_dboption** 301
  - sp\_dboptions** 300
  - sp\_defaultloc** 301
- sp\_dropuser** 301
- sp\_helpdb** 300
- sp\_renamedb** 301
- sp\_tempdb** 288
- suppression 287, 293
- suppression explicite de tables temporaires par une session 285
- yntaxe de **sp\_tempdb** 288
- sysattributes 291
- sysdatabases 291
- table temporaire de session 286
- table temporaire partageable 287, 304
- table temporaire privée 286
- table temporaire procédurale 286
- table temporaire procédurale, suppression 286
- taille et configuration pour des applications 303
- transaction portant sur plusieurs bases de données 302
- troncature du journal 292
- variable globale `@@tempdbid` 291
- tempdbs et tempdb système 284
- tempdbs utilisateur 284
- temps de réponse
- optimisation en parallèle pour 190
- test
- index forcé 45
- traitement des requêtes
- étapes 4
  - parallèle 156–188
- traitement des requêtes parallèles 156–188, 189–231
- traitement en batch
- table temporaire 318
- traitement parallèle des requêtes
- besoin 177
  - configuration des processus de travail 171
  - configuration pour 168
  - device disque et 178
  - E/S 178
  - jointure 165
  - limites des processus de travail 169
  - phase d'exécution 160
  - prélecture asynchrone 278
  - recommandations relatives au matériel 179
  - types de fusion 161
  - types de requête 156
  - utilisation de la CPU 177, 179, 183

transaction  
     consignation dans un journal 316  
 transaction portant sur plusieurs bases  
     de données et tempdbs 302  
 tri croissant 88, 91  
 tri parallèle 233–262  
     analyse de données pour 238, 257  
     buffer de tri 247–248, 257  
     clause **with consumers** 243–244  
     commande modifiée par 233  
     condition d'exécution 234  
     conditions pour un index clusterisé 242  
     conditions pour un index non clusterisé 243  
     configuration d'un nombre suffisant de  
         buffers de tri 248  
     configuration des processus de travail 171  
     contrôle 256–260  
     degré de parallélisation 244, 257  
     exemple 258–260  
     fusion 239  
     fusion des résultats 239  
     journalisation 255  
     option **select into/bulkcopy/plisort** 234  
     option **sort\_resources** 256  
     outils d'optimisation 256  
     paramètre **number of sort buffers** 235  
     partitionnement dynamique des  
         intervalles pour 238  
     plan de répartition 238, 257  
     présentation 236  
     processus de coordination 239  
     processus de production 238  
     ressources requises 234, 240  
     restauration 255  
     segment cible 241  
     sous-index 239  
     table de travail 244, 245  
     **tempdb** 254  
     tri des intervalles 239  
 trigger  
     opération de mise à jour 103, 110  
 troncature du journal et tempdbs 292  
 TRUE, valeur renvoyée 144  
 type de données  
     correspondance dans les requêtes 28

**U**

utilisation d'ALS 56  
 utilisation du service de journaux asynchrones 54  
 utilisation du service de journaux asynchrones,  
     ALS 54

**V**

valeur  
     inconnu, optimisation 38  
 variable  
     optimiseur 24  
 variable globale `@@tempdbid` 291  
 verrou de mise à jour  
     curseur 327  
 verrou partagé  
     curseur en lecture seule 327  
 verrouillage  
     table de travail 315  
     **tempdb** et 315  
 vitesse (serveur)  
     mise à jour 102  
     mise à jour différée 105  
     mise à jour directe 102  
     mise à jour directe courte 103  
     mise à jour directe longue 104  
     mise à jour en place 102  
     opération de tri 240  
     requête lente 7  
     **select into** 316  
     suppression d'index différé 109  
 vue  
     alias 443  
     imbrication de sous-requêtes 418  
     spécification des tables 417

**Z**

zone de vidage  
     tri parallèle 253

