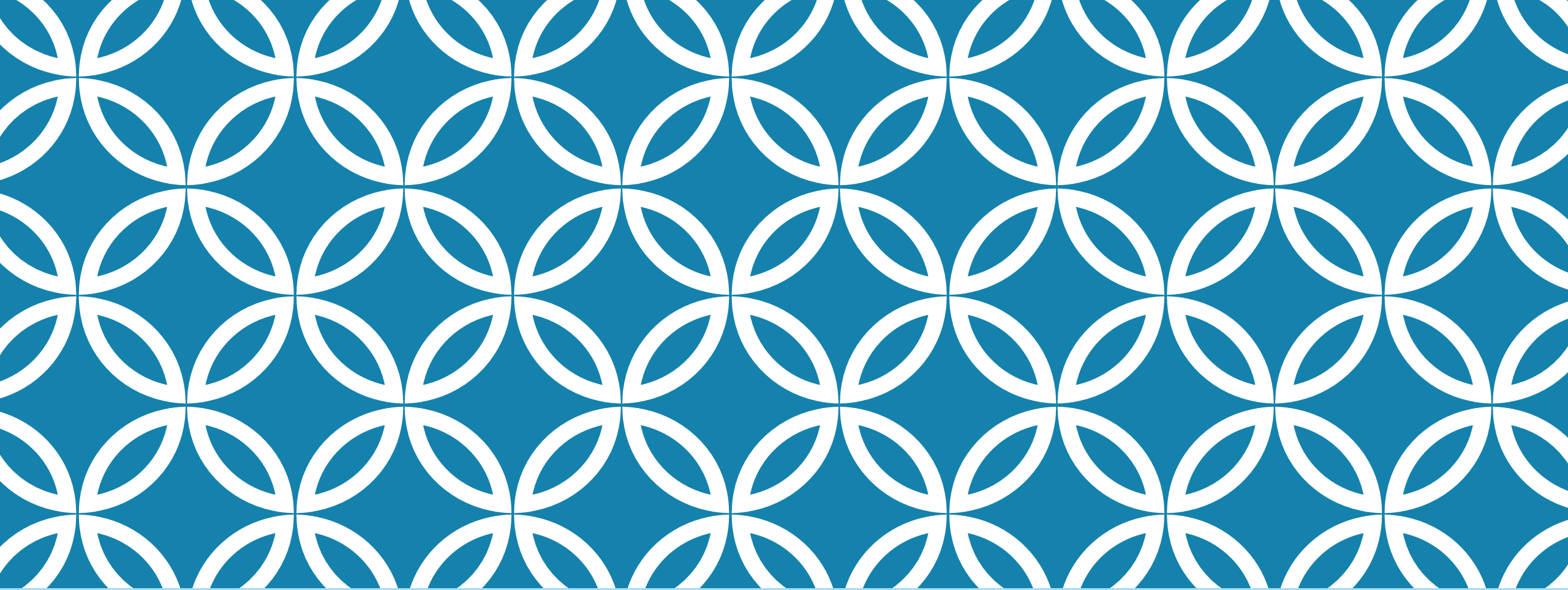


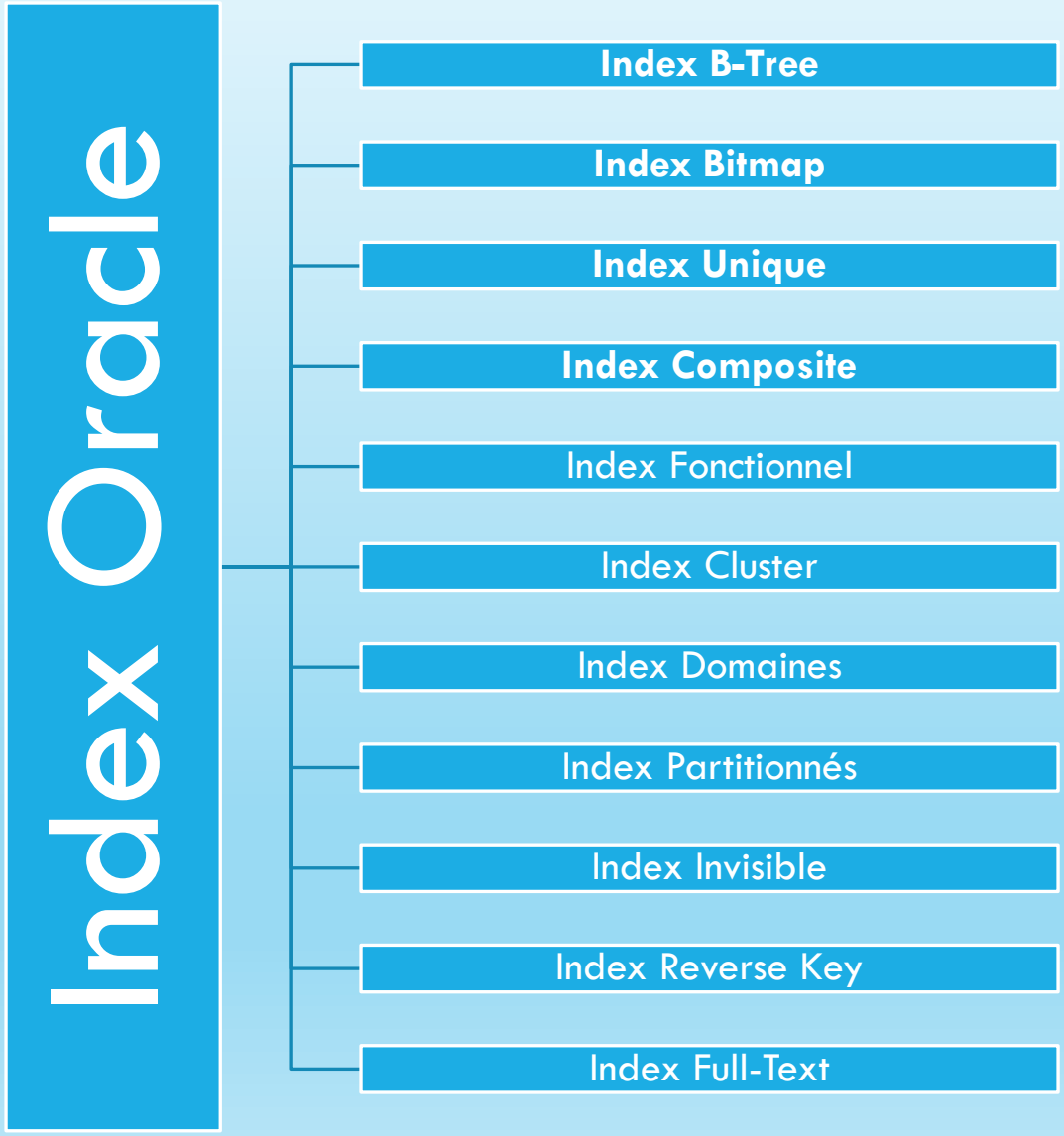
TYPES D'INDEX EN ORACLE

Par Pr Boubker Sbihi

0. LES INDEX EN ORACLE

- ✓En Oracle, les index sont des structures de données utilisées pour améliorer les performances des requêtes en permettant un accès plus rapide aux données dans une table
- ✓Oracle offre une variété d'index pour optimiser les performances des requêtes en fonction des cas d'utilisation.
- ✓Les index B-tree sont les plus courants, mais des index spécifiques comme le Bitmap, Reverse Key, ou Basé sur Fonction peuvent être utilisés pour des besoins particuliers, comme dans les bases de données de grande taille ou avec des types de données spécifiques.

0. LES INDEX EN ORACLE



1. INDEX B-TREE (BALANCED TREE)

- ✓Le type d'index B-Tree est le type par défaut.
- ✓Il organise les données dans une structure d'arbre équilibrée pour une recherche rapide.
- ✓Il est utilisé pour des colonnes avec une grande variété de valeurs (cardinalité élevée).
- ✓Il est efficace pour les opérations d'égalité (=) et de plage (BETWEEN, <, >).

1. INDEX B-TREE (PAR DÉFAUT)

✓Ce type d'index est créé par défaut lorsqu'on utilise un index standard.

```
CREATE INDEX index_nom ON table_nom (colonne_nom);
```

✓Exemple : Index sur une colonne nom dans une table employes.

```
CREATE INDEX idx_employes_nom ON employes(nom);
```

2. INDEX BITMAP

- ✓ Il utilise des bits pour représenter les valeurs et leur occurrence.
- ✓Il est compact et bien adapté pour des colonnes avec une faible cardinalité (peu de valeurs distinctes).
- ✓Il est utilisé pour des colonnes de faible cardinalité (par ex. sexe, état civil).
- ✓Pour des requêtes analytiques ou lorsque plusieurs colonnes sont utilisées dans des conditions combinées.
- ✓Il est moins adapté pour des les données sont fréquemment mises à jour.

2. INDEX BITMAP

✓Utilisé pour les colonnes avec peu de valeurs distinctes (faible cardinalité).

```
CREATE BITMAP INDEX index_nom ON table_nom (colonne_nom);
```

✓**Exemple** : Index bitmap sur une colonne etat_civil (faible cardinalité).

```
CREATE BITMAP INDEX idx_employes_etat_civil ON employes(etat_civil);
```

3. INDEX UNIQUE

- ✓Ce type d’index garantit que chaque valeur dans la ou les colonnes indexées est unique.
- ✓Il est souvent utilisé pour des colonnes avec des contraintes d'unicité (par ex. PRIMARY KEY, UNIQUE).

3. INDEX UNIQUE

✓Assure que les valeurs de la colonne ou des colonnes indexées sont uniques.

```
CREATE UNIQUE INDEX index_nom ON table_nom (colonne_nom);
```

✓Exemple : Index unique sur une colonne email dans une table utilisateurs.

```
CREATE UNIQUE INDEX idx_utilisateurs_email ON utilisateurs(email);
```

4. INDEX COMPOSITE

- ✓C’est un index sur plusieurs colonnes.
- ✓Il est souvent utilisé lorsqu'une requête utilise souvent plusieurs colonnes dans les clauses WHERE.
- ✓Ou dans l’ordre des colonnes dans l'index est obligatoire
- ✓Il doit correspondre à l'ordre des colonnes dans les requêtes.

4. INDEX COMPOSITE

✓Code pour indexer plusieurs colonnes.

```
CREATE INDEX index_nom ON table_nom (colonne1, colonne2);
```

✓Exemple : Index sur les colonnes prenom et nom dans une table clients.

```
CREATE INDEX idx_clients_prenom_nom ON clients(prenom, nom);
```

5. INDEX FONCTIONNEL (FUNCTION-BASED INDEX)

- ✓Il est basé sur une expression ou une fonction appliquée à une colonne.
- ✓Il est utilisé pour les requêtes qui utilisent des expressions ou des fonctions dans leurs conditions (par ex. UPPER(nom) ou TRUNC(date)).

5. INDEX FONCTIONNEL

✓ il est utilisé pour indexer une fonction ou une expression appliquée à une colonne.

```
CREATE INDEX index_nom ON table_nom (UPPER(colonne_nom));
```

Comme exemple on peut citer un Index basé sur la fonction UPPER() appliquée à une colonne nom.

```
CREATE INDEX idx_employes_nom_upper ON employes(UPPER(nom));
```

Utilisation :

```
SELECT * FROM employes WHERE UPPER(nom) = 'DUPONT';
```

6. INDEX CLUSTER

- ✓Il est utilisé avec des tables en cluster, où plusieurs tables partagent une structure physique basée sur des colonnes communes.
- ✓Il sert pour l'optimisation des jointures fréquentes entre des tables en cluster.
- ✓Un index cluster est généralement utilisé dans des tables organisées dans un cluster, mais Oracle ne permet pas directement de créer un index cluster avec une syntaxe CREATE INDEX comme pour les autres types.

6. INDEX CLUSTER

✓Un cluster est configuré lors de la création des tables.

```
CREATE CLUSTER cluster_nom (colonne_nom);
```

6. INDEX CLUSTER

Comme exemple on peut citer la création d'un cluster pour des tables departements et employes, puis un index.

```
-- Création du cluster
CREATE CLUSTER emp_dept_cluster(dept_id NUMBER);
-- Ajout des tables au cluster
CREATE TABLE departements (dept_id NUMBER, nom VARCHAR2(50))
CLUSTER emp_dept_cluster(dept_id);
CREATE TABLE employes (emp_id NUMBER, nom VARCHAR2(50), dept_id NUMBER)
CLUSTER emp_dept_cluster(dept_id);
-- Création de l'index sur le cluster
CREATE INDEX idx_emp_dept_cluster ON CLUSTER emp_dept_cluster;
```

7. INDEX DOMAINES

- ✓C’est des Index définis par l'utilisateur pour des types de données spécifiques, comme les données spatiales, multimédias, ou textuelles.
- ✓Ils est utilisé avec Oracle Text ou Oracle Spatial.

7. INDEX DOMAINE

- ✓Utilisé pour indexer des données géographiques ou spatiales. Oracle offre une fonctionnalité avancée d'indexation spatiale.
- ✓CREATE INDEX index_nom ON table_nom (colonne_spatiale) INDEXTYPE IS MDSYS.SPATIAL_INDEX;
- ✓Exemple : Index pour des recherches textuelles sur une colonne description.
- ✓CREATE INDEX idx_articles_description ON articles(description)
- ✓ INDEXTYPE IS CTXSYS.CONTEXT;

8. INDEX GLOBAL PARTITIONED ET LOCAL PARTITIONED

L'index Global Partitioned est relatif aux données qui sont divisées en partitions basées sur un critère global.

Tandis que l'index local Partitioned est relatif à chaque partition qui correspond directement à une partition de la table.

Ils sont utilisés pour l'amélioration des performances sur des tables partitionnées pour des grandes bases de données.

8. INDEX PARTITIONNÉS

Exemple : Index global et local sur une table partitionnée.

```
-- Table partitionnée
CREATE TABLE ventes ( vente_id NUMBER, region VARCHAR2(50), montant NUMBER)
PARTITION BY LIST (region) (
    PARTITION region_nord VALUES ('Nord'),
    PARTITION region_sud VALUES ('Sud'));
-- Index global
CREATE INDEX idx_ventes_global  ON ventes(montant);
-- Index local
CREATE INDEX idx_ventes_local ON ventes(region)
LOCAL;
```

9. INDEX INVISIBLE

- ✓Un index invisible n'est pas utilisé par défaut par l'optimiseur, sauf si explicitement indiqué.
- ✓Il est utiliser pour tester l'impact d'un index sans affecter immédiatement les performances globales.

9. INDEX INVISIBLE

Exemple : Création d'un index invisible sur une colonne date_creation.

CREATE INDEX idx_commandes_date_creation ON commandes(date_creation)

INVISIBLE;

Utilisation :

ALTER SESSION SET optimizer_use_invisible_indexes = TRUE;

SELECT * FROM commandes WHERE date_creation = SYSDATE;

10. INDEX REVERSE KEY

Dans ce type d’index, les valeurs des colonnes sont inversées avant d'être stockées dans l'index.

Il est utilisé pour répartir les insertions uniformément sur les blocs d'index ou encore pour les colonnes générées séquentiellement (par ex. séquences).

10. INDEX REVERSE KEY

Ce type d'index inverse l'ordre des clés afin d'éviter les "points chauds" d'insertion.

CREATE INDEX index_nom ON table_nom (REVERSE colonne_nom);

Exemple : Index avec clé inversée sur une colonne id_commande.

CREATE INDEX idx_commandes_reverse ON commandes(id_commande) REVERSE;

11. INDEX FULL-TEXT

Un full scan d'index est généralement réalisé automatiquement par Oracle lorsqu'aucune autre méthode de récupération n'est optimale, donc il n'y a pas de création d'index spécifique pour cela.

Ce comportement dépend de la requête et de la taille de l'index.

11. INDEX FULL-TEXT (ORACLE TEXT)

Cet index est conçu pour effectuer des recherches sur de grandes quantités de texte non structuré.

Il est utilisé pour des colonnes contenant des données textuelles volumineuses (par ex. documents, descriptions).

Chaque type d'index a ses cas d'utilisation spécifiques. Le choix dépend des caractéristiques des données et des requêtes exécutées sur la base de données.

Voici des exemples de code SQL pour chaque type d'index dans Oracle. Ces exemples illustrent leur création et leur utilisation.

11. INDEX FULL-TEXT (ORACLE TEXT)

Exemple : Création d'un index pour des recherches avancées sur une colonne textuelle.

```
CREATE INDEX idx_documents_contenu ON documents(contenu)
   INDEXTYPE IS CTXSYS.CONTEXT;
```

Utilisation :

```
SELECT * FROM documents WHERE CONTAINS(contenu, 'oracle') > 0;
```

Ces exemples montrent comment créer et utiliser chaque type d'index. Adaptez-les à votre contexte spécifique selon les besoins de votre base de données.

12. INDEX CODE SQL

Exemple d'index unique avec une table :

```
CREATE TABLE utilisateurs (
    id NUMBER PRIMARY KEY,
    nom VARCHAR2(100),
    email VARCHAR2(100) UNIQUE
);
-- Création de l'index unique sur la colonne "email"
CREATE UNIQUE INDEX idx_email ON utilisateurs(email);
```

12. INDEX SCAN

Table Access by ROWID : Très faible coût car il s'agit d'un accès direct aux lignes via leur ROWID, sans avoir besoin de parcourir les index.

Index Full Scan : Coût élevé, car il nécessite de lire tout l'index, ce qui est coûteux en termes de ressources, surtout pour de grands index.

Index Fast Full Scan : Coût moyen, plus rapide que le full scan classique car il ne nécessite pas d'accès aux données de la table, mais reste coûteux pour de grands ensembles de données.

Index Unique Scan : Très faible coût pour l'accès rapide à une seule ligne grâce à un index unique.

12. INDEX

Index Range Scan : Coût moyen : bien que ce type de scan soit optimisé pour les plages de données, il peut nécessiter de scanner plusieurs valeurs et est plus coûteux que l'accès unique.

En général, les access by ROWID et les index unique scan sont les moins coûteux en termes de performance,

Tandis que les full scans (qu'ils soient classiques ou rapides) peuvent devenir coûteux à grande échelle.

Le range scan offre un bon compromis pour les requêtes sur des plages de données, mais son coût dépend de la taille de la plage et de l'index.

TYPES DE SCANS D'INDEX

Type de Scan / Accès	Structure et Fonctionnement	Caractéristiques	Cas d'usage	Coût
Table Access by ROWID	Accède directement aux lignes en utilisant leur identifiant ROWID.	Très rapide car il permet d'atteindre une ligne spécifique sans recherche dans l'index.	Requêtes après recherche d'une ligne spécifique via un index ou une jointure.	Très faible : accès direct aux lignes.
Index Full Scan	Scanne l'ensemble de l'index pour lire toutes les valeurs indexées.	Plus coûteux, car il examine chaque entrée de l'index. Utilisé si toutes les lignes de l'index doivent être lues.	Requêtes qui nécessitent d'examiner toutes les lignes d'un index.	Élevé : Scanne tout l'index sans distinction.
Index Fast Full Scan	Accède directement aux données de l'index, sans besoin d'accès à la table.	Plus rapide qu'un full scan car il évite l'accès aux données de la table, en utilisant directement l'index pour tout récupérer.	Utilisé pour récupérer toutes les données d'un index sans avoir besoin de consulter la table.	Moyen : plus rapide qu'un full scan, mais peut être coûteux pour de grands index.
Index Unique Scan	Scanne l'index pour récupérer une seule ligne unique via l'index.	Très rapide, utilisé pour des clés primaires ou uniques.	Requêtes avec des clés primaires ou uniques qui retournent une seule ligne.	Très faible : accès direct à une ligne unique.
Index Range Scan	Scanne une plage de valeurs dans l'index.	Utilisé pour les requêtes avec des conditions de plage, comme BETWEEN, >=, <.	Requêtes qui filtrent sur des plages de données avec des opérateurs de comparaison.	Moyen : efficace pour les plages, mais peut être plus coûteux que les scans uniques.

TYPES DE SCANS D'INDEX

Type d'Index	Structure	Caractéristiques	Cas d'Usage
Index B-tree	Arbre binaire équilibré (B-tree)	Index par défaut, efficace pour les requêtes d'égalité et de plages (par exemple, =, <, >, BETWEEN)	Idéal pour les clés primaires, les contraintes uniques, et les colonnes fréquemment interrogées avec une large gamme de valeurs.
Index Bitmap	Tableaux de bits représentant chaque valeur unique du champ indexé	Compact et efficace pour les colonnes avec une faible cardinalité (peu de valeurs distinctes)	Utilisé dans les entrepôts de données pour des colonnes à faible cardinalité, comme les statuts ou les sexes, souvent avec des requêtes complexes.
Index Cluster	Données regroupées selon la clé d'index (non pris en charge directement par Oracle)	Améliore la récupération des données lorsque plusieurs lignes sont souvent récupérées ensemble	Lorsque l'ordre physique des données est important et dans les requêtes sur des plages de valeurs (à travers des tables indexées organisées).
Index Composite	Plusieurs colonnes combinées dans une seule clé d'index	Permet de traiter les requêtes sur plusieurs colonnes, l'ordre des colonnes dans l'index doit correspondre à celui des requêtes	Requêtes filtrant sur plusieurs colonnes dans un ordre spécifique, comme les filtres combinés sur nom et prénom.
Index Basé sur Fonction	Index basé sur des expressions ou des fonctions appliquées aux colonnes	Permet d'indexer des valeurs dérivées ou transformées d'une colonne (par exemple, UPPER(colonne))	Idéal pour les requêtes impliquant des expressions sur les colonnes, comme la recherche insensible à la casse sur des champs de texte.
Index de Domaine	Index pour des types de données personnalisés	Utilisé pour les types de données non standards, comme les données spatiales ou multimédia.	Utilisé dans des applications spécialisées, par exemple pour l'indexation de données géographiques ou de contenu XML.
Index Reverse Key	Valeurs de la clé indexée inversées	Prévention des "points chauds" dans les insertions rapides, notamment pour les clés de séquence.	Utile dans les environnements où les clés de séquence sont utilisées, pour éviter l'accumulation d'insertions dans les mêmes blocs d'index.
Index Bitmap Join	Combinaison de plusieurs indices bitmap pour optimiser les jointures	Optimise les jointures sur des colonnes de faible cardinalité, stockant le résultat des jointures dans l'index.	Idéal pour les requêtes de jointure fréquentes entre tables dans les entrepôts de données avec des colonnes à faible cardinalité.