MODULO III
SESION 03

Ing. César Hijar

chijar@galaxy.edu.pe

GALAXY
TRAINING

AGENDA

GESTION DE INDICES

a) Concepto de índices.
b) Tipos y comparación de índices.
c) Creación, recreación y movimiento de índices.
d) Eliminación y monitoreo de índices.
e) Mejores prácticas para el uso de índices.

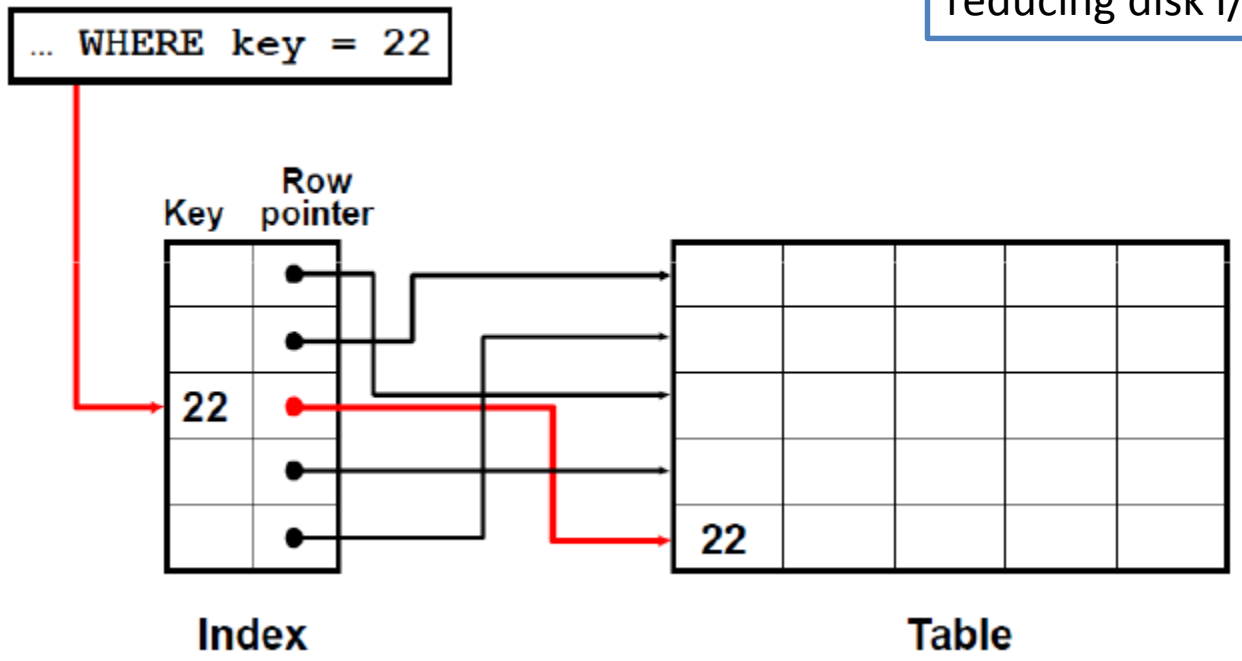
DCL Y CONTROL DE TRANSACCIONES

a) Grant y Revoke.
b) Tipos de privilegios a asignar o revocar.
c) Control de transacciones
d) Uso del commit, rollback, y savepoint.

INDICES

Índices

An index is an optional structure, associated with a table or table cluster, that can sometimes speed data access.



reducing disk I/O.

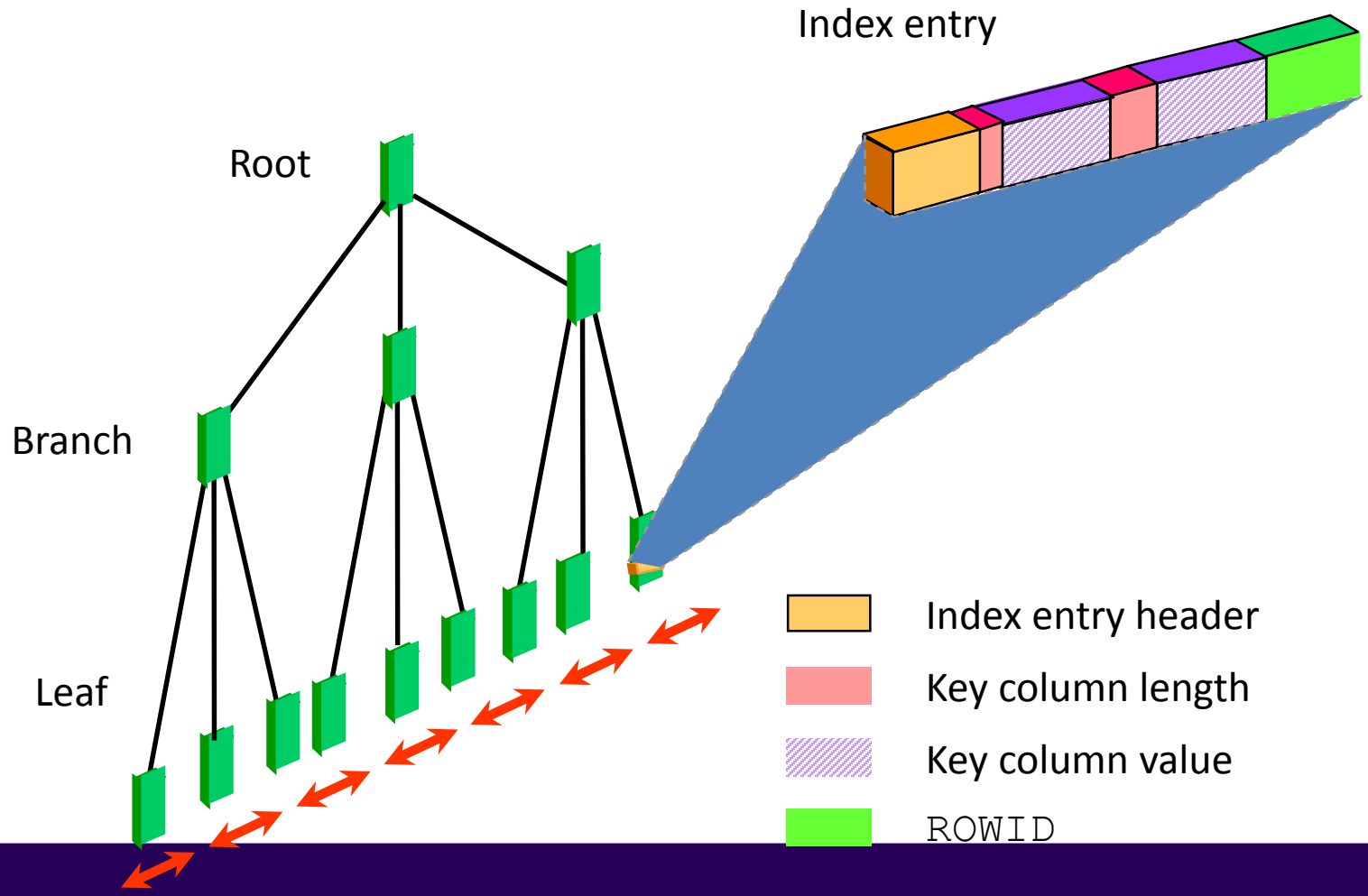
Tipos de índices

Oracle provee varios esquemas de indexado lo cual provee funcionalidad de rapidez complementaria.

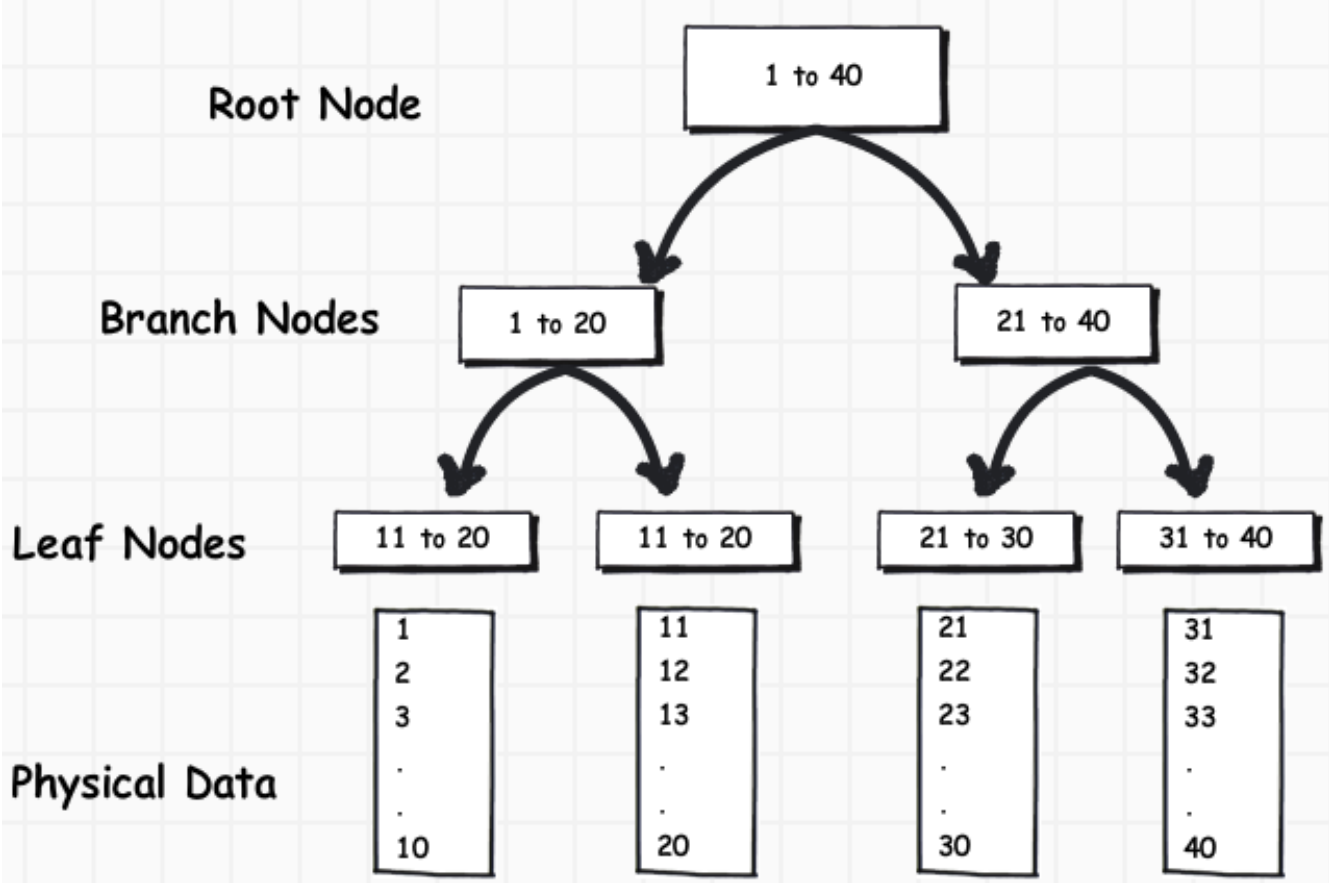
- B-Tree
- Bitmap
- Basados en función

Indice B-Tree

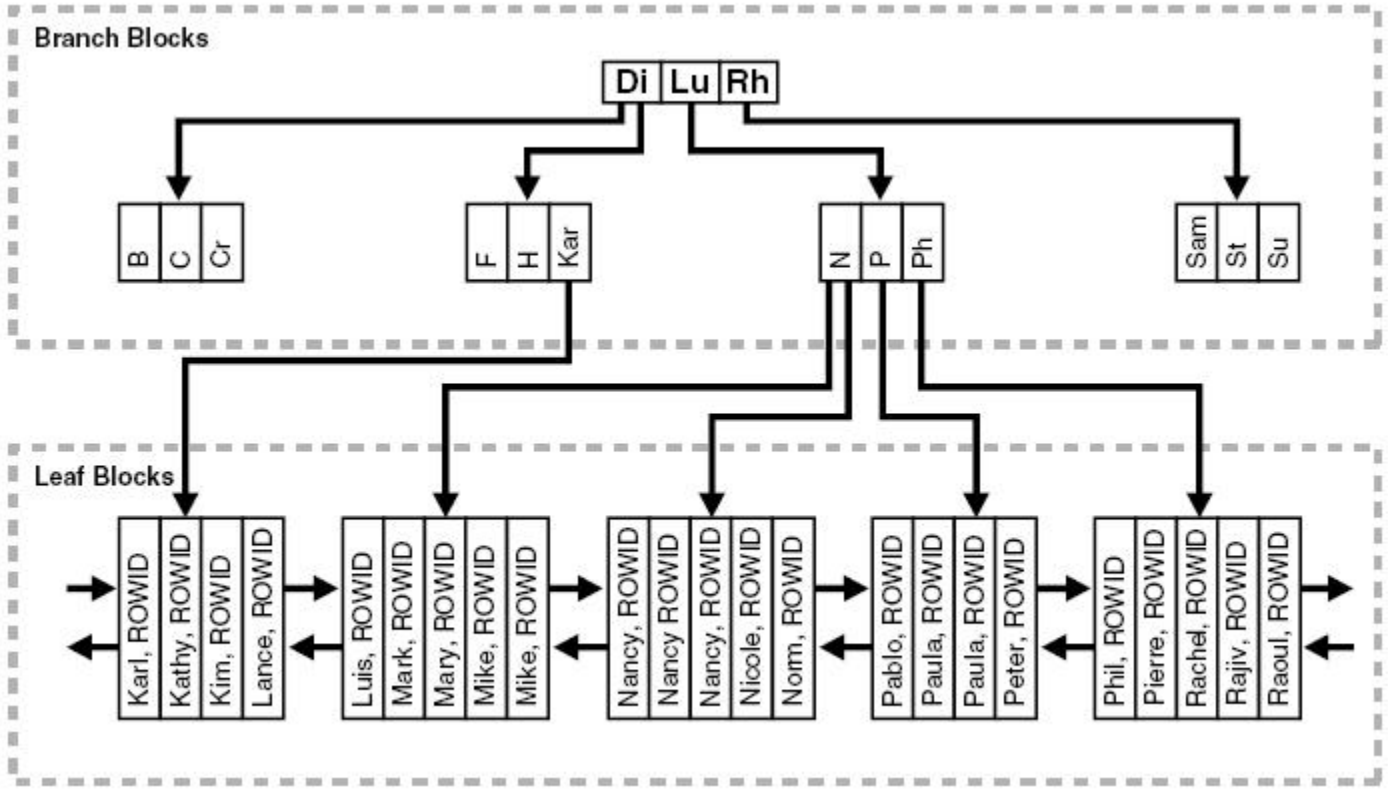
Although all the indexes use a B-tree structure, the term B-tree index is usually associated with an index that stores a list of ROWIDs for each key.



Ejemplo:

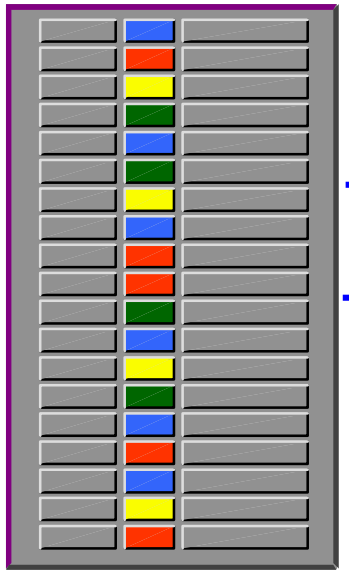

Root Node
1 to 40
Branch Nodes
1 to 20
21 to 40
Leaf Nodes
11 to 20
11 to 20
21 to 30
31 to 40
Physical Data
1
2
3
.
.
10
11
12
13
.
.
20
21
22
23
.
.
30
31
32
33
.
.
40

Ejemplo:


Branch Blocks
Di Lu Rh
B C Cr
F H Kar
N P Ph
Sam St Su
Leaf Blocks
Karl, ROWID
Kathy, ROWID
Kim, ROWID
Lance, ROWID
Luis, ROWID
Mark, ROWID
Mary, ROWID
Mike, ROWID
Mike, ROWID
Nancy, ROWID
Nancy ROWID
Nancy, ROWID
Nicole, ROWID
Norm, ROWID
Pablo, ROWID
Paula, ROWID
Paula, ROWID
Peter, ROWID
Phil, ROWID
Pierre, ROWID
Rachel, ROWID
Rajiv, ROWID
Raoul, ROWID

Indice Bitmap

Table



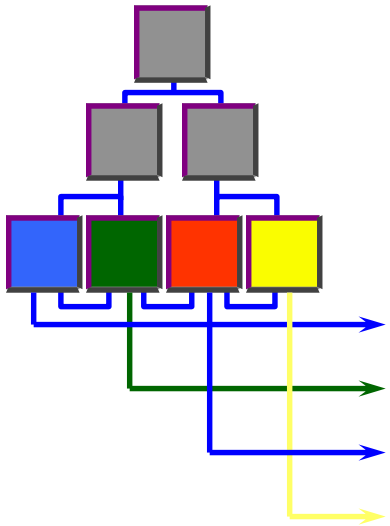
File 3

Block 10

Block 11

Block 12

Index



key	start ROWID	end ROWID	bitmap
<Blue,	10.0.3,	12.8.3,	10001001000100101000>
<Green,	10.0.3,	12.8.3,	00010100001001000000>
<Red,	10.0.3,	12.8.3,	01000000110000010001>
<Yellow,	10.0.3,	12.8.3,	00100010000010000010>

Ejemplo

Tabla CUSTOMER

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

Indice BITMAP

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

Comparando índices B-Tree y Bitmap

B-Tree

Recomendable columnas de alta cardinalidad
Actualizaciones sobre las columnas clave no son costosas
Ineficiente para consultas que utilizan el predicado OR
Usado en OLTP

Bitmap

Recomendable para columnas con baja cardinalidad
Actualización (update) sobre las columnas clave son costosas.
Efectivas para consultas que utilizan el predicado OR
Usado en Data Warehousing

Creando un índice

Usted está considerando crear índices sobre las columnas NAME y REGION de la tabla CUSTOMERS.
¿Qué tipo de índice es apropiado para estas dos columnas?

Sugerencia: Un índice B-Tree es adecuado para una columna con muchos valores distintos, y un índice Bitmap es adecuado para columnas con pocos valores distintos.

Creando un índice

Sintaxis:

```
CREATE [BITMAP] INDEX index_name
ON table (column)
TABLESPACE tablespace_name;
```

Si se desea crear un índice Bitmap se agregará la clausula BITMAP a la sentencia de creación del índice.
La clausula TABLESPACE es opcional y se utiliza para crear la índice en un tablespace diferente al asignado por defecto al usuario.

Ejemplos de creación de índices

Indice B-Tree

```
CREATE INDEX dept_name_idx
ON hr.departments (department_name)
TABLESPÀCE index01;
```

Indice BITMAP

```
CREATE BITMAP INDEX region_name_idx
ON hr.regions(region_name)
TABLESPÀCE index01;
```

Mover un índice a otro tablespace

Sintaxis:

```
ALTER INDEX index_name
REBUILD TABLESPÀCE tablespace_name;
```

Ejemplo:

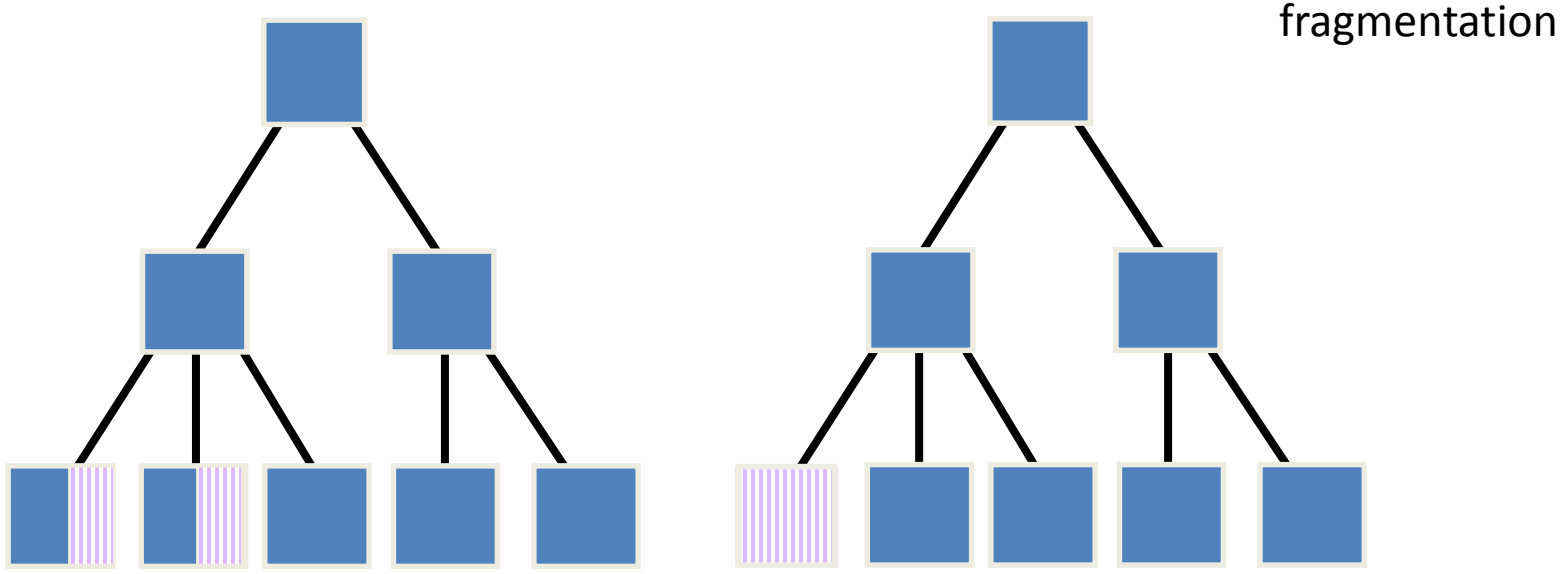
```
ALTER INDEX dept_name_idx
REBUILD TABLESPÀCE index02;
```

Recreando índices ONLINE

La tarea de recrear un índice puede tomar mucho tiempo si es sobre una tabla muy grande.
Cuando se recrea un índice, el mismo se bloquea y no permite ejecutar operaciones de DML.
Oracle provee una forma de recrear índices sin bloquear operaciones DML.

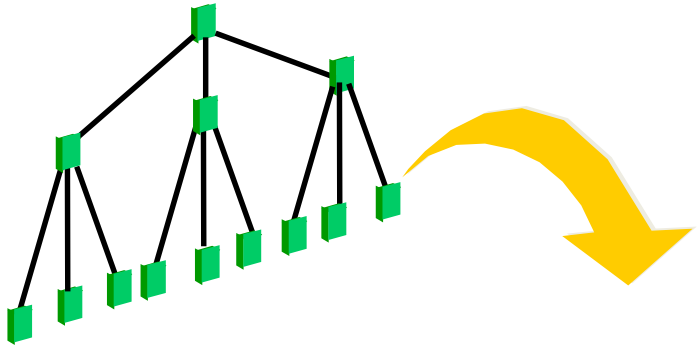
```
ALTER INDEX dept_name_idx REBUILD ONLINE;
```

Coalescing indices

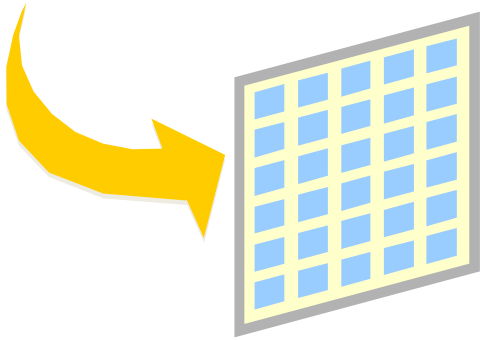



```
ALTER INDEX orders_id_idx COALESCE;
```

Verificando la validez de un índice



```
ANALYZE INDEX orders_region_id_idx
VALIDATE STRUCTURE;
```



INDEX_STATS

Eliminando un indice

```
SQL> DROP INDEX dept_name_idx;
Index dropped.

SQL> DROP INDEX region_name_idx;
Index dropped.
```

Identificando índices no usados

```
ALTER INDEX hr.dept_id_idx
MONITORING USAGE;
```

Para monitorear el uso de un índice:

```
ALTER INDEX hr.dept_id_idx
NOMONITORING USAGE;
```

Para detener el monitoreo del uso de un índice:

Esto llena la vista **V$OBJECT_USAGE**

Obteniendo información de los índices

Información acerca de los índices se puede encontrar consultando:

- DBA_INDEXES
- DBA_IND_COLUMNS
- V$OBJECT_USAGE

BEST PRACTICES - INDEXES

Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key.

Create Indexes After Inserting Table Data
Data is often inserted or loaded into a table using either the SQL*Loader or an import utility. It is more efficient to create an index for a table after inserting or loading the data. If you create one or more indexes before loading data, the database then must update every index as each row is inserted.

Creating an index on a table that already has data requires sort space. Some sort space comes from memory allocated for the index creator. The amount for each user is determined by the initialization parameter SORT_AREA_SIZE. The database also swaps sort information to and from temporary segments that are only allocated during the index creation in the users temporary tablespace.

BEST PRACTICES - INDEXES

Columns That Are Suitable for Indexing

Some columns are strong candidates for indexing. Columns with one or more of the following characteristics are candidates for indexing:

•Values are relatively unique in the column.

•There is a wide range of values (good for regular indexes).

•There is a small range of values (good for bitmap indexes).

Columns That Are Not Suitable for Indexing

Columns with the following characteristics are less suitable for indexing:

•Hay muchos nulos en la columna y no busca en los valores no nulos.

BEST PRACTICES - INDEXES

Order Index Columns for Performance
The order of columns in the CREATE INDEX statement can affect query performance. In general, specify the most frequently used columns first.

If you create a single index across columns to speed up queries that access, for example, col1, col2, and col3; then queries that access just col1, or that access just col1 and col2, are also speeded up. But a query that accessed just col2, just col3, or just col2 and col3 does not use the index.

Limit the Number of Indexes for Each Table
A table can have any number of indexes. However, the more indexes there are, the more overhead is incurred as the table is modified. Specifically, when rows are inserted or deleted, all indexes on the table must be updated as well. Also, when a column is updated, all indexes that contain the column must be updated.
For example, if a table is primarily read-only, having more indexes can be useful; but if a table is heavily updated, having fewer indexes could be preferable.

BEST PRACTICES - INDEXES

Drop Indexes That Are No Longer Required
Consider dropping an index if:
•It does not speed up queries. The table could be very small, or there could be many rows in the table but very few index entries.
•The queries in your applications do not use the index.
•The index must be dropped before being rebuilt

Specify the Tablespace for Each Index
Using different tablespaces (on different disks) for a table and its index produces better performance than storing the table and index in the same tablespace. Disk contention is reduced.

BEST PRACTICES - INDEXES

Consider Parallelizing Index Creation
You can parallelize index creation, much the same as you can parallelize table creation. Because multiple processes work together to create the index, the database can create the index more quickly than if a single server process created the index sequentially.
When creating an index in parallel, storage parameters are used separately by each query server process. Therefore, an index created with an INITIAL value of 5M and a parallel degree of 12 consumes at least 60M of storage during index creation.

Consider Creating Indexes with NOLOGGING
You can create an index and generate minimal redo log records by specifying NOLOGGING in the CREATE INDEX statement.

1. Space is saved in the redo log files.
2. The time it takes to create the index is decreased.
3. Performance improves for parallel creation of large indexes

BEST PRACTICES - INDEXES

Unusable indexes

One reason to make an index unusable is to improve bulk load performance. (Bulk loads go more quickly if the database does not need to maintain indexes when inserting rows.) Instead of dropping the index and later re-creating it, you can make the index unusable, and then rebuild it.

Invisible Indexes

An invisible index is ignored by the optimizer unless you explicitly set the OPTIMIZER_USE_INVISIBLE_INDEXES initialization parameter to TRUE at the session or system level. Unlike unusable indexes, an invisible index is maintained during DML statements.

Un índice *unusable* es una forma sutil de dejar un índice "fuera de combate" temporalmente en TODAS las sesiones. Un índice *invisible* en cambio es selectivo: podemos verlo (y usarlo) en una sesión mientras en el resto de sesiones no saben que existe. Están afectados por las DML que se le pueda hacer a su tabla de orígen , es decir, mientras estén **INVISIBLES**, siguen actualizándose

BEST PRACTICES - INDEXES

La clave de este índice invisible está en el testeo de consultas. Por ejemplo, podremos probar cómo se comporta un SELECT con determinado índice, sin que el ambiente en el cual lo estoy probando se entere. Creo un índice invisible, seteo el valor de OPTIMIZER_USE_INVISIBLE_INDEXES como true, y testeo mi aplicación sin efectos colaterales.

Otro beneficio, razonando en forma opuesta: ¿Cómo funcionaría mi aplicación si determinado índice no existiera?
Alteramos el índice como 'invisible', seteamos en nuestra sesión OPTIMIZER_USE_INVISIBLE_INDEXES = false, y probamos. Los demás usuarios mientras tanto, continuarán utilizándolo normalmente.

BEST PRACTICES - INDEXES

Creating an Index Explicitly
You can create indexes explicitly (outside of integrity constraints) using the SQL statement CREATE INDEX. The following statement creates an index named emp_ename for the ename column of the emp table:

```
CREATE INDEX emp_ename ON emp(ename)
    TABLESPACE users
    STORAGE (INITIAL 20K
    NEXT 20k);
```

Creating a Unique Index Explicitly
Indexes can be unique or non-unique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Non-unique indexes do not impose this restriction on the column values.

Use the CREATE UNIQUE INDEX statement to create a unique index. The following example creates a unique index:

```
CREATE UNIQUE INDEX dept_unique_index ON dept
(dname)
    TABLESPACE indx;
```

BEST PRACTICES - INDEXES

Specifying Storage Options for an Index Associated with a Constraint
You can set the storage options for the indexes associated with UNIQUE and PRIMARY KEY constraints using the USING INDEX clause. The following CREATE TABLE statement enables a PRIMARY KEY constraint and specifies the storage options of the associated index:

```
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY, age INTEGER)
    ENABLE PRIMARY KEY USING INDEX
    TABLESPACE users;
```

BEST PRACTICES - INDEXES

Specifying the Index Associated with a Constraint
If you require more explicit control over the indexes associated with UNIQUE and PRIMARY KEY constraints, the database lets you:
These options are specified using the USING INDEX clause. The following statements present some examples.

Example 1:

```
CREATE TABLE a (
    a1 INT PRIMARY KEY USING INDEX (create index
ai on a (a1)));
```

Example 2:

```
CREATE TABLE b(
    b1 INT,
    b2 INT,
    CONSTRAINT bu1 UNIQUE (b1, b2)
                USING INDEX (create unique index bi on b(b1,
b2)),
    CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
```

Example 3:

```
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1)
USING INDEX ci;
```

BEST PRACTICES - INDEXES

Monitoring Index Usage
Oracle Database provides a means of monitoring indexes to determine whether they are being used. If an index is not being used, then it can be dropped, eliminating unnecessary statement overhead.

To start monitoring the usage of an index, issue this statement:
ALTER INDEX index MONITORING USAGE;
Later, issue the following statement to stop the monitoring:
ALTER INDEX index NOMONITORING USAGE;

The view V$OBJECT_USAGE can be queried for the index being monitored to see if the index has been used. The view contains a USED column whose value is YES or NO, depending upon if the index has been used within the time period being monitored. The view also contains the start and stop times of the monitoring period, and a MONITORING column (YES/NO) to indicate if usage monitoring is currently active.

ACCIONES REVISADAS EN EL USO DE INDICES

- Crear índices después de cargar datos.
- Columnas candidatas
- Columnas NO candidatas
- Orden de las columnas.
- Limitar el nro. De índices por tabla.
- Eliminar índices que ya no se usan.
- Especificar el tablespace en la creación del índice

ACCIONES REVISADAS EN EL USO DE INDICES

- Considerar paralelismo en la creación de índices.
- Considerar la creación con NOLOGGING
- UNUSABLE index
- INVISIBLE index
- MONITORING index

Resumen

1. Un objeto en Oracle debe empezar siempre con un carácter y tener un nombre descriptivo de lo que almacenara.
2. Los índices nos permiten agilizar los tiempos de respuesta de una consulta.
3. NO asigne índices para todas las columnas ya que pueden afectar las operaciones DML.
4. Genere índices según la cantidad de información diferente que pueda tener una columna de la tabla.

Información adicional

Managing Tables:
http://download.oracle.com/docs/cd/B19306_01/server.102/b14231/tables.htm

Managing Indexes:
http://download.oracle.com/docs/cd/B19306_01/server.102/b14196/schema003.htm

Los índices en Oracle, creación, eliminación y reconstrucción:
http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=449

DCL Y CONTROL DE TRANSACCIONES

DATA CONTROL LANGUAGE

SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control

DCL - Data Control Language

- **GRANT**
- **REVOKE**

Otorga y remueve derechos de acceso a objetos de la base de datos.

Sentencia GRANT

- The `GRANT` statement assigns the privilege to perform the following operations:
  - Insert or delete data.
  - Create a foreign key reference to the named table or to a subset of columns from a table.
  - Select data, a view, or a subset of columns from a table.
  - Create a trigger on a table.
  - Execute a specified function or procedure.
- Example:

```
GRANT SELECT any table to PUBLIC;
```

Tipos de privilegios a asignar a través de la sentencia GRANT

Assign the following privileges by using the `GRANT` statement:

- `ALL PRIVILEGES`
- `DELETE`
- `INSERT`
- `REFERENCES`
- `SELECT`
- `UPDATE`

Sentencia REVOKE

- Use the `REVOKE` statement to remove privileges from a user to perform actions on database objects.
- Revoke a *system privilege* from a user:

```
REVOKE DROP ANY TABLE
FROM hr;
```

- Revoke a *role* from a user:

```
REVOKE dw_manager
FROM sh;
```

Control de transacciones

CONTROL DE TRANSACCIONES

SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control

Transacciones

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement

Transacciones

- Begin when the first DML SQL statement is executed.
- End with one of the following events:
  - A `COMMIT` or `ROLLBACK` statement is issued.
  - A DDL or DCL statement executes (automatic commit).
  - The user exits SQL Developer or SQL*Plus.
  - The system crashes.

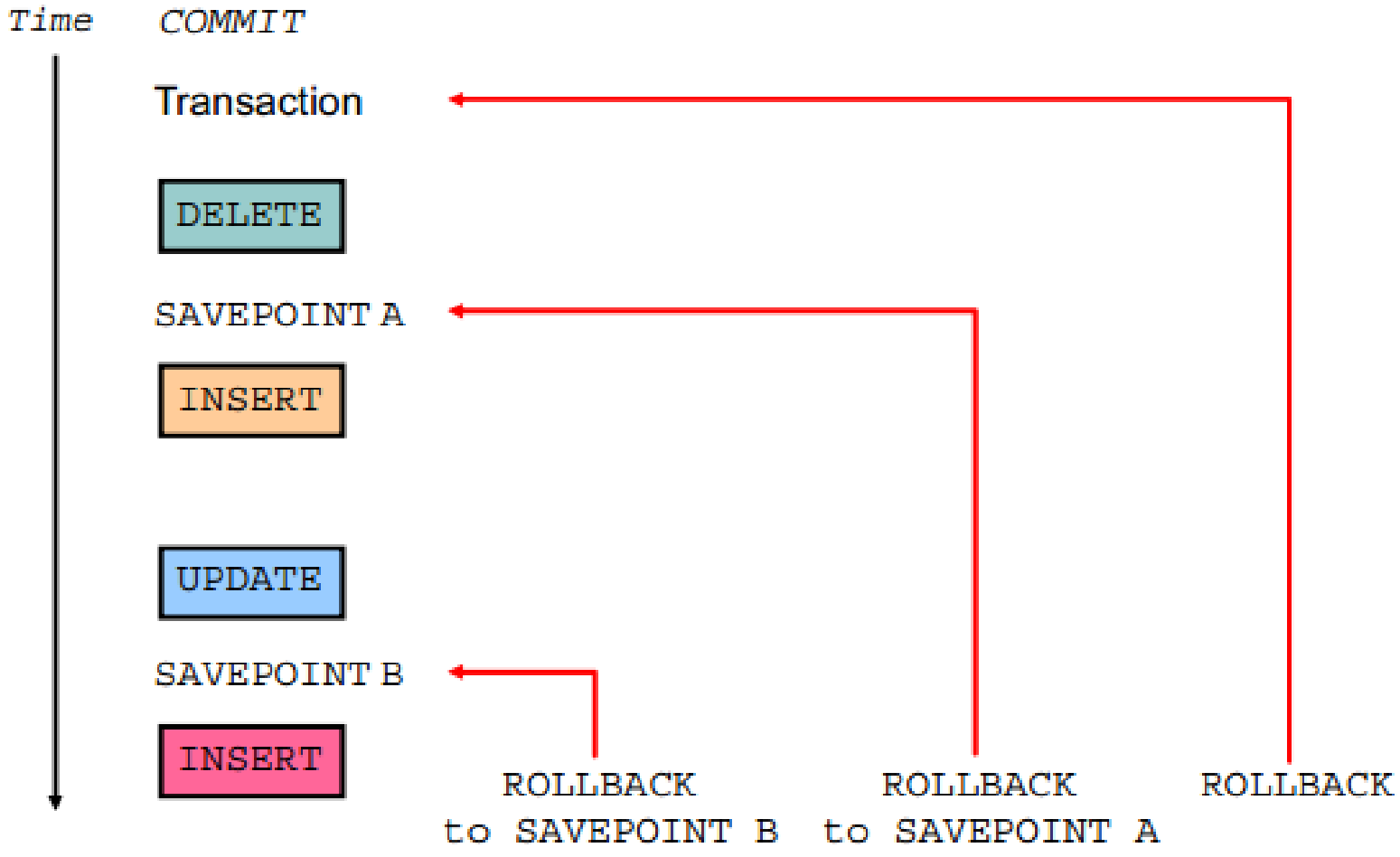
END

Ventajas del COMMIT y ROLLBACK

With `COMMIT` and `ROLLBACK` statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

Controlar las transacciones en la base de datos.



Transacciones implícitas

- An automatic commit occurs in the following circumstances:
  - A DDL statement issued
  - A DCL statement issued
  - Normal exit from SQL Developer or SQL*Plus, without explicitly issuing `COMMIT` or `ROLLBACK` statements
- An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus or a system failure.

Importancia del COMMIT y ROLLBACK

- The previous state of the data can be recovered.
- The current session can review the results of the DML operations by using the `SELECT` statement.
- Other sessions *cannot* view the results of the DML statements issued by the current session.
- The affected rows are *locked*; other session cannot change the data in the affected rows.

Estado de la data después de hacer el COMMIT

- Data changes are saved in the database.
- The previous state of the data is overwritten.
- All sessions can view the results.
- Locks on the affected rows are released; those rows are available for other sessions to manipulate.
- All savepoints are erased.

Estado de la data después de hacer el ROLLBACK

Discard all pending changes by using the `ROLLBACK` statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM copy_emp;
ROLLBACK ;
```


GALAXY
TRAINING