

**Christian Soutou**  
Avec la participation d'Olivier Teste

# **SQL** *pour* **Oracle**

**5<sup>e</sup> édition**

© Groupe Eyrolles, 2004, 2005, 2008, 2010, 2011, ISBN : 978-2-212-13329-5

**EYROLLES**



# Avant-propos

Nombre d'ouvrages traitent de SQL et d'Oracle ; certains résultent d'une traduction hasardeuse et sans vocation pédagogique, d'autres ressemblent à des annuaires téléphoniques. Les survivants, bien qu'intéressants, ne sont quant à eux plus vraiment à jour.

Ce livre a été rédigé avec une volonté de concision et de progression dans sa démarche ; il est illustré par ailleurs de nombreux exemples et figures. Bien que notre source principale d'informations fût la documentation en ligne d'Oracle, l'ouvrage ne constitue pas, à mon sens, un simple condensé de commandes SQL. Chaque notion importante est introduite par un exemple facile et démonstratif (du moins je l'espère). À la fin de chaque chapitre, des exercices vous permettront de tester vos connaissances.

La documentation d'Oracle 11g représente plus de 1 Go de fichiers HTML et PDF (soit plusieurs dizaines de milliers de pages) ! Ainsi, il est vain de vouloir expliquer tous les concepts, même si cet ouvrage ressemblait à un annuaire. J'ai tenté d'extraire les aspects fondamentaux sous la forme d'une synthèse. Ce livre résulte de mon expérience d'enseignement dans des cursus d'informatique à vocation professionnelle (IUT et master Pro).

Cet ouvrage s'adresse principalement aux novices désireux de découvrir SQL et de programmer sous Oracle.

- Les étudiants trouveront des exemples pédagogiques pour chaque concept abordé, ainsi que des exercices thématiques.
- Les développeurs C, C++, PHP ou Java découvriront des moyens de stocker leurs données.
- Les professionnels connaissant déjà Oracle seront intéressés par certaines nouvelles directives du langage.

Les fonctionnalités de la version 11g ont été prises en compte lors de la troisième édition de cet ouvrage (notamment XML DB). Certains mécanismes d'optimisation (index, *clusters*, partitionnement, tables organisées en index, vues matérialisées et dénormalisation) sont apparus lors de la quatrième édition en même temps que quelques nouveautés SQL (pivots, transpositions, requêtes *pipe line*, CTE et récursivité).

Cette cinquième édition actualise la partie Java avec la présentation de certaines fonctionnalités JDBC (connexion à une base MySQL, *Data Sources* et *RowSets*). Par ailleurs, la partie autour du langage PHP s'enrichit de l'API PDO (*PHP Data Objects*).

## Guide de lecture

---

Ce livre s'organise autour de trois parties distinctes mais complémentaires. La première intéressera le lecteur novice en la matière, car elle concerne les instructions SQL et les notions de base d'Oracle. La deuxième partie décrit la programmation avec le langage procédural d'Oracle PL/SQL. La troisième partie attirera l'attention des programmeurs qui envisagent d'utiliser Oracle tout en programmant avec des langages évolués (C, C++, PHP ou Java) ou via des interfaces Web.

### Première partie : SQL de base

Cette partie présente les différents aspects du langage SQL d'Oracle en étudiant en détail les instructions élémentaires. À partir d'exemples simples et progressifs, nous expliquons notamment comment déclarer, manipuler, faire évoluer et interroger des tables avec leurs différentes caractéristiques et éléments associés (contraintes, index, vues, séquences). Nous étudions aussi SQL dans un contexte multi-utilisateur (droits d'accès), et au niveau du dictionnaire de données.

### Deuxième partie : PL/SQL

Cette partie décrit les caractéristiques du langage procédural PL/SQL d'Oracle. Le chapitre 6 aborde des éléments de base (structure d'un programme, variables, structures de contrôle, interactions avec la base, transactions). Le chapitre 7 traite des sous-programmes, des curseurs, de la gestion des exceptions, des déclencheurs et de l'utilisation du SQL dynamique.

### Troisième partie : SQL avancé

Cette partie intéressera les programmeurs qui envisagent d'exploiter une base Oracle en utilisant un langage de troisième ou quatrième génération (C, C++ ou Java), ou en employant une interface Web. Le chapitre 8 est consacré à l'étude des mécanismes de base du précompilateur d'Oracle Pro\*C/C++. Le chapitre 9 présente les principales fonctionnalités de l'API JDBC. Le chapitre 10 décrit la technologie SQLJ (norme ISO) qui permet d'intégrer du code SQL dans un programme Java. Le chapitre 11 traite des procédures stockées et des procédures externes. Le chapitre 12 est consacré aux techniques qu'Oracle propose pour interfacer une base de données sur le Web (*PL/SQL Web Toolkit* et *PSP PL/SQL Server Pages*) ainsi que les deux principales API concernant le langage PHP : OCI8 et PDO. Le chapitre 13 présente les fonctionnalités de XML DB et l'environnement *XML DB Repository*. Enfin, le chapitre 14 est dédié à l'optimisation des requêtes et des schémas relationnels.

### Annexe : bibliographie et webographie

Vous trouverez en annexe une bibliographie consacrée à Oracle ainsi que de nombreux sites Web que j'ai jugé intéressant de mentionner ici.

## Conventions d'écriture et pictogrammes

---

La police *courrier* est utilisée pour souligner les instructions SQL, noms de types, tables, contraintes, etc. (exemple : `SELECT nom FROM Pilote`).

Les majuscules sont employées pour les directives SQL, et les minuscules pour les autres éléments. Les noms des tables, index, vues, fonctions, procédures, etc., sont précédés d'une majuscule (exemple : la table `CompagnieAerienne` contient la colonne `nomComp`).

Les termes d'Oracle (bien souvent traduits littéralement de l'anglais) sont notés en italique (exemple : *row*, *trigger*, *table*, *column*, etc.).

Dans une instruction SQL, les symboles { et } désignent une liste d'éléments, et le symbole | un choix (exemple : `CREATE { TABLE | VIEW }`). Les symboles [ et ] précisent le caractère optionnel d'une directive au sein d'une commande (exemple : `CREATE TABLE Avion (... [ORGANISATION INDEX] ;)`).



Ce pictogramme introduit une définition, un concept ou une remarque importante. Il apparaît soit dans une partie théorique, soit dans une partie technique, pour souligner des instructions importantes ou la marche à suivre avec SQL.



Ce pictogramme annonce soit une impossibilité de mise en œuvre d'un concept, soit une mise en garde. Il est principalement utilisé dans la partie consacrée à SQL.



Ce pictogramme indique une astuce ou un conseil personnel.

## Contact avec l'auteur et site

---

Si vous avez des remarques à formuler sur le contenu de cet ouvrage, n'hésitez pas à m'écrire ([soutou@iut-blagnac.fr](mailto:soutou@iut-blagnac.fr)). Vous trouverez sur le site d'accompagnement de cet ouvrage, accessible via [www.editions-eyrolles.com](http://www.editions-eyrolles.com), les errata ainsi que les codes des exemples et les corrigés de tous les exercices.

# Table des matières

---

<b>Introduction</b> .....	<b>1</b>
<b>SQL, une norme, un succès</b> .....	1
<b>Modèle de données</b> .....	2
Tables et données .....	2
Les clés .....	3
<b>Oracle</b> .....	3
Un peu d'histoire .....	4
Rachat de Sun (et de MySQL) .....	5
Offre du moment .....	6
Notion de schéma .....	8
Accès à Oracle depuis Windows .....	9
Détail d'un numéro de version .....	9
<b>Installation d'Oracle</b> .....	9
Mise en œuvre d'Oracle9i .....	10
Désinstallation de la 9i .....	12
Mise en œuvre d'Oracle 10g .....	12
Désinstallation de la 10g .....	15
Mise en œuvre d'Oracle 10g Express Edition .....	16
Mise en œuvre d'Oracle 11g .....	16
Désinstallation de la 11g .....	21
Mise en œuvre d'Oracle 11g XE Beta .....	21
<b>Les interfaces SQL*Plus</b> .....	23
Généralités .....	23
Premiers pas .....	29
Variables d'environnement .....	31
À propos des accents et jeux de caractères .....	31
<b>Partie I SQL de base</b> .....	<b>35</b>
<b>1 Définition des données</b> .....	<b>37</b>
<b>Tables relationnelles</b> .....	37
Création d'une table (CREATE TABLE) .....	37
Casse et commentaires .....	38
Premier exemple .....	39
Contraintes .....	39

Conventions recommandées . . . . .	41
Types des colonnes . . . . .	42
Structure d'une table (DESC) . . . . .	45
Restrictions . . . . .	46
Commentaires stockés (COMMENT) . . . . .	46
<b>Index</b> . . . . .	47
Classification . . . . .	48
Index B-tree . . . . .	48
Index bitmap . . . . .	48
Index basés sur des fonctions . . . . .	49
Création d'un index (CREATE INDEX) . . . . .	49
Bilan . . . . .	50
<b>Tables organisées en index</b> . . . . .	51
<b>Destruction d'un schéma</b> . . . . .	52
<b>2 Manipulation des données</b> . . . . .	<b>57</b>
<b>Insertions d'enregistrements (INSERT)</b> . . . . .	57
Syntaxe . . . . .	57
Renseigner toutes les colonnes . . . . .	58
Renseigner certaines colonnes . . . . .	58
Ne pas respecter des contraintes . . . . .	59
Dates/heures . . . . .	59
Caractères Unicode . . . . .	62
Données LOB . . . . .	62
<b>Séquences</b> . . . . .	63
Création d'une séquence (CREATE SEQUENCE) . . . . .	64
Manipulation d'une séquence . . . . .	66
Modification d'une séquence (ALTER SEQUENCE) . . . . .	67
Visualisation d'une séquence . . . . .	68
Suppression d'une séquence (DROP SEQUENCE) . . . . .	69
<b>Modifications de colonnes</b> . . . . .	69
Syntaxe (UPDATE) . . . . .	69
Modification d'une colonne . . . . .	70
Modification de plusieurs colonnes . . . . .	70
Ne pas respecter des contraintes . . . . .	70
Dates et intervalles . . . . .	71
<b>Suppressions d'enregistrements</b> . . . . .	76
Instruction DELETE . . . . .	76
Instruction TRUNCATE . . . . .	76
<b>Intégrité référentielle</b> . . . . .	77
Cohérences . . . . .	77
Contraintes côté « père » . . . . .	78
Contraintes côté « fils » . . . . .	78
Clés composites et nulles . . . . .	79

	Cohérence du fils vers le père . . . . .	80
	Cohérence du père vers le fils . . . . .	80
	En résumé . . . . .	81
	<b>Flottants</b> . . . . .	82
	Valeurs spéciales . . . . .	83
	Fonctions pour les flottants . . . . .	83
<b>3</b>	<b>Évolution d'un schéma</b> . . . . .	<b>91</b>
	<b>Renommer une table (RENAME)</b> . . . . .	91
	<b>Modifications structurelles (ALTER TABLE)</b> . . . . .	92
	Ajout de colonnes . . . . .	92
	Renommer des colonnes . . . . .	92
	Modifier le type des colonnes . . . . .	93
	Supprimer des colonnes . . . . .	94
	Colonnes UNUSED . . . . .	94
	<b>Modifications comportementales</b> . . . . .	95
	Ajout de contraintes . . . . .	95
	Suppression de contraintes . . . . .	96
	Désactivation de contraintes . . . . .	98
	Réactivation de contraintes . . . . .	100
	<b>Contraintes différées</b> . . . . .	103
	Directives DEFERRABLE et INITIALLY . . . . .	103
	Instructions SET CONSTRAINT . . . . .	105
	Instruction ALTER SESSION SET CONSTRAINTS . . . . .	105
	Directives VALIDATE et NOVALIDATE . . . . .	105
	Directive MODIFY CONSTRAINT . . . . .	107
	<b>Nouveautés 11g</b> . . . . .	108
	Colonne virtuelle . . . . .	108
	Table en lecture seule . . . . .	110
<b>4</b>	<b>Interrogation des données</b> . . . . .	<b>115</b>
	<b>Généralités</b> . . . . .	115
	Syntaxe (SELECT) . . . . .	116
	Pseudo-table DUAL . . . . .	116
	<b>Projection (éléments du SELECT)</b> . . . . .	117
	Extraction de toutes les colonnes . . . . .	118
	Extraction de certaines colonnes . . . . .	118
	Alias . . . . .	119
	Duplicatas . . . . .	119
	Expressions et valeurs nulles . . . . .	120
	Ordonnancement . . . . .	120
	Concaténation . . . . .	121
	Pseudo-colonne ROWID . . . . .	121
	Pseudo-colonne ROWNUM . . . . .	122
	Insertion multiligne . . . . .	122

<b>Restriction (WHERE)</b> .....	123
Opérateurs de comparaison .....	124
Opérateurs logiques .....	124
Opérateurs intégrés .....	125
<b>Fonctions</b> .....	126
Caractères .....	127
Numériques .....	130
Dates .....	131
Conversions .....	132
Autres fonctions .....	134
<b>Regroupements</b> .....	134
Fonctions de groupe .....	135
Étude du GROUP BY et HAVING .....	136
<b>Opérateurs ensemblistes</b> .....	139
Restrictions .....	139
Exemple .....	140
Opérateur INTERSECT .....	140
Opérateurs UNION et UNION ALL .....	141
Opérateur MINUS .....	141
Ordonner les résultats .....	142
Produit cartésien .....	143
Bilan .....	145
<b>Jointures</b> .....	145
Classification .....	146
Jointure relationnelle .....	146
Jointures SQL2 .....	146
Types de jointures .....	147
Équijointure .....	147
Autojointure .....	149
Inéquijointure .....	150
Jointures externes .....	151
Jointures procédurales .....	156
Jointures mixtes .....	160
Sous-interrogations synchronisées .....	161
Autres directives SQL2 .....	163
<b>Division</b> .....	165
Définition .....	166
Classification .....	166
Division inexacte en SQL .....	167
Division exacte en SQL .....	168
<b>Requêtes hiérarchiques</b> .....	168
Point de départ du parcours (START WITH) .....	169
Parcours de l'arbre (CONNECT BY PRIOR) .....	169
Indentation .....	170

Élagage de l'arbre (WHERE et PRIOR) . . . . .	171
Jointures . . . . .	173
Ordonnancement . . . . .	173
Nouveautés 10g . . . . .	174
<b>Mises à jour conditionnées (fusions)</b> . . . . .	178
Syntaxe (MERGE) . . . . .	178
Exemple . . . . .	179
Nouveautés 10g . . . . .	179
Exemple . . . . .	180
<b>Expressions régulières</b> . . . . .	181
Quelques exemples . . . . .	183
Fonction REGEXP_LIKE . . . . .	183
Fonction REGEXP_REPLACE . . . . .	186
Fonction REGEXP_INSTR . . . . .	187
Fonction REGEXP_SUBSTR . . . . .	189
Nouveautés 11g . . . . .	190
<b>Extractions diverses</b> . . . . .	191
Directive WITH . . . . .	191
Fonction WIDTH_BUCKET . . . . .	193
Récursivité avec WITH (CTE) . . . . .	194
Pivots (PIVOT) . . . . .	203
Transpositions (UNPIVOT) . . . . .	207
<b>5 Contrôle des données</b> . . . . .	<b>215</b>
<b>Gestion des utilisateurs</b> . . . . .	216
Classification . . . . .	216
Création d'un utilisateur (CREATE USER) . . . . .	216
Modification d'un utilisateur (ALTER USER) . . . . .	218
Suppression d'un utilisateur (DROP USER) . . . . .	219
Profils . . . . .	220
Console Enterprise Manager . . . . .	223
<b>Privilèges</b> . . . . .	228
Privilèges système . . . . .	228
Privilèges objets . . . . .	231
Privilèges prédéfinis . . . . .	235
<b>Rôles</b> . . . . .	236
Création d'un rôle (CREATE ROLE) . . . . .	237
Rôles prédéfinis . . . . .	238
Console Enterprise Manager . . . . .	239
Révocation d'un rôle . . . . .	240
Activation d'un rôle (SET ROLE) . . . . .	241
Modification d'un rôle (ALTER ROLE) . . . . .	242
Suppression d'un rôle (DROP ROLE) . . . . .	243

<b>Vues</b> .....	243
Création d'une vue (CREATE VIEW) .....	244
Classification .....	246
Vues monotables .....	246
Vues complexes .....	251
Autres utilisations de vues .....	254
Transmission de droits .....	258
Modification d'une vue (ALTER VIEW) .....	258
Suppression d'une vue (DROP VIEW) .....	258
<b>Synonymes</b> .....	259
Création d'un synonyme (CREATE SYNONYM) .....	259
Transmission de droits .....	261
Suppression d'un synonyme (DROP SYNONYM) .....	261
<b>Dictionnaire des données</b> .....	261
Constitution .....	262
Classification des vues .....	262
Démarche à suivre .....	263
Principales vues .....	265
Objets d'un schéma .....	267
Structure d'une table .....	267
Recherche des contraintes d'une table .....	268
Composition des contraintes d'une table .....	268
Détails des contraintes référentielles .....	268
Recherche du code source d'un sous-programme .....	269
Recherche des utilisateurs d'une base de données .....	270
Rôles reçus .....	270

## **Partie II PL/SQL..... 275**

### **6 Bases du PL/SQL..... 277**

<b>Généralités</b> .....	277
Environnement client-serveur .....	277
Avantages .....	278
Structure d'un programme .....	278
Portée des objets .....	279
Jeu de caractères .....	280
Identificateurs .....	280
Commentaires .....	281
<b>Variables</b> .....	281
Variables scalaires .....	282
Affectations .....	282
Restrictions .....	283

Variables %TYPE . . . . .	283
Variables %ROWTYPE . . . . .	284
Variables RECORD . . . . .	285
Variables tableaux (type TABLE) . . . . .	286
Résolution de noms . . . . .	288
Opérateurs. . . . .	288
Variables de substitution . . . . .	289
Variables de session . . . . .	290
Conventions recommandées . . . . .	290
<b>Types de données PL/SQL . . . . .</b>	<b>291</b>
Types prédéfinis . . . . .	291
Sous-types. . . . .	291
Conversions de types . . . . .	293
Nouveautés 11g . . . . .	293
<b>Structures de contrôles . . . . .</b>	<b>294</b>
Structures conditionnelles . . . . .	294
Structures répétitives . . . . .	297
Nouveautés 11g . . . . .	301
<b>Interactions avec la base . . . . .</b>	<b>302</b>
Extraire des données . . . . .	302
Manipuler des données . . . . .	304
Curseurs implicites . . . . .	306
Paquetage DBMS_OUTPUT . . . . .	307
<b>Transactions . . . . .</b>	<b>310</b>
Caractéristiques . . . . .	310
Début et fin d'une transaction . . . . .	311
Contrôle des transactions . . . . .	312
Transactions imbriquées . . . . .	313
<b>7 Programmation avancée . . . . .</b>	<b>317</b>
<b>Sous-programmes . . . . .</b>	<b>317</b>
Généralités . . . . .	317
Procédures cataloguées . . . . .	318
Fonctions cataloguées . . . . .	319
Codage d'un sous-programme PL/SQL . . . . .	320
Exemples . . . . .	320
Compilation . . . . .	323
Appels . . . . .	323
À propos des paramètres . . . . .	325
Récursivité . . . . .	326
Sous-programmes imbriqués . . . . .	326
Recompilation d'un sous-programme . . . . .	328
Destruction d'un sous-programme . . . . .	328

<b>Paquetages (packages)</b> . . . . .	328
Généralités . . . . .	328
Spécification . . . . .	329
Compilation . . . . .	330
Implémentation . . . . .	330
Appel . . . . .	331
Surcharge . . . . .	331
Recompilation . . . . .	331
Destruction d'un paquetage . . . . .	331
Comment retourner une table ? . . . . .	332
<b>Curseurs</b> . . . . .	332
Généralités . . . . .	333
Instructions . . . . .	333
Parcours d'un curseur . . . . .	334
Utilisation de structures (%ROWTYPE) . . . . .	335
Boucle FOR (gestion semi-automatique) . . . . .	336
Utilisation de tableaux (type TABLE) . . . . .	337
Utilisation de LIMIT et BULK COLLECT . . . . .	338
Paramètres d'un curseur . . . . .	339
Accès concurrents (FOR UPDATE et CURRENT OF) . . . . .	340
Variables curseurs (REF CURSOR) . . . . .	341
Fonctions table pipelined . . . . .	343
<b>Exceptions</b> . . . . .	345
Généralités . . . . .	345
Exception interne prédéfinie . . . . .	347
Exception utilisateur . . . . .	351
Utilisation du curseur implicite . . . . .	353
Exception interne non prédéfinie . . . . .	354
Propagation d'une exception . . . . .	355
Procédure RAISE_APPLICATION ERROR . . . . .	357
<b>Déclencheurs</b> . . . . .	358
À quoi sert un déclencheur ? . . . . .	358
Généralités . . . . .	359
Mécanisme général . . . . .	359
Syntaxe . . . . .	360
Déclencheurs LMD . . . . .	361
Transactions autonomes . . . . .	373
Déclencheurs LDD . . . . .	374
Déclencheurs d'instances . . . . .	374
Appels de sous-programmes . . . . .	375
Gestion des déclencheurs . . . . .	376
Ordre d'exécution . . . . .	377
Tables mutantes . . . . .	377
Nouveautés 11g . . . . .	378

<b>SQL dynamique</b> .....	382
Classification .....	383
Utilisation de EXECUTE IMMEDIATE .....	383
Utilisation d'une variable curseur .....	385
<b>Partie III SQL avancé</b> .....	<b>389</b>
<b>8 Le précompilateur Pro*C/C++</b> .....	<b>391</b>
<b>Généralités</b> .....	391
Ordres SQL intégrés .....	391
Variables .....	392
Variable indicatrice .....	393
Cas du VARCHAR .....	394
Zone de communication (SQLCA) .....	394
Connexion à une base .....	395
Gestion des exceptions .....	395
Transactions .....	396
<b>Extraction d'un enregistrement</b> .....	396
<b>Mises à jour</b> .....	398
<b>Utilisation de curseurs</b> .....	398
Variables scalaires .....	398
Variables tableaux .....	399
<b>Utilisation de Microsoft Visual C++</b> .....	401
<b>9 L'interface JDBC</b> .....	<b>403</b>
<b>Généralités</b> .....	403
Classification des pilotes (drivers) .....	404
Les paquetages .....	405
Structure d'un programme .....	406
Variables d'environnement .....	407
Test de votre configuration .....	408
<b>Connexion à une base</b> .....	408
Base Access .....	409
Base Oracle .....	410
Base MySQL .....	412
Déconnexion .....	413
Interface Connection .....	413
Sources de données .....	413
<b>États d'une connexion</b> .....	414
Interfaces disponibles .....	414
Méthodes génériques pour les paramètres .....	415
États simples (interface Statement) .....	415
Méthodes à utiliser .....	416

<b>Correspondances de types</b> .....	417
<b>Interactions avec la base</b> .....	418
Suppression de données .....	418
Ajout d'enregistrements .....	419
Modification d'enregistrements .....	419
<b>Extraction de données</b> .....	419
Curseurs statiques .....	420
Curseurs navigables .....	421
<b>Curseurs modifiables</b> .....	425
Suppressions .....	427
Modifications .....	428
Insertions .....	428
Restrictions .....	429
<b>Ensembles de lignes (RowSet)</b> .....	430
RowSet sans connexion .....	431
RowSet avec ResultSet .....	431
RowSet pour XML .....	432
Mises à jour d'un RowSet .....	433
Notifications pour un RowSet .....	433
<b>Interface ResultSetMetaData</b> .....	435
<b>Interface DatabaseMetaData</b> .....	436
<b>Instructions paramétrées (PreparedStatement)</b> .....	438
Extraction de données (executeQuery) .....	439
Mises à jour (executeUpdate) .....	439
Instruction LDD (execute) .....	440
<b>Appels de sous-programmes</b> .....	440
Appel d'une fonction .....	441
Appel d'une procédure .....	442
<b>Transactions</b> .....	443
Points de validation .....	443
<b>Traitement des exceptions</b> .....	445
Affichage des erreurs .....	445
Traitement des erreurs .....	446
<b>10 L'approche SQLI</b> .....	<b>449</b>
<b>Généralités</b> .....	449
Blocs SQLJ .....	449
Précompilation .....	450
Configurations .....	450
Affectations (SET) .....	452
<b>Intégration de SQL</b> .....	452
Instructions du LDD .....	452
Instructions du LMD .....	453

Requêtes . . . . .	453
À propos des itérateurs . . . . .	457
<b>Transactions</b> . . . . .	460
Intégration de blocs PL/SQL . . . . .	460
Points de validation . . . . .	461
<b>Appels de sous-programmes</b> . . . . .	462
Résultats scalaires . . . . .	462
Résultats complexes . . . . .	464
<b>Traitement des exceptions</b> . . . . .	465
Définition des données . . . . .	465
Manipulation des données . . . . .	466
Interrogation des données . . . . .	466
<b>Contextes de connexion</b> . . . . .	467
<b>SQL dynamique</b> . . . . .	469
Expression . . . . .	469
Restrictions . . . . .	470
<b>11 Procédures stockées et externes</b> . . . . .	<b>473</b>
<b>Procédures stockées Java</b> . . . . .	473
Stockage d'une procédure . . . . .	474
Interactions avec la base . . . . .	479
Déclencheurs . . . . .	483
<b>Procédures externes Java</b> . . . . .	484
Compilation de la classe . . . . .	485
Création d'une librairie . . . . .	485
Publication d'une procédure externe . . . . .	485
Appel d'une procédure externe . . . . .	486
<b>12 Oracle et le Web</b> . . . . .	<b>487</b>
<b>Configuration minimale d'Apache</b> . . . . .	487
<b>PL/SQL Web Toolkit</b> . . . . .	489
Détail d'une URL . . . . .	489
Paquetages HTP et HTF . . . . .	490
Pose d'hyperliens . . . . .	494
Formulaires . . . . .	496
Tables . . . . .	497
Listes . . . . .	497
<b>PL/SQL Server Pages</b> . . . . .	498
Généralités . . . . .	498
Balises . . . . .	499
Chargement d'un programme PSP . . . . .	500
Appel . . . . .	500
Interaction avec la base . . . . .	500

	<b>Intégration de PHP</b> .....	502
	Configuration adoptée .....	502
	API de PHP pour Oracle (OCI) .....	505
	Interactions avec la base .....	507
	API Objet PHP pour Oracle (PDO) .....	520
<b>13</b>	<b>Oracle XML DB</b> .....	<b>527</b>
	<b>Généralités</b> .....	527
	Comment disposer de XML DB ? .....	527
	Le type de données XMLType .....	528
	Modes de stockage .....	529
	<b>Stockages XMLType</b> .....	530
	Création d'une table .....	531
	Répertoire de travail .....	533
	Grammaire XML Schema .....	533
	Annotation de la grammaire .....	534
	Enregistrement de la grammaire .....	536
	Stockage structuré (object-relational) .....	538
	Stockage non structuré (CLOB) .....	552
	Stockage non structuré (binary XML) .....	553
	<b>Autres fonctionnalités</b> .....	557
	Génération de contenus .....	557
	Vues XMLType .....	558
	Génération de grammaires annotées .....	561
	Dictionnaire des données .....	563
	<b>XML DB Repository</b> .....	565
	Interfaces .....	565
	Configuration .....	565
	Paquetage XML_XDB .....	568
	Accès par SQL .....	568
<b>14</b>	<b>Optimisations</b> .....	<b>577</b>
	<b>Cadre général</b> .....	577
	Les acteurs .....	578
	Contexte et objectifs .....	578
	À éviter .....	579
	Présentation du jeu d'exemple .....	579
	L'offre d'Oracle 11g .....	580
	Les optimiseurs .....	581
	L'estimateur .....	583
	Traitement d'une instruction .....	584
	Configuration de l'optimiseur (les hints) .....	586

<b>Les statistiques destinées à l'optimiseur</b> . . . . .	587
Les histogrammes . . . . .	587
Collecte . . . . .	589
<b>Outils de mesure de performances</b> . . . . .	592
Visualisation des plans d'exécution . . . . .	593
L'outil tkprof . . . . .	599
Utilisation de l'événement 10046 . . . . .	604
Paquetage DBMS_APPLICATION_INFO . . . . .	605
Les vues dynamiques du dictionnaire . . . . .	609
L'utilitaire runstats de Tom Kyte . . . . .	613
Bilan . . . . .	615
<b>Organisation des données</b> . . . . .	616
Des contraintes au plus près des données . . . . .	616
Indexation . . . . .	617
Jointures . . . . .	630
Variables de lien . . . . .	638
Comment réaliser des fetchs multilignes ? . . . . .	640
<b>Clusters</b> . . . . .	641
Un mauvais et un bon exemples . . . . .	642
Création d'un cluster . . . . .	643
Cluster indexé . . . . .	644
Hash clusters . . . . .	648
Les collisions . . . . .	649
Paramétrages . . . . .	650
Cas particuliers . . . . .	650
Bilan . . . . .	653
<b>Tables organisées en index</b> . . . . .	653
Comparatif . . . . .	654
Les débordements . . . . .	655
Création d'une IOT . . . . .	655
Comparaison avec une table en heap . . . . .	656
Limites . . . . .	656
<b>Partitionnement</b> . . . . .	656
La clé de partition . . . . .	657
Partitions par intervalle . . . . .	658
Intervalles automatiques . . . . .	659
Partitions par hachage . . . . .	660
Partitions par liste . . . . .	661
Partitions par référence . . . . .	662
Sous-partitions . . . . .	663
Index partitionné . . . . .	664
Index partitionné local . . . . .	665

Index partitionné global . . . . .	666
Opérations sur les partitions et index . . . . .	667
Partitionnement des tables IOT . . . . .	667
<b>Vues matérialisées</b> . . . . .	668
Réécriture de requêtes . . . . .	669
Le rafraîchissement . . . . .	670
Exemples . . . . .	670
<b>Dénormalisation</b> . . . . .	672
Colonnes calculées . . . . .	672
Duplication de colonnes . . . . .	673
Ajout de clés étrangères . . . . .	674
Exemple de stratégie . . . . .	674
<b>Derniers conseils</b> . . . . .	674
Requêtes inefficaces . . . . .	675
Les 10 commandements de F. Brouard . . . . .	676
 <b>Annexe : Bibliographie et webographie</b> . . . . .	 <b>679</b>
 <b>Index</b> . . . . .	 <b>681</b>

# Chapitre 1

## Définition des données

Ce chapitre décrit les instructions SQL qui constituent l'aspect LDD (langage de définition des données) de SQL. À cet effet, nous verrons notamment comment déclarer une table, ses éventuels contraintes et index.

### Tables relationnelles

---

Une table est créée en SQL par l'instruction `CREATE TABLE`, modifiée au niveau de sa structure par l'instruction `ALTER TABLE` et supprimée par la commande `DROP TABLE`.

#### Création d'une table (CREATE TABLE)

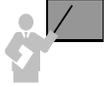
Pour pouvoir créer une table dans votre schéma, il faut que vous ayez reçu le privilège `CREATE TABLE`. Si vous avez le privilège `CREATE ANY TABLE`, vous pouvez créer des tables dans tout schéma. Le mécanisme des privilèges est décrit au chapitre « Contrôle des données ».

La syntaxe SQL simplifiée est la suivante :

```
CREATE TABLE [schéma.] nomTable
    ( colonne1 type1 [DEFAULT valeur1] [NOT NULL]
    [, colonne2 type2 [DEFAULT valeur2] [NOT NULL] ]
    [CONSTRAINT nomContrainte1 typeContrainte1]... ) ;
```

- *schéma* : s'il est omis, il sera assimilé au nom de l'utilisateur connecté. S'il est précisé, il désigne soit l'utilisateur courant soit un autre utilisateur de la base (dans ce cas, il faut que l'utilisateur courant ait le droit de créer une table dans un autre schéma). Nous aborderons ces points dans le chapitre 5 et nous considérerons jusque-là que nous travaillons dans le schéma de l'utilisateur couramment connecté (ce sera votre configuration la plupart du temps).
- *nomTable* : peut comporter des lettres majuscules ou minuscules (accentuées ou pas), des chiffres et les symboles, par exemple : `_`, `$` et `#`. Oracle est insensible à la casse et convertira au niveau du dictionnaire de données les noms de tables et de colonnes en majuscules.

- *colonnei typei* : nom d'une colonne (mêmes caractéristiques que pour les noms des tables) et son type ( NUMBER, CHAR, DATE...). Nous verrons quels types Oracle propose. La directive DEFAULT fixe une valeur par défaut. La directive NOT NULL interdit que la valeur de la colonne soit nulle.



NULL représente une valeur qu'on peut considérer comme non disponible, non affectée, inconnue ou inapplicable. Elle est différente d'un espace pour un caractère ou d'un zéro pour un nombre.

- *nomContrainte typeContrainte* : noms de la contrainte et son type (clé primaire, clé étrangère, etc.). Nous allons détailler dans le paragraphe suivant les différentes contraintes possibles.
- ; : symbole qui termine une instruction SQL d'Oracle. Le slash (/) peut également terminer une instruction à condition de le placer à la première colonne de la dernière ligne.

## Casse et commentaires

Dans toute instruction SQL (déclaration, manipulation, interrogation et contrôle des données), il est possible d'inclure des retours chariots, des tabulations, espaces et commentaires (sur une ligne précédée de deux tirets --, sur plusieurs lignes entre /\* et \*/). De même, la casse n'a pas d'importance au niveau des mots-clés de SQL, des noms de tables, colonnes, index, etc. Les scripts suivants décrivent la déclaration d'une même table en utilisant différentes conventions :

Tableau 1-1 Différentes écritures SQL

Sans commentaire	Avec commentaires
CREATE TABLE MêmesévénementsàNoël (colonne CHAR);	CREATE TABLE -- nom de la table
CREATE TABLE Test (colonne NUMBER(38,8));	<b>TEST</b> ( -- description COLONNE NUMBER(38,8) )
CREATE table test (Colonne NUMBER(38,8));	-- fin, ne pas oublier le point-virgule. ;
	CREATE TABLE Test ( /* une plus grande description des colonnes */ COLONNE NUMBER(38,8));



La casse a une incidence majeure dans les expressions de comparaison entre colonnes et valeurs, que ce soit dans une instruction SQL ou un test dans un programme. Ainsi, l'expression

« nomComp='Air France' » n'aura pas la même signification que l'expression « nomComp='AIR France' ».

Comme nous le conseillons dans l'avant-propos, il est préférable d'utiliser les conventions suivantes.



- Tous les mots-clés de SQL sont notés en MAJUSCULES.
- Les noms de tables sont notés en Minuscules (excepté la première lettre).
- Les noms de colonnes et de contraintes en minuscules.

L'adoption de ces conventions rendra vos requêtes, scripts et programmes plus lisibles (un peu à la mode Java).

## Premier exemple

Le tableau ci-dessous décrit l'instruction SQL qui permet de créer la table `Compagnie` illustrée par la figure suivante dans le schéma `soutou` (l'absence du préfixe « `soutou.` » aurait conduit au même résultat si `soutou` était l'utilisateur qui crée la table).

Figure 1-1 Table à créer

Compagnie

comp	nrue	rue	ville	nomComp

Tableau 1-2 Création d'une table et de ses contraintes

Instruction SQL	Commentaires
<pre>CREATE TABLE soutou.Compagnie (comp CHAR(4), nrue NUMBER(3), rue CHAR(20), ville CHAR(15) DEFAULT 'Paris', nomComp CHAR(15) NOT NULL);</pre>	<p>La table contient cinq colonnes (quatre chaînes de caractères et une valeur numérique de trois chiffres).                      La table inclut en plus deux contraintes :</p> <ul style="list-style-type: none"> <li>▪ <b>DEFAULT</b> qui fixe <i>Paris</i> comme valeur par défaut de la colonne <code>ville</code> ;</li> <li>▪ <b>NOT NULL</b> qui impose une valeur non nulle dans la colonne <code>nomComp</code>.</li> </ul>

## Contraintes

Les contraintes ont pour but de programmer des règles de gestion au niveau des colonnes des tables. Elles peuvent alléger un développement côté client (si on déclare qu'une note doit être comprise entre 0 et 20, les programmes de saisie n'ont plus à tester les valeurs en entrée mais seulement le code retour après connexion à la base ; on déporte les contraintes côté serveur).

Les contraintes peuvent être déclarées de deux manières :

- En même temps que la colonne (valable pour les contraintes monocolumnes), ces contraintes sont dites « en ligne » (*inline constraints*). L'exemple précédent en déclare deux.
- Une fois la colonne déclarée, ces contraintes ne sont pas limitées à une colonne et peuvent être personnalisées par un nom (*out-of-line constraints*).

Oracle recommande de déclarer les contraintes NOT NULL en ligne, les autres peuvent être déclarées soit en ligne, soit nommées. Étudions à présent les types de contraintes nommées (*out-of-line*).

Quatre types de contraintes sont possibles :

**CONSTRAINT** *nomContrainte*

- UNIQUE (*colonne1* [, *colonne2*]...)
- PRIMARY KEY (*colonne1* [, *colonne2*]...)
- FOREIGN KEY (*colonne1* [, *colonne2*]...)
  - REFERENCES [*schéma.*]*nomTablePere* (*colonne1* [, *colonne2*]...)
  - [ON DELETE { CASCADE | SET NULL }]
- CHECK (*condition*)

- La contrainte UNIQUE impose une valeur distincte au niveau de la table (les valeurs nulles font exception à moins que NOT NULL soit aussi appliquée sur les colonnes).
- La contrainte PRIMARY KEY déclare la clé primaire de la table. Un index est généré automatiquement sur la ou les colonnes concernées. Les colonnes clés primaires ne peuvent être ni nulles ni identiques (en totalité si elles sont composées de plusieurs colonnes).
- La contrainte FOREIGN KEY déclare une clé étrangère entre une table enfant (*child*) et une table père (*parent*). Ces contraintes définissent l'intégrité référentielle que nous aborderons plus tard. La directive ON DELETE dispose de deux options : CASCADE propagera la suppression de tous les enregistrements fils rattachés à l'enregistrement père supprimé, SET NULL positionnera seulement leur clé étrangère à NULL (voir la section « Intégrité référentielle » du chapitre 2).
- La contrainte CHECK impose un domaine de valeurs ou une condition simple ou complexe entre colonnes (exemple : CHECK (note BETWEEN 0 AND 20), CHECK (grade='Copilote' OR grade='Commandant')).



Il n'est pas recommandé de définir des contraintes sans les nommer (bien que cela soit possible), car il sera difficile de faire évoluer les contraintes déclarées (désactivation, réactivation, suppression) et la lisibilité des programmes en sera affectée.

Si vous ne nommez pas une contrainte, un nom est automatiquement généré sous la forme suivante : SYS\_Cnnnnnn (*n* entier).

Nous verrons au chapitre 3 comment ajouter, supprimer, désactiver, réactiver et différer des contraintes (options de la commande ALTER TABLE).

## Conventions recommandées

Adoptez les conventions d'écriture suivantes pour vos contraintes :



- Préfixez par `pk_` le nom d'une contrainte clé primaire, `fk_` une clé étrangère, `ck_` une vérification, `un_` une unicité.
- Pour une contrainte clé primaire, suffixez du nom de la table la contrainte (exemple `pk_Pilote`).
- Pour une contrainte clé étrangère, renseignez (ou abrégez) les noms de la table source, de la clé, et de la table cible (exemple `fk_Pil_compa_Comp`).

En respectant nos conventions, déclarons les tables de l'exemple suivant (Compagnie avec sa clé primaire et Pilote avec ses clés primaire et étrangère). Du fait de l'existence de la clé étrangère, la table Compagnie est dite « parent » (ou « père ») de la table Pilote « enfant » (ou « fils »). Cela résulte de l'implantation d'une association *un-à-plusieurs* entre les deux tables (bibliographie *UML 2 pour les bases de données*). Nous reviendrons sur ces principes à la section « Intégrité référentielle » du prochain chapitre.

Figure 1-2 Deux tables à créer

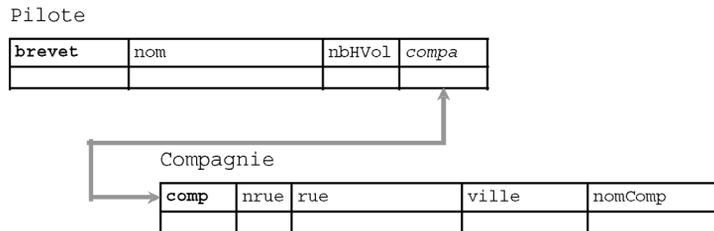
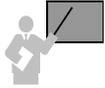


Tableau 1-3 Contraintes en ligne et nommées

Tables	Contraintes
<pre>CREATE TABLE Compagnie   (comp CHAR(4), nrue NUMBER(3),   rue CHAR(20), ville CHAR(15) DEFAULT 'Paris',   nomComp CHAR(15) NOT NULL,   CONSTRAINT pk_Compagnie PRIMARY KEY(comp));</pre>	<p>Deux contraintes en ligne et une contrainte nommée de clé primaire.</p>
<pre>CREATE TABLE Pilote   (brevet CHAR(6), nom CHAR(15),   nbHVol NUMBER(7,2), compa CHAR(4),   CONSTRAINT pk_Pilote PRIMARY KEY(brevet),   CONSTRAINT nn_nom CHECK (nom IS NOT NULL),   CONSTRAINT ck_nbHVol CHECK     (nbHVol BETWEEN 0 AND 20000),   CONSTRAINT un_nom UNIQUE (nom),   CONSTRAINT fk_Pil_compa_Comp FOREIGN KEY     (compa) REFERENCES Compagnie(comp));</pre>	<p>Aucune contrainte en ligne et cinq contraintes nommées :</p> <ul style="list-style-type: none"> <li>• Clé primaire</li> <li>• NOT NULL</li> <li>• CHECK (nombre d'heures de vol compris entre 0 et 20000)</li> <li>• UNIQUE (homonymes interdits)</li> <li>• Clé étrangère</li> </ul>

## Remarques



- L'ordre n'est pas important dans la déclaration des contraintes nommées.
- Une contrainte NOT NULL doit être déclarée dans un CHECK si elle est nommée.
- PRIMARY KEY équivaut à : UNIQUE + NOT NULL + index.
- L'ordre de création des tables est important quand on définit les contraintes en même temps que les tables (on peut différer la création ou l'activation des contraintes, voir le chapitre 3). Il faut créer d'abord les tables « pères » puis les tables « fils ». Le script de destruction des tables suit le raisonnement inverse.

## Types des colonnes

Pour décrire les colonnes d'une table, Oracle fournit les types prédéfinis suivants (*built-in datatypes*) :

- caractères (CHAR, NCHAR, VARCHAR2, NVARCHAR2, CLOB, NCLOB, LONG) ;
- valeurs numériques NUMBER ;
- date/heure (DATE, INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE) ;
- données binaires (BLOB, BFILE, RAW, LONG RAW) ;
- adressage des enregistrements ROWID.

Détaillons à présent ces types. Nous verrons comment utiliser les plus courants au chapitre 2 et les autres au fil de l'ouvrage.

### Caractères

Les types CHAR et NCHAR permettent de stocker des chaînes de caractères de taille fixe.

Les types VARCHAR2 et NVARCHAR2 permettent de stocker des chaînes de caractères de taille variable (VARCHAR est maintenant remplacé par VARCHAR2).

Les types NCHAR et NVARCHAR2 permettent de stocker des chaînes de caractères Unicode (*multibyte*), méthode de codage universelle qui fournit une valeur de code unique pour chaque caractère quels que soient la plate-forme, le programme ou la langue. Unicode est utilisé par XML, Java, JavaScript, LDAP, et WML. Ces types Oracle sont proposés dans le cadre NLS (*National Language Support*).

Les types CLOB et NCLOB permettent de stocker des flots de caractères (exemple : du texte).

Tableau 1-4 Types de données caractères

Type	Description	Commentaire pour une colonne
CHAR ( <i>n</i> [BYTE   CHAR])	Chaîne fixe de <i>n</i> caractères ou octets.	Taille fixe (complétée par des blancs si nécessaire). Maximum de 2 000 octets ou caractères.
VARCHAR2 ( <i>n</i> [BYTE   CHAR])	Chaîne variable de <i>n</i> caractères ou octets.	Taille variable. Maximum de 4 000 octets ou caractères.
NCHAR ( <i>n</i> )	Chaîne fixe de <i>n</i> caractères Unicode.	Taille fixe (complétée par des blancs si nécessaire). Taille double pour le jeu AL16UTF16 et triple pour le jeu UTF8. Maximum de 2 000 caractères.
NVARCHAR2 ( <i>n</i> )	Chaîne variable de <i>n</i> caractères Unicode.	Taille variable. Mêmes caractéristiques que NCHAR sauf pour la taille maximale qui est ici de 4 000 octets.
CLOB	Flot de caractères (CHAR).	Jusqu'à 4 gigaoctets.
NCLOB	Flot de caractères Unicode (NCHAR).	Idem CLOB.
LONG	Flot variable de caractères.	Jusqu'à 2 gigaoctets. Plus utilisé mais fourni pour assurer la compatibilité avec les anciennes applications.

### Valeurs numériques

Le type NUMBER sert à stocker des entiers positifs ou négatifs, des réels à virgule fixe ou flottante. La plage de valeurs possibles va de  $\pm 1 \times 10^{-130}$  à  $\pm 9.99 \times 10^{125}$  (trente-huit 9 suivis de quatre-vingt-huit 0).

Tableau 1-5 Type de données numériques

Type	Description	Commentaires pour une colonne
NUMBER ( <i>t</i> , <i>d</i> )	Flottant de <i>t</i> chiffres dont <i>d</i> décimales.	Maximum pour <i>t</i> : 38. Plage pour <i>d</i> : [- 84, + 127]. Espace maximum utilisé : 21 octets.

Lorsque la valeur de *d* est négative, l'arrondi se réalise à gauche de la décimale comme le montre le tableau suivant.

Tableau 1-6 Représentation du nombre 7456123.89

Type	Description
NUMBER	7456123.89
NUMBER (9)	7456124
NUMBER (9, 2)	7456123.89
NUMBER (9, 1)	7456123.9
NUMBER (6)	Précision inférieure à la taille du nombre.
NUMBER (7, -2)	7456100
NUMBER (-7, 2)	Précision inférieure à la taille du nombre.

## Date/heure

- Le type `DATE` permet de stocker des moments ponctuels, la précision est composée du siècle, de l'année, du mois, du jour, de l'heure, des minutes et des secondes.
- Le type `TIMESTAMP` est plus précis dans la définition d'un moment (fraction de seconde).
- Le type `TIMESTAMP WITH TIME ZONE` prend en compte les fuseaux horaires.
- Le type `TIMESTAMP WITH LOCAL TIME ZONE` permet de faire la dichotomie entre une heure côté serveur et une heure côté client.
- Le type `INTERVAL YEAR TO MONTH` permet d'extraire une différence entre deux moments avec une précision mois/année.
- Le type `INTERVAL DAY TO SECOND` permet d'extraire une différence plus précise entre deux moments (précision de l'ordre de la fraction de seconde).

**Tableau 1-7 Types de données date/heure**

Type	Description	Commentaires pour une colonne
<code>DATE</code>	Date et heure du 1 <sup>er</sup> janvier 4712 avant J.-C. au 31 décembre 4712 après J.-C.	Sur 7 octets. Le format par défaut est spécifié par le paramètre <code>NLS_DATE_FORMAT</code> .
<code>INTERVAL YEAR (an) TO MONTH</code>	Période représentée en années et mois.	Sur 5 octets. La précision de <i>an</i> va de 0 à 9 (par défaut 2).
<code>INTERVAL DAY (jo) TO SECOND (fsec)</code>	Période représentée en jours, heures, minutes et secondes.	Sur 11 octets. Les précisions <i>jo</i> et <i>fsec</i> vont de 0 à 9 (par défaut 2 pour le jour et 6 pour les fractions de secondes).
<code>TIMESTAMP (fsec)</code>	Date et heure incluant des fractions de secondes (précision qui dépend du système d'exploitation).	De 7 à 11 octets. La valeur par défaut du paramètre d'initialisation est située dans <code>NLS_TIMESTAMP_FORMAT</code> . La précision des fractions de secondes va de 0 à 9 (par défaut 6).
<code>TIMESTAMP (fsec) WITH TIME ZONE</code>	Date et heure avec le décalage de Greenwich (UTC) au format ' <i>h:m</i> ' ( <i>heures:minutes</i> par rapport au méridien, exemple : '-5:0').	Sur 13 octets. La valeur par défaut du paramètre de l'heure du serveur est située dans <code>NLS_TIMESTAMP_TZ_FORMAT</code> .
<code>TIMESTAMP (fsec) WITH LOCAL TIME ZONE</code>	Comme le précédent mais cadré sur l'heure locale (client) qui peut être différente de celle du serveur.	De 7 à 11 octets.

### Données binaires

Les types `BLOB` et `BFILE` permettent de stocker des données non structurées (structure opaque pour Oracle) comme le multimédia (images, sons, vidéo, etc.).

**Tableau 1-8** Types de données binaires

Type	Description	Commentaires pour une colonne
<code>BLOB</code>	Données binaires non structurées.	Jusqu'à 4 gigaoctets.
<code>BFILE</code>	Données binaires stockées dans un fichier externe à la base.	idem.
<code>RAW(size)</code>	Données binaires.	Jusqu'à 2 000 octets. Plus utilisé mais fourni pour assurer la compatibilité avec les anciennes applications.
<code>LONG RAW</code>	Données binaires.	Comme <code>RAW</code> , jusqu'à 2 gigaoctets.

### Structure d'une table (DESC)

`DESC` (raccourci de `DESCRIBE`) est une commande SQL\*Plus, car elle n'est comprise que dans l'interface de commandes d'Oracle. Elle permet d'extraire la structure brute d'une table. Elle peut aussi s'appliquer à une vue ou un synonyme. Enfin, elle révèle également les paramètres d'une fonction ou procédure cataloguée.

**DESC** [`schéma.`] *élément*

Si le schéma n'est pas indiqué, il s'agit de celui de l'utilisateur connecté. L'élément désigne le nom d'une table, vue, procédure, fonction ou synonyme.

Retrouvons la structure des tables `Compagnie` et `Pilote` précédemment créées. Le type de chaque colonne apparaît :

**Tableau 1-9** Structure brute des tables

Table Compagnie			Table Pilote		
SQL> DESC Compagnie			SQL> DESC Pilote		
Nom	NULL ?	Type	Nom	NULL ?	Type
-----			-----		
COMP	NOT NULL	CHAR(4)	BREVET	NOT NULL	CHAR(6)
NRUE		NUMBER(3)	NOM		CHAR(15)
RUE		CHAR(20)	NBHVOL		NUMBER(7,2)
VILLE		CHAR(15)	COMPA		CHAR(4)
NOMCOMP	<b>NOT NULL</b>	CHAR(15)			

Les contraintes NOT NULL nommées (définies via les contraintes CHECK) n'apparaissent pas mais sont pourtant actives (c'est le cas de la contrainte nn\_nom sur la colonne nom). Les colonnes clés primaires sont toujours définies en ligne NOT NULL.

## Restrictions



La commande DESC n'affiche que les contraintes NOT NULL définies en ligne au niveau des colonnes (en gras dans le script).

Les noms des tables et contraintes ne doivent pas dépasser 30 caractères. Ces noms doivent être uniques dans le schéma (restriction valable pour les vues, index, séquences, synonymes, fonctions, etc.).

Les noms des colonnes doivent être uniques pour une table donnée (il est en revanche possible d'utiliser le même nom de colonne dans plusieurs tables).

Les noms des objets (tables, colonnes, contraintes, vues, etc.) ne doivent pas emprunter des mots-clés du SQL d'Oracle TABLE, SELECT, INSERT, IF... Si vous êtes francophone, cela ne vous gênera pas.

## Commentaires stockés (COMMENT)

Les commentaires stockés permettent de documenter une table, une colonne ou une vue. L'instruction SQL pour créer un commentaire est COMMENT.

```
COMMENT ON { TABLE [schéma.]nomTable |
COLUMN [schéma.]nomTable.nomColonne }
IS 'Texte décrivant le commentaire';
```

Pour supprimer un commentaire, il suffit de le redéfinir en inscrivant une chaîne vide (' ') dans la clause IS. Une fois définis, nous verrons à la section « Dictionnaire des données » du chapitre 5 comment retrouver ces commentaires.

Le premier commentaire du script ci-après documente la table Compagnie, les trois suivants renseignent trois colonnes de cette table. La dernière instruction supprime le commentaire à propos de la colonne nomComp.

```
COMMENT ON TABLE Compagnie IS 'Table des compagnies aériennes
françaises';
COMMENT ON COLUMN Compagnie.comp IS 'Code abréviation de la
compagnie';
COMMENT ON COLUMN Compagnie.nomComp IS 'Un mauvais commentaire';
COMMENT ON COLUMN Compagnie.ville IS 'Ville de la compagnie,
défaut : Paris';
COMMENT ON COLUMN Compagnie.nomComp IS '';
```

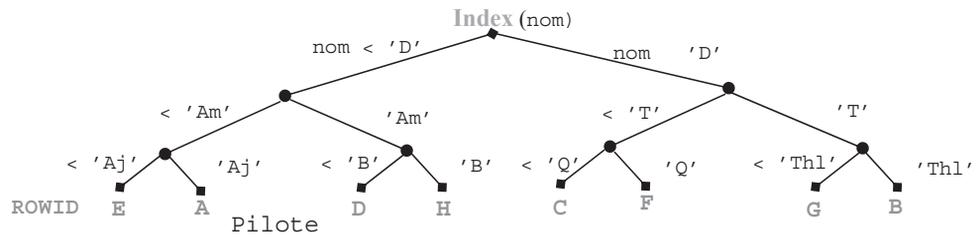
# Index

Comme l'index de cet ouvrage vous aide à atteindre les pages concernées par un mot recherché, un index Oracle permet d'accélérer l'accès aux données d'une table. Le but principal d'un index est d'éviter de parcourir une table séquentiellement du premier enregistrement jusqu'à celui visé (problème rencontré si c'est le Français nommé « Zidane » qu'on recherche dans une table non indexée de plus de soixante millions d'enregistrements...). Le principe d'un index est l'association de l'adresse de chaque enregistrement (ROWID) avec la valeur des colonnes indexées.

Sans index et pour  $n$  enregistrements le nombre moyen d'accès nécessaire pour trouver un élément est égal à  $n/2$ . Avec un index, ce nombre tendra vers  $\log(n)$  et augmentera donc bien plus faiblement en fonction de la montée en charge des enregistrements.

La figure suivante illustre un index sous la forme d'un arbre. Cet index est basé sur la colonne nom de la table Pilote. Cette figure est caricaturale, car un index n'est pas un arbre binaire (plus de deux liens peuvent partir d'un nœud). Dans cet exemple, trois accès à l'index seront nécessaires pour adresser directement un pilote via son nom au lieu d'en analyser huit au plus.

Figure 1-3 Index sur la colonne nom



ROWID	brevet	nom	nbHVol	compa
A	PL-1	Amélie Sulpice	450	AF
B	PL-2	Thomas Sulpice	900	AF
C	PL-3	Paul Soutou	1000	SING
D	PL-4	Aurélia Ente	850	ALIB
E	PL-5	Agnès Bidal	500	SING
F	PL-6	Sylvie Payrissat	2500	SING
G	PL-7	Thierry Guibert	600	ALIB
H	PL-8	Cathy Castaings	400	AF

Un index est associé à une table et peut être défini sur une ou plusieurs colonnes (dites « indexées »). Une table peut « héberger » plusieurs index. Ils sont mis à jour automatiquement après rafraîchissement de la table (ajouts et suppressions d'enregistrements ou modification des colonnes indexées). Un index peut être déclaré unique si on sait que les valeurs des colonnes indexées seront toujours uniques.

## Classification

Plusieurs types d'index sont proposés par Oracle :

- l'arbre équilibré (*B-tree*), le plus connu, qui peut être défini sur trente-deux colonnes ;
- inverse (*reverse key*) qui concerne les tables « clusterisées » ;
- chaîne de bits (*bitmap*) qui regroupe chaque valeur de la (ou des) colonne(s) indexée(s) sous la forme d'une chaîne de bits. Ce type d'index peut être défini sur trente colonnes. Option disponible seulement avec la version *Enterprise Edition* ;
- basés sur des calculs entre colonnes (*function-based indexes*).

## Index B-tree

La particularité de ce type d'index est qu'il conserve en permanence une arborescence symétrique (*balanced*). Toutes les feuilles sont à la même profondeur. Le temps de recherche est ainsi à peu près constant quel que soit l'enregistrement cherché. Le plus bas niveau de l'index (*leaf blocks*) contient les valeurs des colonnes indexées et le *rowid*. Toutes les feuilles de l'index sont chaînées entre elles. Pour les index non uniques (par exemple si on voulait définir un index sur la colonne `compa` de la table `Pilote`) le *rowid* est inclus dans la valeur de la colonne indexée. Ces index, premiers apparus, sont désormais très fiables et performants, ils ne se dégradent pas lors de la montée en charge de la table.

## Index bitmap

Alors qu'un index *B-tree*, permet de stocker une liste de *rowids* pour chaque valeur de la colonne indexée, un *bitmap* ne stocke qu'une chaîne de bits. Chacun d'eux correspond à une possible valeur de la colonne indexée. Si le bit est positionné à 1, pour une valeur donnée de l'index, cela signifie que la ligne courante contient la valeur. Une fonction de transformation convertit la position du bit en un *rowid*. Si le nombre de valeurs de la colonne indexée est faible, l'index *bitmap* sera très peu gourmand en occupation de l'espace disque.

Cette technique d'indexage est intéressante dans les applications décisionnelles (*On Line Analytical Processing*) qui manipulent de grandes quantités de données mais ne mettent pas en jeu un grand nombre de transactions. Pour les applications transactionnelles (*On Line Transaction Processing*), les index *B-tree* conviennent mieux.

La figure suivante présente un index *bitmap* basé sur la colonne `compa`. Chaque ligne est associée à une chaîne de bits de taille variable (égale au nombre de valeurs de la colonne indexée, ici trois compagnies sont recensées dans la table `Pilote`).

Les index *bitmaps* sont très bien adaptés à la recherche d'informations basée sur des critères d'égalité (exemple : `compa = 'AF'`), mais ne conviennent pas du tout à des critères de comparaison (exemple : `nbHVol > 657`).

Figure 1-4 Index bitmap sur la colonne compa

Index bitmap sur compa				Pilote				
ROWID	AF	SING	ALIB	ROWID	brevet	nom	nbHVol	compa
A	1	0	0	A	PL-1	Amélie Sulpice	450	AF
B	1	0	0	B	PL-2	Thomas Sulpice	900	AF
C	0	1	0	C	PL-3	Paul Soutou	1000	SING
D	0	0	1	D	PL-4	Aurélia Ente	850	ALIB
E	0	1	0	E	PL-5	Agnès Bidal	500	SING
F	0	1	0	F	PL-6	Sylvie Payrissat	2500	SING
G	0	0	1	G	PL-7	Thierry Guibert	600	ALIB
H	1	0	0	H	PL-8	Cathy Castaings	400	AF

## Index basés sur des fonctions

Une fonction de calcul (expressions arithmétiques ou fonctions SQL, PL/SQL ou C) peut définir un index. Celui-ci est dit « basé sur une fonction » (*function based index*).

Dans le cas des fonctions SQL (étudiées au chapitre 4), il ne doit pas s’agir de fonctions de regroupement (SUM, COUNT, MAX, etc.). Ces index servent à accélérer les requêtes contenant un calcul pénalisant s’il est effectué sur de gros volumes de données.

Dans l’exemple suivant, on accède beaucoup aux comptes bancaires sur la base du calcul bien connu de ceux qui sont souvent en rouge :  $(credit-debit)*(1+(txInt/100))-agios$ .

Figure 1-5 Index basé sur une fonction

ROWID	Index fonction $(credit-debit)*(1+(txInt/100))-agios$	CompteEpargne						
ROWID	ncompte	titulaire	debit	credit	txInt	agios		
C	-7,29	A	C1	Guibert	560	1000	3.6	5.7
D	228,67	B	C2	Soutou	250	850	4.1	50.5
A	450,14	C	C3	Teste	40	45	4.2	12.5
B	574,1	D	C4	Albaric	670	900	3.9	10.3

Un index basé sur une fonction peut être de type *B-tree* ou *bitmap*.



Il n’est pas possible de définir un tel index sur une colonne LOB, REF, ou collection (*nested table* et *varray*). Un index *bitmap* ne peut pas être unique.

## Création d’un index (CREATE INDEX)

Pour pouvoir créer un index dans son schéma, la table à indexer doit appartenir au schéma. Si l’utilisateur a le privilège INDEX sur une table d’un autre schéma, il peut en créer un dans un autre schéma. Si l’utilisateur a le privilège CREATE ANY INDEX, il peut en constituer un dans tout schéma.

Un index est créé par l'instruction `CREATE INDEX`, modifié par la commande `ALTER INDEX` et supprimé par `DROP INDEX`.

En ce qui concerne les index basés sur des fonctions, l'utilisateur doit avoir le privilège `QUERY REWRITE`. La syntaxe de création d'un index est la suivante :

#### CREATE INDEX

```
{ UNIQUE | BITMAP } [schéma.]nomIndex
ON [schéma.]nomTable ( {colonne1 | expressionColonne1 } [ASC |
DESC ] ... ) ;
```

- `UNIQUE` permet de créer un index qui ne supporte pas les doublons.
- `BITMAP` fabrique un index « chaîne de bits ».
- `ASC` et `DESC` précisent l'ordre (croissant ou décroissant).

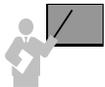
Créons plusieurs index sur la table des comptes bancaires. Le dernier (basé sur une fonction), doit faire apparaître les colonnes calculées dans ses paramètres après l'expression du calcul.

```
CREATE TABLE CompteEpargne
(ncompte CHAR(4), titulaire VARCHAR(30), debit NUMBER(10,2),
credit NUMBER(10,2), txInt NUMBER(2,1), agios NUMBER(5,2));
```

Tableau 1-10 Création d'index

Instruction SQL	Commentaires
<pre>CREATE UNIQUE INDEX idx_titulaire_CompteEpargne ON CompteEpargne (titulaire DESC);</pre>	Index <i>B-tree</i> , ordre inverse.
<pre>CREATE INDEX idx_debitenFF_CompteEpargne ON CompteEpargne (debit*6.56);</pre>	Index <i>B-tree</i> , expression d'une colonne.
<pre>CREATE BITMAP INDEX idx_bitmap_txInt_CompteEpargne ON CompteEpargne (txInt);</pre>	Index <i>bitmap</i> .
<pre>CREATE INDEX idx_fct_Solde_CompteEpargne ON CompteEpargne ((credit-debit)*(1+(txInt/100))-agios, credit, debit, txInt, agios);</pre>	Index basé sur une fonction.

## Bilan



- Un index ralentit les rafraîchissements de la base (conséquence de la mise à jour de l'arbre ou des *bitmaps*). En revanche il accélère les accès.
- Il est conseillé de créer des index sur des colonnes (majoritairement des clés étrangères) utilisées dans les clauses de jointures (voir chapitre 4).

- Les index *bitmaps* sont conseillés quand il y a peu de valeurs distinctes de la (ou des) colonne(s) à indexer. Dans le cas inverse, utilisez un index *B-tree*.
- Les index sont pénalisants lorsqu'ils sont définis sur des colonnes très souvent modifiées ou si la table contient peu de lignes.

## Tables organisées en index

Une table organisée en index (*index-organized table*) peut être considérée comme la fusion d'une table et d'un index *B-tree*. Contrairement aux tables ordinaires (*heap-organized*) qui stockent des données sans ordre, toutes les valeurs d'une table organisée en index sont stockées au sein d'un index *B-tree*.

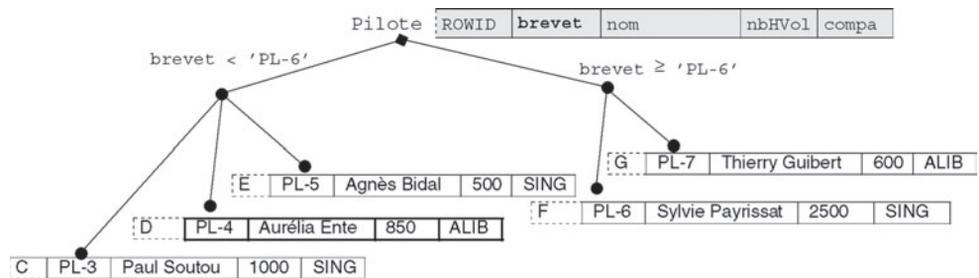
Apparu en version 8, ce type de tables est particulièrement utile pour les applications qui doivent extraire des informations basées essentiellement sur les clés primaires ou des éléments plus complexes (textes, images et sons). Le tableau suivant détaille les différences entre les deux types de tables.

Tableau 1-11 Caractéristiques des tables

Tables ordinaires	Tables organisées en index
La pseudo-colonne ROWID identifie chaque enregistrement. La clé primaire est optionnelle.	La clé primaire est obligatoire pour identifier chaque enregistrement.
Le ROWID physique permet de construire des index secondaires.	Le ROWID logique permet de construire des index secondaires.
Utilisation de clusters possible.	Utilisation interdite de clusters.
Peut contenir une colonne de type LONG et plusieurs colonnes de type LOB.	Peut contenir plusieurs colonnes LOB mais aucune de type LONG.

La figure suivante illustre la table *Pilote* organisée en index basé sur la clé primaire *brevet*.

Figure 1-6 Table organisée en index



La création d'une table organisée en index nécessite l'utilisation de la directive `ORGANIZATION INDEX` dans l'instruction `CREATE TABLE`. La clé primaire doit être obligatoirement déclarée. Des paramètres d'optimisation (`OVERFLOW` et `PCTTHRESHOLD`) peuvent également être mis en œuvre.

Dans notre exemple la syntaxe à utiliser est la suivante :

```
CREATE TABLE Pilote
  (brevet CHAR(6), nom CHAR(15), nbHVol NUMBER(7,2), compa CHAR(4),
  CONSTRAINT pk_Pilote PRIMARY KEY(brevet))
  ORGANIZATION INDEX ;
```

Les autres options de la directive `ORGANIZATION` sont :

- `HEAP` qui indique que les données ne sont pas stockées dans un ordre particulier (option par défaut) ;
- `EXTERNAL` qui précise que la table est en lecture seule et est située à l'extérieur de la base (sous la forme d'un fichier ASCII par exemple).

## Destruction d'un schéma

---

- Il vous sera utile d'écrire un script de destruction d'un schéma (ici j'entends « schéma » comme ensemble de tables, contraintes et index composant une base de données et non pas en tant qu'ensemble de tous les objets d'un utilisateur) pour pouvoir recréer une base propre. Bien entendu si des données sont déjà présentes dans les tables, et que vous souhaitez les garder, il faudra utiliser une stratégie pour les réimporter dans les nouvelles tables. À ce niveau de l'ouvrage, vous n'en êtes pas là et le script de destruction va vous permettre de corriger vos erreurs de syntaxe du script de création des tables.
- Nous avons vu qu'il fallait créer d'abord les tables « pères » puis les tables « fils » (si des contraintes sont définies en même temps que les tables). L'ordre de destruction des tables, pour des raisons de cohérence, est inverse (il faut détruire les tables « fils » puis les tables « pères »). Dans notre exemple, il serait malvenu de supprimer la table `Compagnie` avant la table `Pilote`. En effet la clé étrangère `compa` n'aurait plus de sens.

Pour pouvoir supprimer une table dans son schéma, il faut que la table appartienne à l'utilisateur. Si l'utilisateur a le privilège `DROP ANY TABLE`, il peut supprimer une table dans tout schéma.

L'instruction `DROP TABLE` entraîne la suppression des données, de la structure, de la description dans le dictionnaire des données, des index, des déclencheurs associés (*triggers*) et la récupération de la place dans l'espace de stockage.

```
DROP TABLE [schéma.] nomTable [CASCADE CONSTRAINTS] ;
```

- CASCADE CONSTRAINTS permet de s'affranchir des clés étrangères actives contenues dans d'autres tables et qui référencent la table à supprimer. Cette option détruit les contraintes des tables « fils » associées sans rien modifier aux données qui y sont stockées (voir *Intégrité référentielle* du prochain chapitre).

Les éléments qui utilisaient la table (vues, synonymes, fonctions ou procédures) ne sont pas supprimés mais sont temporairement inopérants. Attention, une suppression ne peut pas être par la suite annulée.



Il suffit de relire à l'envers le script de création de vos tables pour en déduire l'ordre de suppression à écrire dans le script de destruction de votre schéma.

Attention à l'utilisation de CASCADE CONSTRAINTS (effets de bord).

Le tableau suivant présente deux écritures possibles pour détruire des schémas.

**Tableau 1-12 Scripts équivalents de destruction**

Avec CASCADE CONSTRAINTS	Les « fils » puis les « pères »
<pre>--schéma Compagnie DROP TABLE Compagnie CASCADE CONSTRAINTS; DROP TABLE Pilote;</pre>	<pre>--schéma Compagnie DROP TABLE Pilote; DROP TABLE Compagnie;</pre>
<pre>--schéma Banque DROP INDEX idx_fct_Solde_CompteEpargne; DROP INDEX idx_bitmap_txInt_CompteEpargne; DROP INDEX idx_debitenFF_CompteEpargne; DROP INDEX idx_titulaire_CompteEpargne; DROP TABLE CompteEpargne; /* Aurait aussi supprimé les index */</pre>	

## Exercices

---

L'objectif de ces exercices est de créer des tables, leur clé primaire et des contraintes de vérification (NOT NULL et CHECK). La première partie des exercices (de 1.1 à 1.4 concerne la base *Parc Informatique*). Le dernier exercice traite d'une autre base (*Chantiers*) qui s'appliquera à une base 11g.

### Exercice 1.1 Présentation de la base de données

Une entreprise désire gérer son parc informatique à l'aide d'une base de données. Le bâtiment est composé de trois étages. Chaque étage possède son réseau (ou segment distinct) Ethernet. Ces réseaux traversent des salles équipées de postes de travail. Un poste de travail est une machine sur laquelle sont installés certains logiciels. Quatre catégories de postes de travail sont recensées (stations Unix, terminaux X, PC Windows et PC NT). La base de données devra aussi décrire les installations de logiciels.

Les noms et types des colonnes sont les suivants :

**Tableau 1-13** Caractéristiques des colonnes

Colonne	Commentaires	Types
indIP	Trois premiers groupes IP (exemple : 130.120.80).	VARCHAR2 (11)
nomSegment	Nom du segment.	VARCHAR2 (20)
etage	Étage du segment.	NUMBER (2)
nSalle	Numéro de la salle.	VARCHAR2 (7)
nomSalle	Nom de la salle.	VARCHAR2 (20)
nbPoste	Nombre de postes de travail dans la salle.	NUMBER (2)
nPoste	Code du poste de travail.	VARCHAR2 (7)
nomPoste	Nom du poste de travail.	VARCHAR2 (20)
ad	Dernier groupe de chiffres IP (exemple : 11).	VARCHAR2 (3)
typePoste	Type du poste (Unix, TX, PCWS, PCNT).	VARCHAR2 (9)
dateIns	Date d'installation du logiciel sur le poste.	DATE
nLog	Code du logiciel.	VARCHAR2 (5)
nomLog	Nom du logiciel.	VARCHAR2 (20)
dateAch	Date d'achat du logiciel.	DATE
version	Version du logiciel.	VARCHAR2 (7)
typeLog	Type du logiciel (Unix, TX, PCWS, PCNT).	VARCHAR2 (9)
prix	Prix du logiciel.	NUMBER (6, 2)
numIns	Numéro séquentiel des installations.	NUMBER (5)
dateIns	Date d'installation du logiciel.	DATE
delai	Intervalle entre achat et installation.	INTERVAL DAY (5) TO SECOND (2) ,
typeLP	Types des logiciels et des postes.	VARCHAR2 (9)
nomType	Noms des types (Termiaux X, PC Windows...).	VARCHAR2 (20)

## Exercice 1.2 Création des tables

Écrivez puis exécutez le script SQL (que vous appellerez `creParc.sql`) de création des tables avec leur clé primaire (en gras dans le schéma suivant) et les contraintes suivantes :

- Les noms des segments, des salles et des postes sont non nuls.
- Le domaine de valeurs de la colonne `ad` s'étend de 0 à 255.
- La colonne `prix` est supérieure ou égale à 0.
- La colonne `dateIns` est égale à la date du jour par défaut.

Figure 1-7 Schéma des tables

Segment

<b>indIP</b>	nomSegment	etage
--------------	------------	-------

Salle

<b>nSalle</b>	nomSalle	nbPoste	indIP
---------------	----------	---------	-------

Poste

<b>nPoste</b>	nomPoste	indIP	ad	typePoste	nSalle
---------------	----------	-------	----	-----------	--------

Logiciel

<b>nLog</b>	nomLog	dateAch	version	typeLog	prix
-------------	--------	---------	---------	---------	------

Installer

<b>nPoste</b>	<b>nLog</b>	numIns	dateIns	delai
---------------	-------------	--------	---------	-------

Types

<b>typeLP</b>	nomType
---------------	---------

## Exercice 1.3 Structure des tables

Écrivez puis exécutez le script SQL (que vous appellerez `descParc.sql`) qui affiche la description de toutes ces tables (en utilisant des commandes `DESC`). Comparez avec le schéma.

## Exercice 1.4 Destruction des tables

Écrivez puis exécutez le script SQL de destruction des tables (que vous appellerez `dropParc.sql`). Lancez ce script puis à nouveau celui de la création des tables.

## Exercice 1.5 Schéma de la base *Chantiers* (Oracle 11g)

Une société désire informatiser les visites des chantiers de ses employés. Pour définir cette base de données, une première étude fait apparaître les informations suivantes :

- Chaque employé est modélisé par un numéro, un nom et une qualification.
- Un chantier est caractérisé par un numéro, un nom et une adresse.
- L'entreprise dispose de véhicules pour lesquels il est important de stocker pour le numéro d'immatriculation, le type (un code valant par exemple 0 pour une camionnette, 1 pour une moto et 2 pour une voiture) ainsi que le kilométrage en fin d'année.
- Le gestionnaire a besoin de connaître les distances parcourues par un véhicule pour chaque visite d'un chantier.
- Chaque jour, un seul employé sera désigné conducteur des visites d'un véhicule.
- Pour chaque visite, il est important de pouvoir connaître les employés transportés.

Les colonnes à utiliser sont les suivantes :

**Tableau 1-14** Caractéristiques des colonnes à ajouter

Colonne	Commentaires	Types
kilometres	Kilométrage d'un véhicule lors d'une sortie.	NUMBER
n_conducteur	Numéro de l'employé conducteur.	VARCHAR2 (4)
n_transporte	Numéro de l'employé transporté.	VARCHAR2 (4)

L'exercice consiste à compléter le schéma relationnel ci-après (ajout de colonnes et définition des contraintes de clé primaire et étrangère).

```
CREATE TABLE Employe (n_emp VARCHAR(4), nom_emp VARCHAR(20),
  qualif_emp VARCHAR(12), CONSTRAINT pk_emp PRIMARY KEY(n_emp));

CREATE TABLE Chantier (n_chantier VARCHAR(10), nom_ch VARCHAR(10),
  adresse_ch VARCHAR(15), CONSTRAINT pk_chan PRIMARY KEY(n_chantier));

CREATE TABLE Vehicule (n_vehicule VARCHAR(10), type_vehicule VARCHAR(1),
  kilometrage NUMBER, CONSTRAINT pk_vehi PRIMARY KEY(n_vehicule));

CREATE TABLE Visite(n_chantier VARCHAR(10), n_vehicule VARCHAR(10),
  date_jour DATE, ...
  CONSTRAINT pk_visite PRIMARY KEY(...),
  CONSTRAINT fk_depl_chantier FOREIGN KEY(n_chantier) ...,
  CONSTRAINT fk_depl_vehicule FOREIGN KEY(n_vehicule) ...,
  CONSTRAINT fk_depl_employe FOREIGN KEY(n_conducteur) ... );

CREATE TABLE Transporter (...
  CONSTRAINT pk_transporter PRIMARY KEY (...),
  CONSTRAINT fk_transp_visite FOREIGN KEY ... ,
  CONSTRAINT fk_transp_employe FOREIGN KEY ...);
```

## Modifications comportementales

Nous étudions dans cette section les mécanismes d'ajout, de suppression, d'activation et de désactivation des contraintes.

Faisons évoluer le schéma suivant. Les clés primaires sont nommées `pk_Compagnie` pour la table `Compagnie` et `pk_Avion` pour la table `Avion`.

Figure 3-4 Schéma à faire évoluer

Compagnie

comp	nrue	rue	ville	nomComp
AF	124	Port Royal	Paris	Air France
SING	7	Camparols	Singapour	Singapore AL

Affreter

compAff	immat	dateAff	nbPax
AF	F-WTSS	13-05-2003	85
SING	F-GAFU	05-02-2003	155
AF	F-WTSS	15-05-2003	82

Avion

immat	typeAvion	nbHVol	proprio
F-WTSS	Concorde	6570	SING
F-GAFU	A320	3500	AF
F-GLFS	TB-20	2000	SING

## Ajout de contraintes

Jusqu'à présent, nous avons créé des tables en même temps que les contraintes. Il est possible de créer des tables seules (dans ce cas l'ordre de création n'est pas important et on peut même les créer par ordre alphabétique), puis d'ajouter les contraintes. Les outils de conception (*Win'Design*, *Designer* ou *PowerAMC*) adoptent cette démarche lors de la génération automatique de scripts SQL.

La directive `ADD CONSTRAINT` de l'instruction `ALTER TABLE` permet d'ajouter une contrainte à une table. La syntaxe générale est la suivante :

```
ALTER TABLE [schéma.]nomTable
ADD [CONSTRAINT nomContrainte] typeContrainte;
```

Comme pour l'instruction `CREATE TABLE`, quatre types de contraintes sont possibles :

- `UNIQUE (colonne1 [,colonne2]...)`
- `PRIMARY KEY (colonne1 [,colonne2]...)`  
`FOREIGN KEY (colonne1 [,colonne2]...)`  
`REFERENCES [schéma.]nomTablePère (colonne1 [,colonne2]...)`  
`[ON DELETE { CASCADE | SET NULL }]`
- `CHECK (condition)`

## Clé étrangère

Ajoutons la clé étrangère à la table Avion au niveau de la colonne `proprio` en lui assignant une contrainte `NOT NULL` :

```
ALTER TABLE Avion
  ADD (CONSTRAINT nn_proprio CHECK (proprio IS NOT NULL),
       CONSTRAINT fk_Avion_comp_Compag FOREIGN KEY(proprio)
       REFERENCES Compagnie(comp));
```

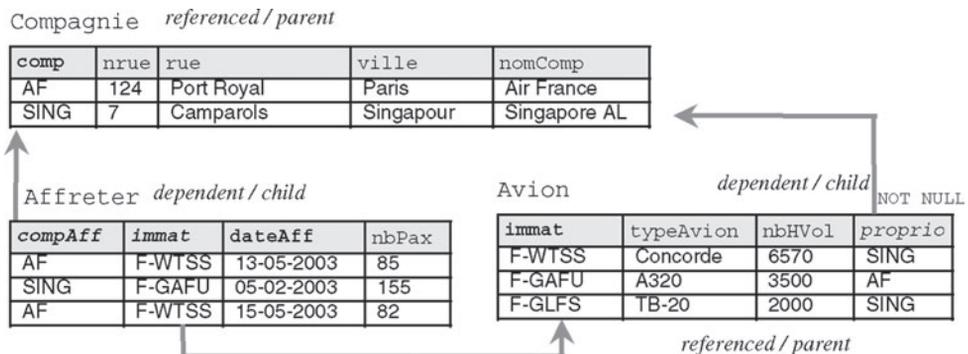
## Clé primaire

Ajoutons la clé primaire de la table `Affreter` et deux clés étrangères (vers les tables `Avion` et `Compagnie`) :

```
ALTER TABLE Affreter ADD (
  CONSTRAINT pk_Affreter PRIMARY KEY (compAff, immat, dateAff),
  CONSTRAINT fk_Aff_na_Avion FOREIGN KEY(immat) REFERENCES
  Avion(immat),
  CONSTRAINT fk_Aff_comp_Compag FOREIGN KEY(compAff)
  REFERENCES Compagnie(comp));
```

Pour que l'ajout d'une contrainte soit possible, il faut que les données présentes dans la table respectent la nouvelle contrainte (nous étudierons plus tard les moyens de pallier ce problème). Les tables contiennent les contraintes suivantes :

Figure 3-5 Après ajout de contraintes



## Suppression de contraintes

La directive `DROP CONSTRAINT` de l'instruction `ALTER TABLE` permet d'enlever une contrainte d'une table. La syntaxe générale est la suivante :

```
ALTER TABLE [schéma.]nomTable DROP CONSTRAINT nomContrainte [CASCADE];
```

La directive `CASCADE` supprime les contraintes référentielles des tables « pères ». On comprend mieux maintenant pourquoi il est si intéressant de nommer les contraintes plutôt que d'utiliser les noms automatiquement générés.

Supprimons la contrainte `NOT NULL` qui porte sur la colonne `proprio` de la table `Avion` :

```
ALTER TABLE Avion DROP CONSTRAINT nn_proprio;
```

### Clé étrangère

Supprimons la clé étrangère de la colonne `proprio`. Il n'est pas besoin de spécifier `CASCADE`, car il s'agit d'une table « fils » pour cette contrainte d'intégrité référentielle.

```
ALTER TABLE Avion DROP CONSTRAINT fk_Avion_comp_Compag;
```

### Clé primaire (ou candidate)

Supprimons la clé primaire de la table `Avion`. Il faut préciser `CASCADE`, car cette table est référencée par une clé étrangère dans la table `Affreter`. Cette commande supprime à la fois la clé primaire de la table `Avion` mais aussi les contraintes clés étrangères des tables dépendantes (ici seule la clé étrangère de la table `Affreter` est supprimée).

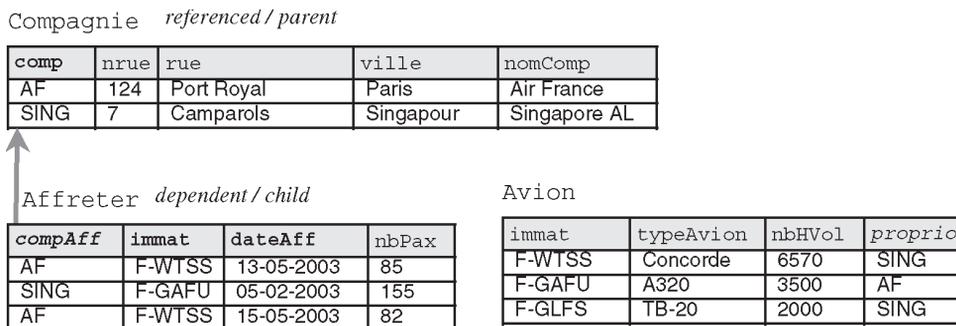
```
ALTER TABLE Avion DROP CONSTRAINT pk_Avion CASCADE;
```



Si l'option `CASCADE` n'avait pas été spécifiée, Oracle aurait renvoyé l'erreur « `ORA-02273: cette clé unique/primaire est référencée par des clés étrangères` ».

La figure suivante illustre les trois contraintes qui restent : les clés primaires des tables `Compagnie` et `Affreter` et la clé étrangère de la table `Affreter`.

Figure 3-6 Après suppression de contraintes



Les deux possibilités pour supprimer ces trois contraintes sont décrites dans le tableau suivant. La deuxième écriture est plus rigoureuse car elle prévient des effets de bord. Il suffit, pour les

éviter, de détruire les contraintes dans l'ordre inverse d'apparition dans le script de création (tables « fils » puis « pères »).

**Tableau 3-3 Suppression de contraintes**

Avec CASCADE	Sans CASCADE
ALTER TABLE Compagnie DROP CONSTRAINT pk_Compagnie <b>CASCADE</b> ; ALTER TABLE Affreter DROP CONSTRAINT pk_Affreter ;	ALTER TABLE Affreter DROP CONSTRAINT fk_Aff_comp_Compag ; ALTER TABLE Compagnie DROP CONSTRAINT pk_Compagnie ; ALTER TABLE Affreter DROP CONSTRAINT pk_Affreter ;

## Désactivation de contraintes

La désactivation de contraintes peut être intéressante pour accélérer des procédures de chargement (importation par SQL\*Loader) et d'exportation massive de données. Ce mécanisme améliore aussi les performances de programmes *batches* qui ne modifient pas des données concernées par l'intégrité référentielle ou pour lesquelles on vérifie la cohérence de la base à la fin.

La directive `DISABLE CONSTRAINT` de l'instruction `ALTER TABLE` permet de désactiver temporairement (jusqu'à la réactivation) une contrainte existante.

### Syntaxe

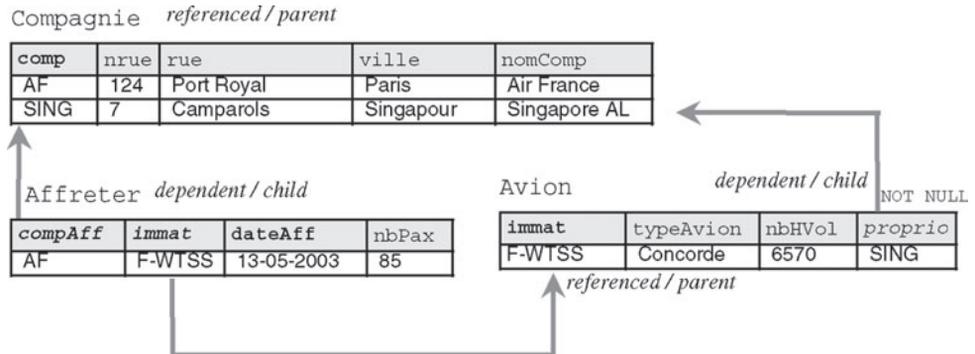
La syntaxe générale est la suivante :

```
ALTER TABLE [schéma.]nomTable
  DISABLE [ VALIDATE | NOVALIDATE ] CONSTRAINT nomContrainte
  [ CASCADE ] [ { KEEP | DROP } INDEX ] ;
```

- CASCADE répercute la désactivation des clés étrangères des tables « fils » dépendantes. Si vous voulez désactiver une clé primaire référencée par une clé étrangère sans cette option, le message d'Oracle renvoyé est : « ORA-02297: impossible désactiver contrainte... - les dépendences existent ».
- Les options `KEEP INDEX` et `DROP INDEX` permettent de préserver ou de détruire l'index dans le cas de la désactivation d'une clé primaire.
- Nous verrons plus loin l'explication des options `VALIDATE` et `NOVALIDATE`.

En considérant l'exemple suivant, désactivons quelques contraintes et insérons des enregistrements ne respectant pas les contraintes désactivées.

Figure 3-7 Avant la désactivation de contraintes



### Contrainte de vérification

Désactivons la contrainte NOT NULL qui porte sur la colonne `proprio` de la table `Avion` et insérons un avion qui n'est rattaché à aucune compagnie :

```
ALTER TABLE Avion DISABLE CONSTRAINT nn_proprio;
INSERT INTO Avion VALUES ('Bidon1', 'TB-20', 2000, NULL);
```

### Clé étrangère

Désactivons la contrainte de clé étrangère qui porte sur la colonne `proprio` de la table `Avion` et insérons un avion rattaché à une compagnie inexistante :

```
ALTER TABLE Avion DISABLE CONSTRAINT fk_Avion_comp_Compag;
INSERT INTO Avion VALUES ('F-GLFS', 'TB-22', 500, 'Toto');
```

### Clé primaire

Désactivons la contrainte de clé primaire de la table `Avion`, en supprimant en même temps l'index, et insérons un avion ne respectant plus la clé primaire :

```
ALTER TABLE Avion DISABLE CONSTRAINT pk_Avion CASCADE DROP INDEX;
INSERT INTO Avion VALUES ('Bidon1', 'TB-21', 1000, 'AF');
```

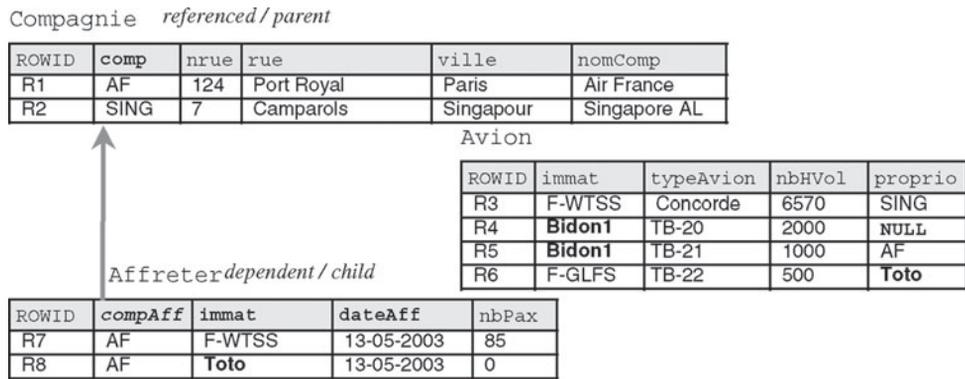
La désactivation de cette contrainte par `CASCADE` supprime aussi une des clés étrangères de la table `Affreter`. Insérons un affrètement qui référence un avion inexistant :

```
INSERT INTO Affreter VALUES ('AF', 'Toto', '13-05-2003', 0);
```

L'état de la base est désormais comme suit. Les *rowids* sont précisés pour illustrer les options de réactivation.

Bien qu'il semble incohérent de réactiver les contraintes sans modifier les valeurs ne respectant pas les contraintes (notées en gras), nous verrons que plusieurs alternatives sont possibles.

Figure 3-8 Après désactivation de contraintes



## Réactivation de contraintes

La directive `ENABLE CONSTRAINT` de l'instruction `ALTER TABLE` permet de réactiver une contrainte.

### Syntaxe

La syntaxe générale est la suivante :

```
ALTER TABLE [schéma.] nomTable
    ENABLE [ VALIDATE | NOVALIDATE ] CONSTRAINT nomContrainte
    [ USING INDEX ClauseIndex ] [ EXCEPTIONS INTO tableErreurs ] ;
```

- La clause d'index permet, dans le cas des clés primaires ou candidates ( `UNIQUE` ), de pouvoir recréer l'index associé.
- La clause d'exceptions permet de retrouver les enregistrements ne vérifiant pas la nouvelle contrainte (cas étudié au paragraphe suivant).



Il n'est pas possible de réactiver une clé étrangère tant que la contrainte de clé primaire référencée n'est pas active.

En supposant que les tables contiennent des données qui respectent les contraintes à réutiliser, la réactivation de la clé primaire (en recréant l'index) et d'une contrainte `NOT NULL` de la table `Avion` se programmerait ainsi :

```
ALTER TABLE Avion ENABLE CONSTRAINT pk_Avion
    USING INDEX (CREATE UNIQUE INDEX pk_Avion ON Avion (immat));
ALTER TABLE Avion ENABLE CONSTRAINT nn_proprio;
```

### Récupération de données erronées

L'option `EXCEPTIONS INTO` de l'instruction `ALTER TABLE` permet de récupérer automatiquement les enregistrements qui ne respectent pas des contraintes afin de les traiter (modifier, supprimer ou déplacer) avant de réactiver les contraintes en question sur une table saine.

Il faut créer une table composée de quatre colonnes :

- La première, de type `ROWID`, contiendra les adresses des enregistrements ne respectant pas la contrainte ;
- la deuxième colonne de type `varchar2(30)` contiendra le nom du propriétaire de la table ;
- la troisième colonne de type `varchar2(30)` contiendra le nom de la table ;
- la quatrième, de type `varchar2(30)`, contiendra le nom de la contrainte.

Le tableau suivant décrit deux tables permettant de stocker les enregistrements erronés après réactivation de contraintes.



Il est permis d'utiliser des noms de table ou de colonne différents mais il n'est pas possible d'utiliser une structure de table différente.

Tableau 3-4 Tables de rejets

Tables conventionnelles ( <i>heap</i> )	Toutes tables ( <i>heap, index-organized</i> )
<pre>CREATE TABLE Problemes (adresse ROWID, utilisateur VARCHAR2(30), nomTable VARCHAR2(30), nomContrainte VARCHAR2(30));</pre>	<pre>CREATE TABLE ProblemesBis (adresse UROWID, utilisateur VARCHAR2(30), nomTable VARCHAR2(30), nomContrainte VARCHAR2(30));</pre>



La commande de réactivation d'une contrainte avec l'option `met` automatiquement à jour la table des rejets et renvoie une erreur s'il existe un enregistrement ne respectant pas la contrainte.

Réactivons la contrainte `NOT NULL` concernant la colonne `proprio` de la table `Avion` (enregistrement incohérent de `ROWID R4`) :

```
ALTER TABLE Avion ENABLE CONSTRAINT nn_proprio EXCEPTIONS INTO
Problemes;
```

```
ORA-02293: impossible de valider (SOUTOU.NN_PROPRIO) - violation
d'une contrainte de contrôle
```

Réactivons la contrainte de clé étrangère sur cette même colonne (enregistrement incohérent : `ROWID R6` n'a pas de compagnie référencée).

```
ALTER TABLE Avion ENABLE CONSTRAINT fk_Avion_comp_Compag
    EXCEPTIONS INTO Problemes;

ORA-02298: impossible de valider (SOUTOU.FK_AVION_COMP_COMPAG) -
clés parents introuvables
```

Réactivons la contrainte de clé primaire de la table Avion (enregistrements incohérents : ROWID R5 et R6 ont la même immatriculation) :

```
ALTER TABLE Avion ENABLE CONSTRAINT pk_Avion EXCEPTIONS INTO
    Problemes;

ORA-02437: impossible de valider (SOUTOU.PK_AVION) - violation de
la clé primaire
```

La table Problemes contient à présent les enregistrements suivants :

**Figure 3-9** Table des rejets

Problemes

adresse	utilisateur	nomTable	nomContrainte
R4	nomUserOracle	AVION	NN_PROPRIO
R6	nomUserOracle	AVION	FK_AVION_COMP_COMPAG
R5	nomUserOracle	AVION	PK_AVION
R4	nomUserOracle	AVION	PK_AVION

Il apparaît que les trois enregistrements (R4, R5 et R6) ne respectent pas des contraintes dans la table Avion. Il convient de les traiter au cas par cas et par type de contrainte. Il est possible d'automatiser l'extraction des enregistrements qui ne respectent pas les contraintes en faisant une jointure (voir le chapitre suivant) entre la table des exceptions et la table des données (on testera la valeur des *rowids*).

Dans notre exemple, choisissons :

- de modifier l'immatriculation de l'avion 'Bidon1' (*rowid* R4) en 'F-TB20' dans la table Avion :

```
UPDATE Avion SET immat = 'F-TB20'
    WHERE immat = 'Bidon1' AND typeAvion = 'TB-20';
```

- d'affecter la compagnie 'AF' aux avions n'appartenant pas à la compagnie 'SING' dans la table Avion (mettre à jour les enregistrements de *rowid* R4 et R6) :

```
UPDATE Avion SET proprio = 'AF' WHERE NOT(proprio = 'SING');
```

- de modifier l'immatriculation de l'avion 'Toto' en 'F-TB20' dans la table Affreter :

```
UPDATE Affreter SET immat = 'F-TB20' WHERE immat = 'Toto';
```

Avant de réactiver à nouveau les contraintes, il convient de supprimer les lignes de la table d'exceptions (ici Problemes). La réactivation de toutes les contraintes avec l'option EXCEPTIONS INTO ne génère plus aucune erreur et la table d'exceptions est encore vide.

# Exceptions

Afin d'éviter qu'un programme s'arrête à la première erreur (requête ne retournant aucune ligne, valeur incorrecte à écrire dans la base, conflit de clés primaires, division par zéro, etc.), il est indispensable de prévoir tous les cas potentiels d'erreurs et d'associer à chacun de ces cas la programmation d'une exception PL/SQL. Dans le vocabulaire des programmeurs on dit qu'on *garde la main* pendant l'exécution du programme. Le mécanisme des exceptions (*handling errors*) est largement utilisé par tous les programmeurs car il est prépondérant dans la mise en œuvre des transactions.

Les exceptions peuvent se programmer dans un bloc PL/SQL, un sous-programme (fonction ou procédure cataloguée), dans un paquetage ou un déclencheur.

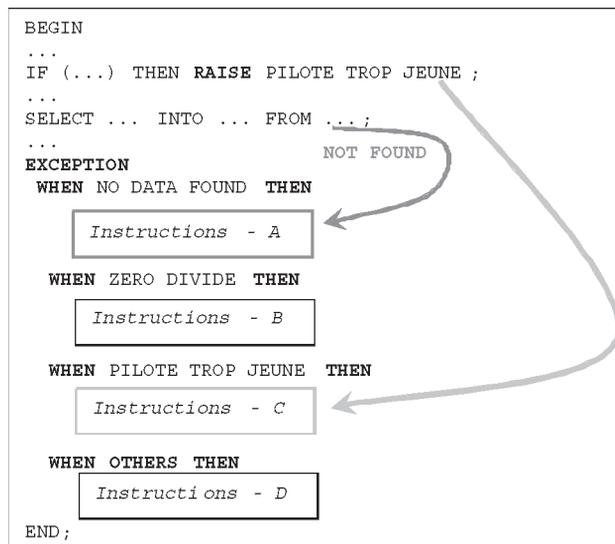
## Généralités

Une exception PL/SQL correspond à une condition d'erreur et est associée à un identificateur. Une exception est détectée (aussi dite « levée ») au cours de l'exécution d'une partie de programme (entre un BEGIN et un END). Une fois levée, l'exception termine le corps principal des instructions et renvoie au bloc EXCEPTION du programme en question.

La figure suivante illustre les deux mécanismes qui peuvent déclencher une exception :

- Une erreur Oracle se produit, l'exception associée est déclenchée automatiquement (exemple du SELECT ne ramenant aucune ligne, ce qui déclenche l'exception ORA-01403 d'identificateur NO\_DATA\_FOUND).

Figure 7-6 Principe général des exceptions



- Le programmeur désire dérouter volontairement (par l'intermédiaire de l'instruction `RAISE`) son programme dans le bloc des exceptions sous certaines conditions. L'exception est ici manuellement déclenchée et peut appartenir à l'utilisateur (ici la condition `PILOTE_TROP_JEUNE`) ou être prédéfinie au niveau d'Oracle (division par zéro d'identificateur `ZERO_DIVIDE` qui sera automatiquement déclenchée).

Si aucune erreur ne se produit, le bloc est ignoré et le traitement se termine (ou retourne à son appelant s'il s'agit d'un sous-programme).

La syntaxe générale d'un bloc d'exceptions est la suivante. Il est possible de grouper plusieurs exceptions pour programmer le même traitement. La dernière entrée (`OTHERS`) doit être éventuellement toujours placée en fin du bloc d'erreurs.

#### **EXCEPTION**

```

WHEN exception1 [OR exception2 ...] THEN
    instructions;
[WHEN exception3 [OR exception4 ...] THEN
    instructions; ]
[WHEN OTHERS THEN
    instructions; ]

```

Si une anomalie se produit, le bloc `EXCEPTION` s'exécute.

- Si le programme prend en compte l'erreur dans une entrée `WHEN...`, les instructions de cette entrée sont exécutées et le programme se termine.
- Si l'exception n'est pas prise en compte dans le bloc `EXCEPTION` :
  - il existe une section `OTHERS` où des instructions s'exécutent ;
  - il n'existe pas une section `OTHERS` et l'exception sera propagée au programme appelant (une section traite de la propagation des exceptions).

Étudions à présent les trois types d'exceptions qui existent sous PL/SQL, en programmant des procédures simples interrogeant la table `Pilote` illustrée à la figure 7-3.

## Exception interne prédéfinie

Les exceptions prédéfinies sont celles qui se produisent le plus souvent. Oracle affecte un nom de manière à les traiter plus facilement dans le bloc `EXCEPTION`. Le tableau suivant les décrit :

**Tableau 7-22 Exceptions prédéfinies**

Nom de l'exception	Numéro	Commentaires
<code>ACCESS_INTO_NULL</code>	ORA-06530	Affectation d'une valeur à un objet non initialisé.
<code>CASE_NOT_FOUND</code>	ORA-06592	Aucun des choix de la structure <code>CASE</code> sans <code>ELSE</code> n'est effectué.
<code>COLLECTION_IS_NULL</code>	ORA-06531	Utilisation d'une méthode autre que <code>EXISTS</code> sur une collection ( <i>nested table</i> ou <i>varray</i> ) non initialisée.
<code>CURSOR_ALREADY_OPEN</code>	ORA-06511	Ouverture d'un curseur déjà ouvert.
<code>DUP_VAL_ON_INDEX</code>	ORA-00001	Insertion d'une ligne en doublon (clé primaire).
<code>INVALID_CURSOR</code>	ORA-01001	Ouverture interdite sur un curseur.
<code>INVALID_NUMBER</code>	ORA-01722	Échec d'une conversion d'une chaîne de caractères en <code>NUMBER</code> .
<code>LOGIN_DENIED</code>	ORA-01017	Connexion incorrecte.
<code>NO_DATA_FOUND</code>	ORA-01403	Requête ne retournant aucun résultat.
<code>NOT_LOGGED_ON</code>	ORA-01012	Connexion inexistante.
<code>PROGRAM_ERROR</code>	ORA-06501	Problème PL/SQL interne (invitation au contact du support...).
<code>ROWTYPE_MISMATCH</code>	ORA-06504	Incompatibilité de types entre une variable externe et une variable PL/SQL.
<code>SELF_IS_NULL</code>	ORA-30625	Appel d'une méthode d'un type sur un objet <code>NULL</code> (extension objet).
<code>STORAGE_ERROR</code>	ORA-06500	Dépassement de capacité mémoire.
<code>SUBSCRIPT_BEYOND_COUNT</code>	ORA-06533	Référence à un indice incorrect d'une collection ( <i>nested table</i> ou <i>varray</i> ) ou variables de type <code>TABLE</code> .
<code>SUBSCRIPT_OUTSIDE_LIMIT</code>	ORA-06532	
<code>SYS_INVALID_ROWID</code>	ORA-01410	Échec d'une conversion d'une chaîne de caractères en <code>ROWID</code> .
<code>TIMEOUT_ON_RESOURCE</code>	ORA-00051	Dépassement du délai alloué à une ressource.
<code>TOO_MANY_ROWS</code>	ORA-01422	Requête retournant plusieurs lignes.
<code>VALUE_ERROR</code>	ORA-06502	Erreur arithmétique (conversion, troncature, taille) d'un <code>NUMBER</code> .
<code>ZERO_DIVIDE</code>	ORA-01476	Division par zéro.

Le code d'erreur (SQLCODE) qui peut être récupéré par un programme d'application (Java par exemple sous JDBC), est inclus dans le numéro interne de l'erreur (pour la deuxième exception, il s'agit de -6 592).



Concernant l'erreur `NO_DATA_FOUND`, rappelez-vous qu'elle n'est opérationnelle qu'avec l'instruction `SELECT`. Une mise à jour ou une suppression (`UPDATE` et `DELETE`) d'un enregistrement inexistant ne déclenche pas l'exception. Pour gérer ces cas d'erreurs, il faut utiliser un curseur explicite et une exception utilisateur (voir la section « Utilisation du curseur explicite »).

Si vous désirez programmer une erreur qui n'apparaît pas dans cette liste (exemple : erreur référentielle pour une suppression d'un enregistrement d'une table identifiée par une clé étrangère), il faudra programmer une exception non prédéfinie (voir la section suivante).

### Plusieurs erreurs

Le tableau suivant décrit une procédure qui gère deux erreurs : aucun pilote n'est associé à la compagnie de code passé en paramètre (`NO_DATA_FOUND`) et plusieurs pilotes le sont (`TOO_MANY_ROWS`). Le programme se termine correctement si la requête retourne une seule ligne (cas de la compagnie de code 'CAST').

Tableau 7-23 Deux exceptions traitées

Code PL/SQL	Commentaires
<pre>CREATE PROCEDURE procException1 (p_comp IN VARCHAR2) IS   var1 Pilote.nom%TYPE; BEGIN   SELECT nom INTO var1 FROM Pilote     WHERE comp = p_comp;   DBMS_OUTPUT.PUT_LINE('Le pilote de la compagnie '        p_comp    ' est '    var1);</pre>	Requête déclenchant potentiellement deux exceptions prévues.
<pre><b>EXCEPTION</b> <b>WHEN NO_DATA_FOUND THEN</b>   DBMS_OUTPUT.PUT_LINE('La compagnie '        p_comp    ' n'a aucun pilote!');</pre>	Aucun résultat renvoyé.
<pre><b>WHEN TOO_MANY_ROWS THEN</b>   DBMS_OUTPUT.PUT_LINE('La compagnie '        p_comp    ' a plusieurs pilotes!');</pre>	Plusieurs résultats renvoyés.
<pre>END;</pre>	

La trace de l'exécution de cette procédure est la suivante :

```
SQL> EXECUTE procException1('AF');
```

```

La compagnie AF a plusieurs pilotes!
Procédure PL/SQL terminée avec succès.

SQL> EXECUTE procException1('RIEN');
La compagnie RIEN n'a aucun pilote!
Procédure PL/SQL terminée avec succès.

SQL> EXECUTE procException1('CAST');
Le pilote de la compagnie CAST est Thierry Millan
Procédure PL/SQL terminée avec succès.

```

Si une autre erreur se produit, en l'absence de la directive `OTHERS` dans le bloc d'exceptions, le programme se termine anormalement en renvoyant l'erreur en question. Dans notre exemple, seule une erreur interne pourrait éventuellement se produire (`PROGRAM_ERROR`, `STORAGE_ERROR`, `TIMEOUT_ON_RESOURCE`).

### *Même erreur sur différentes instructions*

Le tableau 7-21 décrit une procédure qui gère deux fois l'erreur non trouvée (`NO_DATA_FOUND`) sur deux requêtes distinctes. La première requête extrait le nom du pilote de code passé en paramètre. La deuxième extrait le nom du pilote ayant un nombre d'heures de vol égal à celui passé en paramètre. Le programme se termine correctement si les deux requêtes ne retournent qu'un seul enregistrement.

La directive `OTHERS` permet d'afficher en clair une autre erreur déclenchée par une des deux requêtes (ici notamment `TOO_MANY_ROWS` qui n'est pas prise en compte). Notez ici l'utilisation des deux variables d'Oracle : `SQLERRM` qui contient le message en clair de l'erreur et `SQLCODE` le code associé.

La trace de l'exécution de cette procédure est la suivante :

```

SQL> EXECUTE procException2('PL-1', 1000);
Le pilote de PL-1 est Gilles Laborde
Le pilote ayant 1000 heures est Florence Périssel
Procédure PL/SQL terminée avec succès.

SQL> EXECUTE procException2('PL-0', 2450);
Pas de pilote de brevet : PL-0
Procédure PL/SQL terminée avec succès.

```

Dans cette procédure, une erreur sur la première requête fait sortir le programme (après avoir traité l'exception) et de ce fait la deuxième requête n'est pas évaluée. Pour cela, il est intéressant d'utiliser des blocs imbriqués pour poursuivre le traitement après avoir traité une ou plusieurs exceptions.

Tableau 7-24 Une exception traitée pour deux instructions

Code PL/SQL	Commentaires
<pre>CREATE PROCEDURE procException2   (p_brevet IN VARCHAR2, p_heures IN NUMBER) IS   var1  Pilote.nom%TYPE;   requete NUMBER := 1; BEGIN   SELECT nom INTO var1 FROM Pilote     WHERE brevet = p_brevet;   DBMS_OUTPUT.PUT_LINE('Le pilote de '        p_brevet    ' est '    var1);    requete := 2;   SELECT nom INTO var1 FROM Pilote     WHERE nbhVol = p_heures;   DBMS_OUTPUT.PUT_LINE('Le pilote ayant '        p_heures    ' heures est '    var1);</pre>	Requêtes déclenchant potentiellement une exception prévue.
<pre><b>EXCEPTION</b> <b>WHEN NO_DATA_FOUND THEN</b>   IF requete = 1 THEN     DBMS_OUTPUT.PUT_LINE('Pas de pilote de brevet : '          p_brevet);   ELSE     DBMS_OUTPUT.PUT_LINE('Pas de pilote ayant ce       nombre d''heures de vol : '    p_heures);   END IF; <b>WHEN OTHERS THEN</b>   DBMS_OUTPUT.PUT_LINE('Erreur d''Oracle '        <b>SQLERRM</b>    ' ('    <b>SQLCODE</b>    ')');</pre>	Aucun résultat. Traitement pour savoir quelle requête a déclenché l'exception.
<pre>END;</pre>	Autre erreur.

### Imbrication de blocs d'erreurs

Le tableau suivant décrit une procédure qui inclut un bloc d'exceptions imbriqué au code principal. Ce mécanisme permet de poursuivre l'exécution après qu'Oracle a levé une exception. Dans cette procédure, les deux requêtes sont évaluées indépendamment du résultat retourné par chacune d'elles.

L'exécution suivante de cette procédure déclenche les deux exceptions. Le message d'erreur est contrôlé par le dernier cas d'exception, il ne s'agit pas d'une interruption anormale du programme.

```
SQL> EXECUTE procException3('PL-0', 2450);
Pas de pilote de brevet : PL-0
Erreur d'Oracle ORA-01422: l'extraction exacte ramène plus que le
nombre de lignes demandé (-1422)
```

Tableau 7-25 Bloc d'exceptions imbriqué

Code PL/SQL	Commentaires
<pre> CREATE PROCEDURE procException3     (p_brevet IN VARCHAR2, p_heures IN NUMBER) IS     var1 Pilote.nom%TYPE; BEGIN     BEGIN         SELECT nom INTO var1 FROM Pilote             WHERE brevet = p_brevet;         DBMS_OUTPUT.PUT_LINE('Le pilote de '    p_brevet                ' est '    var1);         EXCEPTION         WHEN NO_DATA_FOUND THEN             DBMS_OUTPUT.PUT_LINE('Pas de pilote de brevet : '                    p_brevet);         WHEN OTHERS THEN             DBMS_OUTPUT.PUT_LINE('Erreur d''Oracle '                    SQLERRM    ' ('    SQLCODE    ')');     END;         </pre>	<p>Bloc imbriqué.</p> <p>Gestion des exceptions de la première requête.</p>
<pre> SELECT nom INTO var1 FROM Pilote     WHERE nbhVol = p_heures ;     DBMS_OUTPUT.PUT_LINE('Le pilote ayant '    p_heures            ' heures est '    var1);     EXCEPTION     WHEN NO_DATA_FOUND THEN         DBMS_OUTPUT.PUT_LINE('Pas de pilote ayant ce nombre             d''heures de vol : '    p_heures);     WHEN OTHERS THEN         DBMS_OUTPUT.PUT_LINE('Erreur d''Oracle '                SQLERRM    ' ('    SQLCODE    ')');     END;         </pre>	<p>Suite du traitement.</p> <p>Gestion des exceptions de la deuxième requête.</p>

## Exception utilisateur

Il est possible de définir ses propres exceptions. Cela pour bénéficier des blocs de traitements d'erreurs et aborder une erreur applicative comme une erreur renvoyée par la base. Cela améliore et facilite la maintenance et l'évolution des programmes car les erreurs applicatives peuvent très facilement être propagées aux programmes appelants.

### Déclaration

La déclaration du nom de l'exception doit se trouver dans la section déclarative du sous-programme.

```

| nomException EXCEPTION;
    
```

## Déclenchement

Une exception utilisateur ne sera pas levée de la même manière qu'une exception interne. Le programme doit explicitement dérouter le traitement vers le bloc des exceptions par la directive RAISE. L'instruction RAISE permet également de déclencher des exceptions prédéfinies.

Dans notre exemple, programmons les deux exceptions suivantes :

- erreur\_piloteTropJeune qui va interdire l'insertion des pilotes ayant moins de 200 heures de vol ;
- erreur\_piloteTropExpérimenté qui va interdire l'insertion des pilotes ayant plus de 20 000 heures de vol.

Le tableau suivant décrit cette procédure qui intercepte ces deux erreurs applicatives :

**Tableau 7-26 Exceptions utilisateur**

Code PL/SQL	Commentaires
<pre>CREATE PROCEDURE saisiePilote   (p_brevet IN VARCHAR2,p_nom IN VARCHAR2,    p_nbHVol IN NUMBER, p_comp IN VARCHAR2) IS   erreur_piloteTropJeune      EXCEPTION;   erreur_piloteTropExpérimenté EXCEPTION;</pre>	Déclaration de l'exception.
<pre>BEGIN   INSERT INTO Pilote (brevet,nom,nbHVol,comp)     VALUES (p_brevet,p_nom,p_nbHVol,p_comp);   IF p_nbHVol &lt; 200 THEN RAISE erreur_piloteTropJeune;   END IF;   IF p_nbHVol &gt; 20000 THEN     RAISE erreur_piloteTropExpérimenté;   END IF;   COMMIT;</pre>	Corps du traitement (validation).
<pre>EXCEPTION   WHEN erreur_piloteTropJeune THEN     ROLLBACK;     DBMS_OUTPUT.PUT_LINE ('Désolé, le pilote manque       d'expérience');</pre>	Gestion de l'exception.
<pre>  WHEN erreur_piloteTropExpérimenté THEN     ROLLBACK;     DBMS_OUTPUT.PUT_LINE ('Désolé, le pilote a       trop d'expérience');</pre>	Gestion des autres exceptions.
<pre>  WHEN OTHERS THEN     ROLLBACK;     DBMS_OUTPUT.PUT_LINE('Erreur d'Oracle '    SQLERRM          '('    SQLCODE    ')');</pre>	
<pre>END;</pre>	

La trace de l'exécution de cette procédure où l'on passe des valeurs en paramètres qui déclenchent les deux exceptions est la suivante.

```
SQL> EXECUTE saisiePilote('PL-9','Tuffery Michel', 199, 'AF');
Désolé, le pilote manque d'expérience
Procédure PL/SQL terminée avec succès.

SQL> EXECUTE saisiePilote('PL-9','Tuffery Michel', 2001, 'AF');
Désolé, le pilote a trop d'expérience
Procédure PL/SQL terminée avec succès.
```

## Utilisation du curseur implicite

Étudiés dans le chapitre 6, les curseurs implicites permettent ici de pallier le fait qu'Oracle ne lève pas l'exception `NO_DATA_FOUND` pour les instructions `UPDATE` et `DELETE`. Ce qui est en théorie valable (aucune action sur la base peut ne pas être considérée comme une erreur), en pratique il est utile de connaître le code retour de l'instruction de mise à jour.

Considérons à nouveau la procédure `détruitCompagnie` en prenant en compte l'erreur applicative `erreur_compagnieInexistante` qui intercepte une suppression non réalisée. Le test du curseur implicite de cette instruction déclenche l'exception utilisateur associée.

Tableau 7-27 Utilisation du curseur implicite

Code PL/SQL	Commentaires
<pre>CREATE OR REPLACE PROCEDURE détruitCompagnie     (p_comp IN VARCHAR2) IS     erreur_ilResteUnPilote EXCEPTION;     PRAGMA EXCEPTION_INIT(erreur_ilResteUnPilote , -2292);     erreur_compagnieInexistante EXCEPTION;</pre>	Déclaration des exceptions.
<pre>BEGIN     DELETE FROM Compagnie WHERE comp = p_comp;     IF SQL%NOTFOUND THEN         RAISE erreur_compagnieInexistante;     END IF;     COMMIT;     DBMS_OUTPUT.PUT_LINE('Compagnie '    p_comp    ' détruite.');</pre>	Corps du traitement (validation).
<pre>EXCEPTION     WHEN erreur_ilResteUnPilote THEN         DBMS_OUTPUT.PUT_LINE ('Désolé, il reste encore un             pilote à la compagnie '    p_comp);     WHEN erreur_compagnieInexistante THEN         DBMS_OUTPUT.PUT_LINE ('La compagnie '    p_comp                ' n''existe pas dans la base!');</pre>	Gestion des exceptions.
<pre>WHEN OTHERS THEN     DBMS_OUTPUT.PUT_LINE('Erreur d'Oracle '    SQLERRM            '('    SQLCODE    ')');</pre>	Gestion des autres exceptions.
<pre>END;</pre>	

L'exécution de cette procédure où l'on passe un code compagnie inexistant fait maintenant dérouler la section des exceptions.

```
SQL> EXECUTE détruitCompagnie('rien');
La compagnie rien n'existe pas dans la base!
```

## Exception interne non prédéfinie

Pour intercepter une erreur Oracle qui n'a pas été prédéfinie (pour laquelle Oracle n'a pas associé de nom), et être ainsi plus précis qu'avec la clause `OTHERS`, il faut utiliser la directive `PRAGMA EXCEPTION_INIT`. Celle-ci indique au compilateur d'associer un nom d'exception, que vous aurez choisi, à un code d'erreur Oracle existant. La directive `PRAGMA` (appelée aussi pseudo-instruction) est un mot-clé signifiant que l'instruction est destinée au compilateur (elle n'est pas traitée au moment de l'exécution).

### Déclaration

Deux commandes sont nécessaires dans la section déclarative à la mise en œuvre de ce mécanisme : déclarer le nom de l'exception et associer cet identificateur à l'erreur Oracle.

```
nomException EXCEPTION;
PRAGMA EXCEPTION_INIT(nomException, numéroErreurOracle);
```



Pour connaître le numéro de l'erreur qui vous intéresse, consultez la liste des erreurs dans la documentation d'Oracle (*Error Messages* qui est classée par numéros croissants et non pas par fonctionnalités). Cherchez par exemple les entrées correspondant à *foreign key* dans le chapitre des erreurs `ORA-02100` to `ORA-04099`.

Vous pouvez aussi écrire un bloc PL/SQL qui programme volontairement l'erreur pour voir sous SQL\*Plus le numéro qu'Oracle renvoie.

### Déclenchement

Une exception non prédéfinie sera levée de la même manière qu'une exception prédéfinie, à savoir suite à une instruction SQL pour laquelle le serveur aura renvoyé une erreur.

Considérons les deux tables suivantes. La colonne `comp` de la table `Pilote` est clé étrangère vers la table `Compagnie`. Programmons une procédure qui supprime une compagnie de code passé en paramètre.

Figure 7-7 Deux tables

Compagnie

comp	ville	nomComp
AF	Paris	Air France
SING	Singapour	Singapore AL
CAST	Blagnac	Castanet AL
EJET	Dublin	Easy Jet

à détruire

Pilote

brevet	nom	nbHVol	comp
PL-1	Gilles Laborde	2450	AF
PL-2	Frédéric D'Almeyda	900	AF
PL-3	Florence Périssel	1000	SING
PL-4	Thierry Millan	2450	CAST
PL-5	Christine Royo	200	AF
PL-6	Aurélia Ente	2450	SING

Le tableau suivant décrit la procédure `détruitCompagnie` qui intercepte l'erreur `ORA-02292: enregistrement fils existant`. Il s'agit de contrôler le programme si la compagnie à détruire possède encore des pilotes référencés dans la table `Pilote`.

Tableau 7-28 Exception interne non prédéfinie

Code PL/SQL	Commentaires
<pre>CREATE PROCEDURE détruitCompagnie(p_comp IN VARCHAR2) IS   erreur_ilResteUnPilote EXCEPTION;   PRAGMA EXCEPTION_INIT(erreur_ilResteUnPilote , -2292);</pre>	Déclaration de l'exception.
<pre>BEGIN   DELETE FROM Compagnie WHERE comp = p_comp;   COMMIT;   DBMS_OUTPUT.PUT_LINE ('Compagnie '    p_comp        ' détruite.');</pre>	Corps du traitement (validation).
<pre>EXCEPTION   WHEN erreur_ilResteUnPilote THEN     DBMS_OUTPUT.PUT_LINE ('Désolé, il reste encore un       pilote à la compagnie '    p_comp);   WHEN OTHERS THEN     DBMS_OUTPUT.PUT_LINE('Erreur d''Oracle '    SQLERRM          '('    SQLCODE    ')');</pre>	Gestion de l'exception.
<pre>END;</pre>	Gestion des autres exceptions.

La trace de l'exécution de cette procédure est la suivante. Notez que si on applique cette procédure à une compagnie inexistante, le programme se termine normalement sans passer dans la section des exceptions.

```
SQL> EXECUTE détruitCompagnie('AF');
Désolé, il reste encore un pilote à la compagnie AF
Procédure PL/SQL terminée avec succès.

SQL> EXECUTE détruitCompagnie('EJET');
Compagnie EJET détruite.
Procédure PL/SQL terminée avec succès.
```

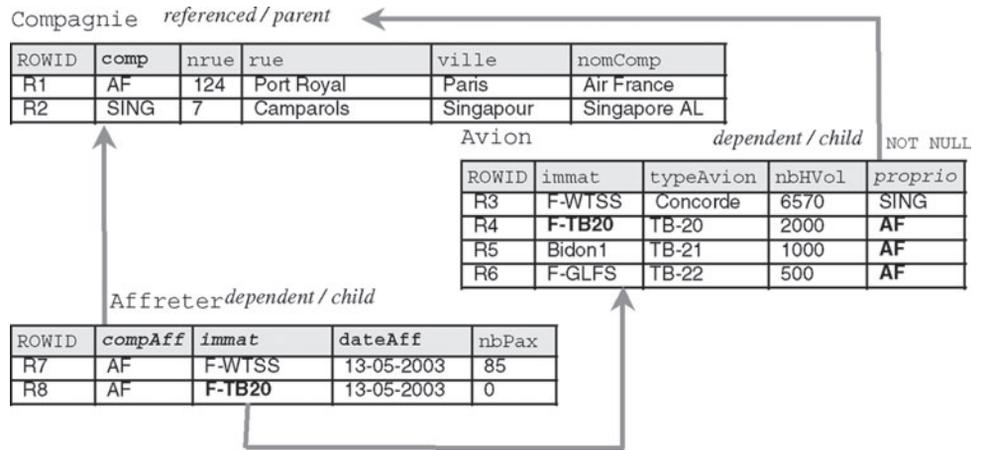
## Propagation d'une exception

Nous avons vu jusqu'à présent que lorsqu'un bloc `EXCEPTION` traite correctement une exception (car il existe soit une entrée dans le bloc correspondant à l'exception, soit l'entrée `OTHERS`), l'exécution du traitement se poursuit en séquences après l'instruction `END` du bloc `EXCEPTION`.

```
DELETE FROM Problemes ;
ALTER TABLE Avion ENABLE CONSTRAINT nn_proprio EXCEPTIONS INTO
Problemes;
ALTER TABLE Avion ENABLE CONSTRAINT fk_Avion_comp_Compag
EXCEPTIONS INTO Problemes;
ALTER TABLE Avion ENABLE CONSTRAINT pk_Avion EXCEPTIONS INTO
Problemes;
ALTER TABLE Affreter ENABLE CONSTRAINT fk_Aff_na_Avion
EXCEPTIONS INTO Problemes;
```

L'état de la base avec les contraintes réactivées est le suivant (les mises à jour sont en gras) :

Figure 3-10 Tables après modification et réactivation des contraintes



## Contraintes différées

Une contrainte est dite « différée » (*deferred*) si elle déclenche sa vérification seulement en atteignant le premier COMMIT rencontré. Si la contrainte n'existe pas, aucune commande de la transaction (suite d'instructions terminées par COMMIT) n'est réalisée. Les contraintes que nous avons étudiées jusqu'à maintenant étaient des contraintes immédiates (*immediate*) qui sont contrôlées après chaque instruction.

### Directives DEFERRABLE et INITIALLY

Depuis la version 8i, il est possible de différer à la fin d'un traitement la vérification des contraintes par les directives DEFERRABLE et INITIALLY.

Le second tableau compare les outils en fonction d'autres paramètres d'utilisation.

**Tableau 14-30 Autres facteurs influant les utilitaires**

	Explain plan	Autotrace	Runstats	SQL Trace/10046
<i>Affecté par la mise en cache</i>	Non	Oui	Oui	Oui
<i>Estimé ou réalisé</i>	Estimé	Estimé	Réalisé	Réalisé
<i>Fourni par Oracle</i>	Oui	Oui	Non	Oui
<i>Post-processing</i>	Non	Non	Non	Oui (tkprof, OraSRP)
<i>Facilité de comparaison</i>	Difficile	Oui	Oui	Difficile

L'outil *SQL Trace/Event 10046* semble le plus polyvalent du fait de la production de plans réels. *Explain Plan* et *Autotrace* sont des solutions simples à mettre en œuvre mais toutefois moins fiables (plans prévisionnels). Les résultats de runstats peuvent fluctuer fortement (cache et version du SGBD utilisé).

## Organisation des données

Cette section décrit les composants de la boîte à outils qui vous servira à optimiser vos applications. Plusieurs mécanismes peuvent être conjointement mis en œuvre : les contraintes, les index, la mise en *cluster*, le partitionnement, les vues matérialisées et la dénormalisation.

### Des contraintes au plus près des données

Vous devez définir, sur vos colonnes, le maximum de contraintes d'intégrité afin de renseigner au mieux l'optimiseur. Bien que la contrainte *CHECK* ne soit pas encore utilisée par l'optimiseur, il est possible que dans le temps cette fonctionnalité soit présente.

#### *Les colonnes NOT NULL*

Le fait de déclarer des contraintes *NOT NULL* ne vous empêche pas de réaliser aussi des tests du côté de l'application. En effet, il peut être utile de vérifier qu'une valeur est présente dans un champ de saisie d'un formulaire plutôt que d'attendre d'envoyer un grand nombre d'octets au serveur qui renverra une erreur du fait d'un *NOT NULL*.

En supposant que la table *Sport* dispose de la colonne *federation* (dont les valeurs actuelles sont non nulles), le tableau suivant présente deux déclarations de contrainte *NOT NULL*.

**Tableau 14-31 Déclaration de NOT NULL**

Déclaration avec <i>CHECK</i>	Déclaration en ligne ( <i>in line</i> )
<pre>ALTER TABLE Sport   ADD CONSTRAINT ck_federation   CHECK (federation IS NOT NULL);</pre>	<pre>ALTER TABLE Sport   MODIFY federation NOT NULL;</pre>



Définissez NOT NULL sur le plus de colonnes possibles pour renseigner l'optimiseur.

Préférez toujours la seconde écriture (*in line constraint*), pour que l'optimiseur puisse intégrer cette information, alors qu'il ignorera la contrainte déclarée avec CHECK.

### Les colonnes UNIQUE

Pour toute contrainte UNIQUE, un index (unique) est créé. Une contrainte UNIQUE diffère d'une contrainte PRIMARY KEY par le fait que les valeurs NULL sont autorisées ; elle n'a donc pas vocation à identifier toute ligne.



Définissez UNIQUE sur les colonnes potentiellement uniques de sorte que l'optimiseur puisse bénéficier d'un index supplémentaire (la désactivation d'une contrainte UNIQUE provoque la suppression de l'index).

Le tableau suivant présente la déclaration d'une contrainte UNIQUE (création implicite d'un index de nom un\_nom\_prenom\_tel) et sa désactivation (suppression implicite d'un index). Comme il existe des homonymes au sein des adhérents, la contrainte UNIQUE minimale à mettre en œuvre est composée du nom, prénom et numéro de téléphone.

Tableau 14-32 Déclaration de UNIQUE

Déclaration de la contrainte	Désactivation
ALTER TABLE Adherent ADD CONSTRAINT un_nom_prenom_tel <b>UNIQUE</b> (nom, prenom, tel);	ALTER TABLE Adherent DISABLE CONSTRAINT un_nom_prenom_tel;



L'index multicolonne (nom+prenom+tel) sera bénéfique pour les extractions dont un prédicat est basé sur le nom, le prénom et le numéro, et sur un accès aux trois colonnes simultanément.

## Indexation

Les différents types d'index ont été brièvement présentés au chapitre 1. Sans index, toute recherche s'apparente à un parcours séquentiel de toute la table. Ainsi pour  $n$  lignes, le nombre moyen de lectures est égal  $n/2$ , ce qui est très pénalisant dès que le volume de données devient important. De plus, ce nombre d'accès croît proportionnellement avec le nombre de lignes (100 fois plus de lignes implique un temps d'accès 100 fois plus long).

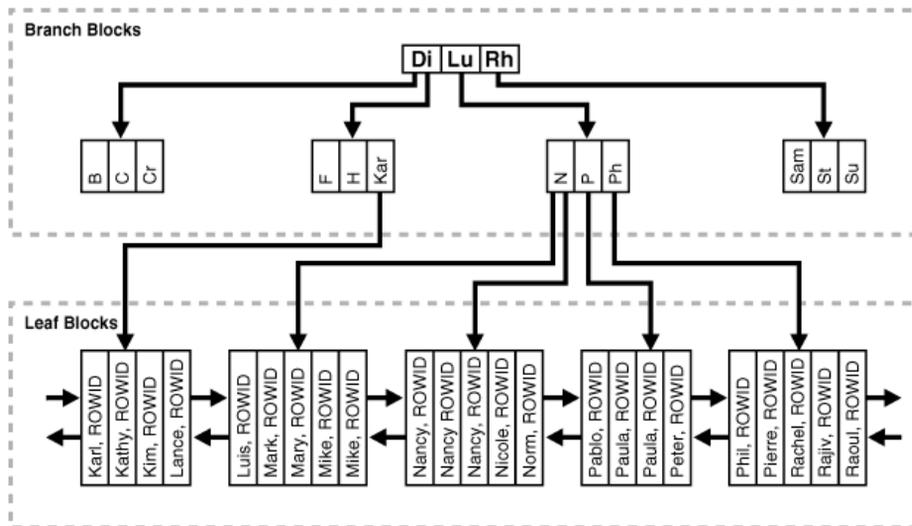
Étudions les cas d'utilisation des index d'Oracle de sorte à rendre une requête plus optimale.

## Index *B-tree*

Les index *B-tree* (*B* comme *Balanced*) sont constitués comme des arbres dont les noeuds aiguillent vers des sous-noeuds (suivant la valeur recherchée) jusqu'aux blocs feuille (*leaf blocks*) qui contiennent toutes les valeurs de l'index et les adresses de ligne (*rowid*) identifiant le segment de données associé. Les blocs feuilles sont doublement chaînés de sorte que l'index puisse être parcouru dans les deux sens sans passer par la racine.

Ce mécanisme est bien plus performant qu'un accès séquentiel car pour  $n$  lignes, le nombre moyen de lectures n'est plus proportionnel à  $n$  mais à  $\log(n)$ . La taille maximale d'une entrée d'index est en viron égale à la moitié de la taille des blocs de données (soit de l'ordre de 4 000 pointeurs pour une taille de bloc de 8 Ko).

Figure 14-7 Index *B-tree* (© doc. Oracle)



Un index *B-tree* est conçu automatiquement lors de la création de la clé primaire d'une table et d'une contrainte `UNIQUE`. Les arbres *B-tree* présentent de nombreux avantages :

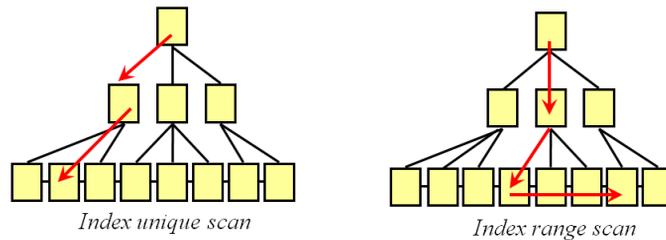
- Malgré les mises à jour de la table, ils restent équilibrés (les blocs feuilles sont au même niveau). En conséquence, quelle que soit la valeur cherchée, le temps de parcours est sensiblement identique. Les blocs intermédiaires sont remplis, en moyenne, aux trois-quarts de leur capacité.
- Les performances d'extraction, répondant à la majorité des prédicats des requêtes, sont excellentes, notamment les comparaisons d'égalité et d'intervalles.
- Les répercussions des mises à jour sont efficaces et ne se dégradent pas en fonction d'une forte augmentation de la taille des tables.

Nous ne traiterons pas ici des caractéristiques physiques des index (partitions, compression, pourcentages des tailles de blocs, etc.).

Les principales opérations que l'optimiseur réalise sur un index sont les suivantes :

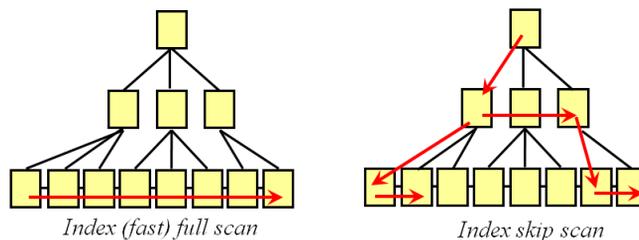
- *index unique scan* passe par la racine de l'arbre ; généralement toutes les colonnes de l'index sont concernées par une égalité dans le prédicat WHERE. Il s'agit en principe de la manière la plus optimale, mais qui n'est pas toujours utilisée par l'optimiseur au profit de *range scan*.
- *index range scan* passe par la racine de l'arbre et accède séquentiellement aux blocs feuille (doublement chaînés). Opération très utilisée par l'optimiseur, notamment lorsque une colonne de l'index est concernée par une inégalité dans le prédicat WHERE, et que l'index n'est pas unique. Dans tous ces cas, l'optimiseur juge qu'il est plus rapide de parcourir les feuilles de l'index plutôt que l'index lui-même.

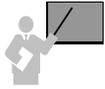
Figure 14-8 Accès direct et par parcours par intervalles d'un index B-tree



- *index full scan* et *index fast full scan* sont une alternative au parcours *full table scan* quand l'index contient toutes les colonnes nécessaires à la requête et qu'au moins une de ces colonnes est NOT NULL. Il ne peut pas être utilisé sur un index bitmap ; le parcours de l'index entier est plus rapide car il se réalise en mode lecture multibloc et peut être parallélisé.
- *index skip scan* (concerne les index multicolonne) utilise l'index alors que la (ou les) première(s) colonne(s) de l'index n'est (ne sont) pas présente(s) dans le prédicat WHERE.

Figure 14-9 Parcours séquentiel et par saut d'un index B-tree





Généralement, afin d'isoler le stockage physique des index, on utilise un *tablespace* dédié (qui peut se trouver sur un autre disque que celui des données). Il est aussi d'usage de créer un index pour chaque clé étrangère afin de rendre plus efficace les jointures.

Le tableau suivant présente d'une part la création de l'espace de stockage pour héberger les index et, d'autre part, la création d'index affectés à cet espace (une clé primaire et une clé étrangère non unique).

**Tableau 14-33 Création d'index en association avec un tablespace**

Création de l'espace	Création d'index
<pre>CREATE TABLESPACE tbs_index   DATAFILE 'tbs_index.dat'   SIZE 500M REUSE   AUTOEXTEND ON   NEXT 500K MAXSIZE 2000M;</pre>	<pre>--colonne "clé primaire" ALTER TABLE Adherent   ADD CONSTRAINT pk_Adherent PRIMARY KEY   (adhid)   USING INDEX TABLESPACE tbs_index; ... --colonne "clé étrangère" CREATE INDEX idx_Pratique_adhid   ON Pratique (adhid) TABLESPACE tbs_index;</pre>

Pour se convaincre de l'utilité des index, exécutez la requête avec et sans index (il s'agit d'une division) qui extrait les adhérents inscrits à tous les sports. L'adhérente la plus sportive est, sans conteste, Céline Larrazet et il faut 36 secondes sans indexage pour découvrir l'identité de la championne alors que la réponse est quasi instantanée en présence d'index sur les clés étrangères. Les chiffres sont éloquent même pour une volumétrie réduite (24 000 adhérents dans 1 800 blocs) : sans index, on recense 20 fois plus d'accès aux blocs et de nombreux tris.

**Tableau 14-34 Performances d'une extraction avec et sans index**

Avec index		Sans index	
<pre>SELECT a.civilite, a.prenom, a.nom, a.tel FROM Adherent a   WHERE NOT EXISTS     (SELECT spid FROM Sport      MINUS SELECT spid FROM Pratique WHERE adhid = a.adhid)   AND NOT EXISTS     (SELECT spid FROM Pratique WHERE adhid = a.adhid      MINUS SELECT spid FROM Sport);</pre>			
CIVILITE	PRENOM	NOM	TEL
-----			
Mme.	CELINE	LARRAZET	05-62-18-04-76
3 centièmes de secondes		36	secondes
114	recursive calls	533	recursive calls
72734	consistent gets	1442719	consistent gets
0	sorts (memory)	48080	sorts (memory)



Bien que les index *B-tree* soient majoritairement employés, ils ne conviennent pas aux conditions suivantes :

- Données de faible cardinalité : on considère qu'une colonne disposant de moins de 200 valeurs distinctes n'est pas une bonne candidate à un index *B-tree* (par exemple, la civilité qui ne comporte que 3 valeurs). Les index *bitmap* sont une alternative à cette limitation.
- Quand l'accès aux données s'effectue par une fonction SQL (*built-in function*), l'index *B-tree* n'est pas utilisé (par exemple `WHERE UPPER(prenom) = 'PAUL'` n'emploiera pas l'index sur `prenom`). Le fait de créer un index sur cette fonction est une alternative à cette limitation.

Avant la version 9i, l'optimiseur optait systématiquement pour un parcours entier de la table (*full scan*) si les requêtes contenaient des prédicats basés sur des expressions. Depuis, Oracle a répondu à ces problématiques à l'aide de mécanismes complémentaires avec les index *bitmap* et ceux basés sur des expressions ou des fonctions (*functions based index*).

### ***Index et expressions (built-in function)***

Si vous utilisez des fonctions caractères (`UPPER`, `SUBSTR`, `RTRIM`, etc.) ou des fonctions numériques (`MOD`, `ROUND`, `TRUNC`, etc.) dans le prédicat de vos requêtes, n'espérez pas utiliser vos index.

Le tableau 14-35 présente les résultats de différentes requêtes selon deux stratégies d'indexage. La volumétrie de la table `Adherentbis` est de plus d'un million d'adhérents (88 Mo de données occupant près de 90 000 blocs). Pour chaque requête, sont donnés : le type de parcours de l'index (*table access full* : l'index n'est pas utilisé), le nombre de blocs lus (*b*) et le coût (*c*).

Les remarques que l'on peut déduire à propos de la première stratégie d'indexage sont les suivantes :

- Les fonctions `ROUND` et `UPPER` rendent inopérants les index définis pourtant sur les colonnes concernées.
- Les index sur les colonnes numériques sont plus performants que les index sur les colonnes chaînes de caractères.

Concernant la deuxième stratégie d'indexage, les fonctions `ROUND` et `UPPER` rendent opérationnels les index, mais les conditions simples sur les colonnes entraînent un parcours entier de la table.

Tableau 14-35 Utilisation d'index B-tree sur des expressions de colonnes

Index existants	Prédicats et résultats	
CREATE INDEX idx_nom ON Adherentbis ( <b>nom</b> ) TABLESPACE tbs_index;	WHERE <b>nom</b> = 'DUCLOS' AND civilite='Mr.' AND tel LIKE '+33%'  <i>Index range scan</i> (578 b – 110 c)	WHERE UPPER( <b>nom</b> )='DUCLOS' AND civilite='Mr.' AND tel LIKE '+33%'  <i>Table access full</i> (10236 b – 2797 c)
CREATE INDEX idx_solde ON Adherentbis ( <b>solde</b> ) TABLESPACE tbs_index;	WHERE ROUND( <b>solde</b> ,1)=9030.8  <i>Table access full</i> (10235 b – 2802 c)	WHERE <b>solde</b> =9030.75  <i>Index range scan</i> (3 b – 3 c)
CREATE INDEX idx_UPPERnom ON Adherentbis ( <b>UPPER(nom)</b> ) TABLESPACE tbs_index;	WHERE <b>nom</b> = 'DUCLOS' AND civilite='Mr.' AND tel LIKE '+33%'  <i>Table access full</i> (10229 b – 2795 c)	WHERE <b>UPPER(nom)</b> = 'DUCLOS' AND civilite='Mr.' AND tel LIKE '+33%'  <i>Index range scan</i> (578 b – 107 c)
CREATE INDEX idx_ROUNDsolde ON Adherentbis ( <b>ROUND(solde,1)</b> ) TABLESPACE tbs_index;	WHERE <b>ROUND(solde,1)</b> =9030.8  <i>Index range scan</i> (3 b – 3 c)	WHERE <b>solde</b> =9030.75  <i>Table access full</i> (10229 b – 2795 c)

### Index et valeurs NULL

Le principe de fonctionnement des index *B-tree* ne permet pas une recherche directe (*unique scan*) sur une absence de valeur (NULL) ; en conséquence, si un index existe sur une colonne non nulle, il ne sera pas utilisé au mieux lors de la recherche des NULL (prédicat IS NULL ou IS NOT NULL).

La valeur NULL n'est pas à éviter à tout prix, elle est parfois bien utile pour exprimer « je ne sais pas encore » ou « sans objet ». Il est aussi pratique de vouloir indexer une colonne qui contiendra des valeurs NULL. Plusieurs solutions existent, elles sont basées sur la création d'un index :

- Sur une fonction déterministe qui retourne un entier quand la colonne est nulle.
- Composé dont une colonne n'est jamais nulle.
- Basé sur la fonction NVL2(chaine, valeur\_si\_NOT\_null, valeur\_si\_null), qui retourne une des deux valeurs suivant que chaine est nulle ou non.

Appliquez ces différentes solutions à votre base de sorte à déterminer la plus performante. Le tableau suivant présente quelques résultats d'après la recherche du nombre d'adhérents en fonction de leur numéro de téléphone donné ( NULL, valeur, NOT NULL). Concernant les données, 37 485 adhérents n'ont pas de numéro de téléphone (soit 3 % de la population). Pour

chaque type d'indexage, sont donnés la taille de l'index en Mo, le type de parcours de l'index, le nombre de blocs lus (*b*) et le coût (*c*).

**Tableau 14-36 Utilisation d'index B-tree sur une colonne ayant des valeurs NULL**

SELECT COUNT(nom) FROM Adherent WHERE...	Condition sur la nullité	tel='06-81-94- 44-31'	tel IS NOT NULL
Sans index	tel IS NULL <i>Table access full</i> (10228 <i>b</i> – 2793 <i>c</i> )	<i>Table access full</i> (10228 <i>b</i> – 2796 <i>c</i> )	<i>Table access full</i> (10228 <i>b</i> – 2793 <i>c</i> )
Index <i>B-tree</i> (taille : 38 Mo) <b>CREATE INDEX idx_tel_btree ON Adherent (tel);</b>	tel IS NULL <i>Table access full</i> (10228 <i>b</i> – 2793 <i>c</i> )	<i>Index range scan</i> (4 <i>b</i> – 5 <i>c</i> )	<i>Index fast full scan</i> (4756 <i>b</i> – 1293 <i>c</i> )
Index fonction (taille : 0,68 Mo) CREATE FUNCTION <b>f_tel_null</b> (p_tel Adherentbis.tel%type) RETURN NUMBER <b>DETERMINISTIC</b> AS BEGIN IF p_tel IS NULL THEN RETURN 1; ELSE RETURN NULL; END IF; END f_tel_null; <b>CREATE INDEX idx_tel_btree ON Adherent (f_tel_null(tel));</b>	f_tel_null(tel)=1 <i>Index fast full scan</i> (28 <i>b</i> – 84 <i>c</i> )	<i>Table access full</i> (10228 <i>b</i> – 2796 <i>c</i> )	Sans objet
Index composé (taille : 41 Mo) <b>CREATE INDEX idx_tel_btree ON Adherent (tel,0);</b>	tel IS NULL <i>Index range scan</i> (166 <i>b</i> – 76 <i>c</i> )	<i>Index range scan</i> (4 <i>b</i> – 5 <i>c</i> )	<i>Index fast full scan</i> (5150 <i>b</i> – 1399 <i>c</i> )
Index fonction NVL2 (taille : 0,62 Mo) CREATE INDEX idx_tel_btree ON Adherent (NVL2(tel,NULL,0));	NVL2(tel,NULL,0)=0 <i>Index fast full scan</i> (74 <i>b</i> – 21 <i>c</i> )	<i>Table access full</i> (10228 <i>b</i> – 2796 <i>c</i> )	Sans objet
Index unique (taille : 37 Mo) <b>CREATE UNIQUE INDEX idx_tel_btree ON Adherentbis (tel);</b>	tel IS NULL <i>Table access full</i> (10228 <i>b</i> – 2793 <i>c</i> )	<i>Index unique scan</i> (4 <i>b</i> – 3 <i>c</i> )	<i>Index fast full scan</i> (4599 <i>b</i> – 1250 <i>c</i> )

Les index les plus performants sont :

- Pour répondre au prédicat `IS NULL`, ceux qui utilisent une fonction (déterministe pour l'un et `NVL2` pour l'autre).
- Pour répondre au prédicat `col=valeur`, l'index classique, unique ou composé, qui offre les mêmes résultats.
- Pour répondre au prédicat `IS NOT NULL`, l'index unique.