Tablas en Oracle

Tablas: son la unidad básica de almacenamiento de datos en Oracle. Los datos son almacenados en filas y columnas. La tabla se define a través de:

- Nombre.
- Conjunto de columnas (nombre y tipo)

Al crear una tabla(no-clusterizada), Oracle asigna un segmento de datos en el *tablespace* para mantener los datos de la tabla. Se puede controlar la asignación de espacio para el segmento de datos de la tabla y el uso de este espacio reservado de la siguiente manera:

- Cantidad de espacio asignado al segmento de datos, asignando los parámetros de almacenamiento del segmento.
- Controlando el uso del espacio vacío en los bloques que constituyen los extents del segmento, asignando el PCUSED y PCFREE.

Tipos de Tablas:

- Regulares:
- Particionadas:
- Organizadas por Índices:
- Agrupadas (clusterizadas)

Tablas Regulares

- Forma comunmente usada para almacenar los datos.
- Tipo por defecto.
- Se tiene muy poco control sobre la distribución de las filas de la tabla en su almacenamiento físico:
  - Se realiza de acuerdo a la actividad propia de los bloques de datos.

PCTUSED y PCTFREE

PCTFREE: valor entre 0 y 99. El valor de 0 permite que el blqoue entero se llene con inserciones de tuplas nuevas.

- Valor por defecto 10.
- Si no existen actualizaciones, PCTFREE= 0.
- En cualquier otro caso: PCTFREE=100*upd/(upd + ins)

- PCTUSED: valor entre 1 y 99. No es un parámetro configurable para tablas organizadas por índice.
  - Valor por defecto 40.
  - Se especifica si se eliminan filas.
  - PCTUSED=100-(PCTFREE +100*upd/blocksize) + 100ins/blocksize

Donde:

upd: cantidad promedio en bytes, añadido en una actualización.

ins: tamaño promedio de una fila al insertarse.

El PCTUSED permite inserciones en la tabla cuando hay suficiente espacio en el bloque para actualizaciones de filas y para una fila más.

En una tabla con muchas inserciones, cambiar el PCTUSED puede mejorar el desempeño de los bloques de datos.

La suma del PCTFREE y PCTUSED debe ser menor a 100.

PCTUSED y PCTFREE

PCTFREE: valor entre 0 y 99. El valor de 0 permite que el bloque entero se llene con inserciones de tuplas nuevas.

- Valor por defecto 10.
- Si no existen actualizaciones, asignar 0.
- En cualquier otro caso: PCTFREE=100*tam_prom_tupla/(tam_prom_tupla + tam_inic_tupla)

- PCTUSED: valor entre 1 y 99. No es un parámetro configurable para tablas organizadas por índice.
  - Valor por defecto 40.
  - Se especifica si se eliminan filas.
  - PCTUSED=100-PCTFREE -100*tam_prom_tupla/blocksize

Donde:

tam_prom_tupla: tamaño promedio de una fila

tam_inic_tupla: tamaño promedio de una fila al insertarse.

La suma del PCTFREE y PCTUSED debe ser menor a 100.

PCTUSED y PCTFREE

Un PCTFREE alto permite más actualizaciones en un bloque y el bloque puede acomodar menos filas.

Coloque un valor alto si la tabla contiene:

- Columnas que son inicialmente NULL y luego actualizadas.
- Columnas que pueden aumentar de tamaño cuando se actualizan.

El PCTUSED se configura para que el bloque retorne a la lista de ‘libres’ cuando hay suficiente espacio para acomodar una tupla promedio.

Encadenamiento y Migración

Una fila puede no ser almacenada completamente en un bloque si:

- Encadenamiento (chaining): la fila es demasiado larga para ser almacenada en un bloque. Esto puede ocurrir durante una inserción o una modificación.
  - Oracle almacena la fila en una cadena de uno o más bloques.
- Migración: si un UPDATE aumenta la cantidad de espacio ocupado por la fila, de forma que la misma no pueda ser almacenada en un bloque de datos.
  - Oracle trata de encontrar otro bloque con suficiente espacio para almacenar la fila completa. Si el bloque existe, se mueve la fila completa. Si no, Oracle separa la fila en varios “row pieces”, se mueve el (los) pedazo(s) que pueda(n) ser almacenado(s) y se realiza **encadenamiento**.
  - Oracle mantiene en el bloque original de una fila migrada a apuntar a un nuevo bloque que contiene la fila actual; el ROWID de una fila migrada no cambia. Los índices no se actualizan, ellos apuntan a la localización original del registro.

Tienen un efecto negativo en el tiempo de ejecución:

- INSERT y UPDATE que causan migración o encadenamiento, pueden tener un rendimiento pobre, porque requieren extra procesamiento.
- Consultas que usan un índice para seleccionar filas migradas o seleccionadas, deben ejecutar operaciones extras de I/O’s.

Encadenamiento y Migración

El comando ANALYZE permite identificar las filas de una tabla que se encuentran encadenadas o que han sufrido una migración.

ANALYZE <nombreTable> LIST CHAINED ROWS

Para poder utilizar el comando se requiere de una tabla especial llmada CHAINED_ROWS, que debe ser creada de la siguiente manera:

```
CREATE TABLE CHAINED_ROWS (
  owner_name varchar2(30),
  table_name varchar2(30),
  cluster_name varchar2(30),
  patition_name varchar2(30),
  head_rowid rowid,
  analyze_tiemstamp date);
```

Encadenamiento y Migración

Suponga que la tabla CHAINED_ROWS ya existe y se quiere chequear el encadenamiento y migración sobre la tabla X:

- Utilizar el comando ANALYZE para obtener las filas encadenadas o migradas de X:

```
ANALYZE TABLE X LIST CHAINED_ROWS
```

- Verificar si existen filas encadenadas o migardas en la tabla X del usuario Y:

```
SELECT COUNT(*)
FROM CHAINED_ROWS
WHERE OWNER_NAME="Y" AND TABLE_NAME="X"
```

Encadenamiento y Migración

Si existen filas encadenadas o migradas, seguir el siguiente procedimiento:

- Crear una tabla temporal con la misma estructura de X, con aquellas tuplas encadenadas o migradas:

```
CREATE TABLE X_TEMP
AS   SELECT * FROM X
WHERE ROWID IN (SELECT head_rowid
                  FROM CHAINED_ROWS WHERE table_name="X"
                and table_name="Y")
```

- Eliminar de la tabla las tuplas que tengan encadenamiento o migradas:

```
DELETE FROM X
WHERE ROWID IN (SELECT head_rowid
                  FROM CHAINED_ROWS WHERE table_name="X"
                and table_name="Y")
```

Encadenamiento y Migración

- Insertar las tuplas de la tabla temporal X_TEMP en la tabla original

  INSERT INTO X

  SELECT * FROM X_TEMP;

- Eliminar la tabla temporal X_TEMP.

  DROP TABLE X_TEMP;

- Usar el ANALYZE otra vez.
- Las tuplas que aparezcan en la salida, están encadenadas. Se pueden eliminar únicamente incrementando el tamaño del bloque de datos.

Tablas Particionadas

Una tabla particionada consiste de un número de piezas que tienen los mismos atributos lógicos.

Se puede crear una tabla particionada y tener una sola partición.

Una tabla particionada puede:

- Poseer una o más particiones, cada una de las cuales almacena las filas que poseen ciertas propiedades.
- Una tabla puede ser particionada en hasta 64.000 particiones separadas. Cualquier tabla puede particionarse, excepto aquellas que usen los tipos LONG o LONG RAW.
- Cada partición se coloca en un segmento que puede, a su vez, estar ubicado en un tablespace diferente. También se puede configurar atributos físicos tales como pctfree y pctused.
- La noción de partición permite manejar tablas que almacenan grandes volúmenes de datos de forma optimal, y permiten que un gran número de procesos pueda acceder a las mismas de forma concurrente.
- El DBMS provee comandos para manejar las particiones separadamente.
- Se recomiendan cuando:
  - La tabla tiene un tamaño superior a 2 GB.
  - Tablas que mantienen históricos.

Tablas Particionadas-Métodos de Particionamiento

- Particionamiento por Rango: se hace corresponder datos en particiones haciendo uso de los rangos de los valores de la clave de particionamiento. Se deben considerar las siguientes reglas:
  - Cada partición se define con la clausula VALUES LESS THAN, la cual especifica un límite superior no inclusive para las particiones. Cualquier valor de la clave de la partición igual o superior, es añadida a la proxima partición.
  - Todas las particiones, excepto la primera, tienen un límite inferior implícito, especificado en la clausula VALUES LESS THAN de la partición previa.
  - Un literal MAXVALUE puede ser definido para la última partición; representa un valor virtual de infinito.

Tablas Particionadas

```
CREATE TABLE est
 (carnet char(10),
   cohorte number(2),
   nombre char(20),
   carrera char(20))
   partition by range (carrera)
   (partition sx1992 values less than (90) tablespace ts0
    partition sx1993 values less than (96) tablespace ts1
   partition sx1994 values less than (04) tablespace ts2)
```

Tablas Particionadas-Métodos de Particionamiento

- Particionamiento por Lista: permite especificar explicitamente la correspondencia entre las filas y las particiones. Se especifica una lista de valores discretos para la clave de particionamiento. No se soportan claves de particionamiento formadas por varios atributos.

```
CREATE TABLE sales_list
(salesman_id NUMBER(5),
  sales_name VARCHAR2(30),
  sales_state VARCHAR2(20),
  sales_amount NUMBER(10),
  sales_date DATE)
PARTITION BY LIST(sales_state)
(PARTITION sales_west VALUES(‘California,’Hawai’’),
PARTITION sales_east VALUES (‘New York’, ‘Virginia’, ‘Florida’)
PARTITION sales_central VALUES (‘Texas’, ‘Illinois’)
PARTITION sales_other VALUES(DEFAULT));
```

Tablas Particionadas-Métodos de Particionamiento

- Particionamiento por Hash: la correspondencia entre las filas y las particiones se realiza a través de una función de hash. Es una opción útil cuando:
  - Se desconoce la correspondencia en función de los rangos.
  - El rango de las particiones difiere sustancialmente o es difícil balancearla manualmente.

```
CREATE TABLE sales_hash
(salesman_id NUMBER(5),
 sales_name VARCHAR2(30),
 sales_amount NUMBER(10),
 week_no NUMBER(2))
PARTITION BY HASH(salesman_id)
PARTITIONS 4
STORE IN (data1, data2, data3, data4);
```

La tabla sales_hash se particiona en función de los valores de la columna salesman_id. Las particiones se almacenan en los tablespaces: data1, data2, data3, y data4.

Particionamiento por Composición: es un método de particionamiento donde cada partición se particiona a su vez es sub-particiones.

Las particiones más generales se hacen con el método de rango.

Cada partición se sub-particiona con el método de hash o por lista.

```
CREATE TABLE sales_composite
(salesman_id NUMBER(5),
  sales_name VARCHAR2(30),
  sales_amount NUMBER(10),
  sales_date DATE)
PARTITION BY RANGE(sales_date)
SUBPARTITION BY HASH(salesman_id)
SUBPARTITION TEMPLATE(
SUBPARTITION sp1 TABLESPACE data1,
SUBPARTITION sp2 TABLESPACE data2,
SUBPARTITION sp3 TABLESPACE data3,
 SUBPARTITION sp4 TABLESPACE data4)
(PARTITION sales_jan2000 VALUES THAN (TO_DATE('02/01/2000','DD/MM/YYYY')
PARTITION sales_fec2000 VALUES THAN (TO_DATE('03/01/2000','DD/MM/YYYY'
PARTITION sales_mar2000 VALUES THAN (TO_DATE('04/01/2000','DD/MM/YYYY'
PARTITION sales_apr2000 VALUES THAN (TO_DATE('05/01/2000','DD/MM/YYYY'))
```

La partición sales_jan2000_sp1 se almacena en el tablespace data1….

Tablas Agrupadas o Clustered

Un cluster es un grupo de tablas que se almacenan en mismo bloque de datos porque comparten columnas y usualmente se usan juntas.

Emp(ci,nombre,cod_dep)

Dep(cod_dep,nombre_dep,presupuesto)

Al *clusterizar* las tablas Emp y Dep, se almacena físicamente todas las filas por cada departamento en las tablas Emp y Dep en los mismos bloques.

Beneficios de clusterizar dos tablas:

- I/O's a disco se reducen en joins de tablas clusterizadas.
- Tiempo de acceso se mejora para joins de tablas clusterizadas.
- En un cluster, un valor de la clave del cluster, es el valor de las columnas claves del cluster. Cada valor de la clave del cluster, se almacena una sola vez, en el cluster de datos o de índice, no importa cuantas filas tengan ese valor.

Tablas Agrupadas o Clustered

Los clusters pueden reducir el rendimiento de instrucciones de INSERT en comparación a la misma operación en una tabla almacenada independientemente de su índice.

- Registros de múltiples tablas se almacenan en un mismo bloque y se requieren más bloques que si la tabla no se clusterizara.
- Columnas que se actualizan frecuentemente no son buenas candidatas a claves de un cluster.

Las tablas candidatas a formar un cluster son aquellas que se relacionan por una restricción de integridad referencial y tablas que se acceden frecuentemente juntas haciendo el uso del join:

Todas las tuplas que satisfacen el join están en un mismo bloque.

Un cluster puede mantener una tabla simple.

Los datos de una tabla clusterizada son almacenados en el segmento creado para el cluster en lugar de en un segmento de datos en un tablaspace. Parámetros de almacenamiento no pueden ser especificados cuando una tabla cluster es creada o alterada. Los parámetros de almacenamiento para el cluster siempre controlan el almacenamiento de todas las tablas en el cluster.

Tablas Agrupadas o Clustered

```
CREATE TABLE emp (
  empno NUMBER(5) PRIMARY KEY,
  ename VARCHAR2(15) NOT NULL,
  …..
  deptno NUMBER(3) REFERENCES dept)
  CLUSTER emp_dept (deptno);

CREATE TABLE dept (
 deptno NUMBER(3) PRIMARY KEY,…)
  CLUSTER emp_dept (deptno);
```

Tablas Agrupadas- Índice sobre los Clusters

Una vez creado un cluster de tablas, se debe crear un índice sobre el cluster, que contiene una entrada por cada valor de la clave del cluster.

Para localizar una fila en el cluster A, el índice se utiliza para localizar el valor de la clave del cluster, el cual apunta al bloque de datos asociado con el cluster A.

El índice sobre un cluster A debe ser creado antes que cualquier instrucción de DML se ejecute sobre A.

Un índice sobre un cluster se diferencia de un índice sobre una tabla en:

•Claves que son nulas tienen una entrada en el índice sobre el cluster.

•Entradas del índice apuntan al primer bloque en la cadena para un valor de clave de cluster dado.

•Un índice sobre cluster contiene una entrada por valor de la clave del cluster.

•La ausencia de una tabla de índice no afecta a los usuarios, sin embargo, los datos clusterizados no pueden ser accedidos hasta que no exista un índice sobre el cluster.

Tablas Agrupadas- Hash Clusters

Un hash cluster hace corresponder una fila con el cluster al cual debe pertenecer a través de una función de hash.

La función de hashing puede ser definida por el usuario o generada por el DBMS.

Cuando una fila se inserta en una tabla agrupada por hashing, se realizan las siguientes acciones:

- Se utilizan las columnas de la clave del cluster para calcular la función de hashing.
- La fila se almacena en la ubicación obtenida a partir de la función de hashing.

Índices

Son estructuras opcionales asociadas a tablas o clusters.

- Oracle ofrece:
- Indices B*-tree.
- Clusters B*-tree
- Hash clusters
- Reverse key
- Bitmaps.
- Los índices pueden ser:
- únicos: dos filas no tiene el mismo valor en el índice
- no-únicos: dos o más filas pueden tener el mismo valor en el índice.
- Compuestos: definidos sobre más de un atributo.

Índices

Al crearse un índice, Oracle reserva espacio en un segmento de índice para mantener los datos de índice en un tablespace particular. Se puede controlar el espacio de un segmento de índice y usar el espacio reservado de la siguiente forma:

- Asignar los parametros del segmento de índice para controlar la asignación de extents a los segmentos de índice.
- Asignar el parámetro de PCTFREE del segmento de indice para controlar el espacio vacío en los bloques de datos que constituyen.

Un índice no tiene porque ser almacenado en el mismo tablespace que sus datos.

Tablas Organizadas por Índices

- Son tablas que mantienen los datos ordenados por la clave primaria (Indice Clustered Nomenclatura Teoría).

- Una tabla organizada por índice es una alternativa a:
  - Una tabla noclustered indexada por la clave primaria usando el comando CREATE INDEX.
  - Una tabla clustered almacenada en un indice cluster que ha sido creado usnado el comando CREATE CLUSTER que hace corresponder la clave primaria como clave del cluster.
- Se mantienen las filas de la tabla en un B*-tree construido en la clave primaria. Cada fila del índice contiene los valores de la clave y del resto de los atributos no claves:
- <primary_key_value, non_primary_key_column_values>

Tablas Organizadas por Índices

Tablas Regulares	Tablas Organizadas por Indice
ROWID identifica univocamente a una fila: claves primarias son opcionales	Una Clave primaria identifica una fila: la clave primaria debe ser identificada
ROWIDs físicos permiten construir índices secundarios	ROWIDs lógicos(basado en la clave primaria), pueden tener índices secundarios.
Accesos basados en el ROWID físico	Acceso basado en ROWID lógico
Scan retorna todas las tuplas	Scan del índice retorna todas las filas ordenadas por la clave primaria.
Una tabla puede ser almacenada en un cluster que contenga otras tablas	No puede ser almacenada en un cluster
Puede contener una columna de tipo Long and columnas de tipo LOB	Puede contener columnas de tipo LOB pero no de tipo LONG

Tablas Organizadas por Índices

- Beneficios de las tablas organizadas por índices:
- Acceso rápido para consultas con matching exacto y busqueda por rangos.
- No se requiere espacio para almacenar los ROWID de los registros.

Areas de OverFlow:

Las entradas de una tabla organizada por índice pueden ser muy grandes y requerir varios bloques. Para manejar el problema se puede especificar:

- un overflow tablespace:
- un threshold value: porcentaje del tamaño del bloque que indica el tamano a partir del cual se almacenara el registro en el area de overflow. Si un registro excede el tamano del threshold, entonces las columnas cuyos valores de las columnas para la fila que excede el threshold son almacenados en el área de overflow.

Tablas Organizadas por Índices

```
CREATE TABLE emp
  (ci char(8) ,
   nombre char(30),
   profesion char(20)
   constraint pk_emp PRIMARY KEY (ci))
  ORGANIZATION INDEX TABLESPACE emp_collection
  PCTTHRESHOLD 20 INCLUDING nombre
  OVERFLOW TABLESPACE emp_collection_overflow;
```

Cuando el tamano de una fila excede al 20% del tamano del bloque, todas las columnas que estan despues de la columna **nombre**, seran movidas al segmento del tablespace emp_collection_overflow.

Si no se coloca el INCLUDING, se mueven todas las columnas excepto la clave primeria.

Notar que:

- Índices: son estructuras usadas para proveer acceso más rápido a los datos.
- Clave Primaria: conjunto minimal de atributos que identifica univocamente a las tuplas de una tabla.

Almacenando las Filas de una Tabla

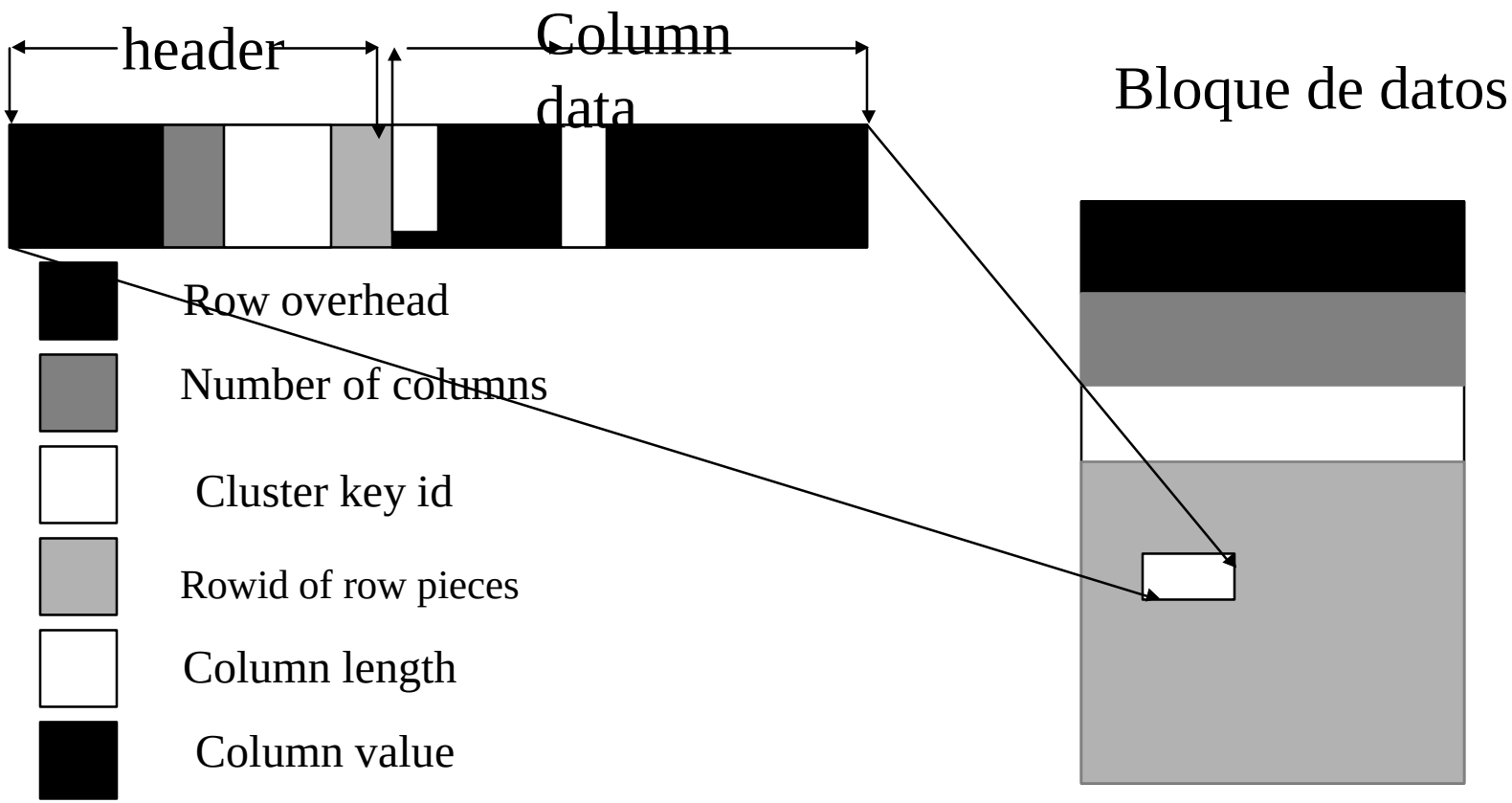
Oracle almacena una fila de una tabla en una o varias piezas de fila (row pieces).

Si un bloque tiene suficiente espacio para almacenar una fila completa, entonces se almacena como una sola pieza.

En caso contrario, se almacena en diferentes bloques como diferentes piezas encadenadas.

Si una fila tiene más de 256 columnas, a partir de la columna 255, se almacena encadenada en el mismo bloque (encadenamiento interno de bloques).

Cada pieza de fila, encadenada o no, mantiene un encabezado y datos por todas o algunas de las columnas de la fila.



Almacenando las Filas de una Tabla

El row header permite almacenar las características sobre la estructura y uso de una fila como información de control. Es la unidad básica que permitirá efectuar la lectura lógica de la fila cuando el bloque de datos Oracle se encuentre cargado en memoria principal. precede los datos y contiene información sobre:

- Overhead: inidica que comienza una pieza de fila (row piece)
- Número de Columnas que aparecen en el row piece.
- Clave del cluster: en caso de “clusters”, la información sobre las columnas que forman parte de la clave del cluster.
- Encadenamiento (para filas encadenadas) Row pieces: si hay encadenamiento, el ROWID del siguiente “row piece” que permite componer la fila. El ROWID es un valor único en Oracle para identificar univocamente a una fila.

Una fila contenida complementamente en un bloque tiene al menos 3 bytes de header. Después del header, cada columna contiene la longitud de la columna y su valor. La longitud de la columna requiere de 1 byte para columnas que almacenan 250 bytes o menos, o 3 bytes para las columnas que almacenan para más de 250 bytes, y precede la columna de datos. Espacio requerido para la columna de datos depende del tipo de datos.

Cada fila también usa 2 bytes en el directorio de fila (row directory) en el header del bloque de datos

Si el tipo de datos es variable, entonces el espacio requerido para mantener el valor puede crecer o encogerse.

Para ahorar espacio, un valor nulo se representa con una columna de longitud cero. Si las últimas K coumnas de una fila son nulas, no se almacena no el valor de longitud igual a cero.

Una regla general es colocar las filas con un alto porcentage de nulos al final de la sentencia CREATE TABLE.

ROWID-Tipo de datos

El tipo de datos ROWID permite distinguir una columna especial que posee toda tabla en Oracle y que puede ser consultada en cualquier momento.

Toda fila en una tabla no agrupada, posee un único ROWID que permite ubicar el primer row piece de la fila. En caso de tablas agrupadas, las filas de tablas diferentes que se encuentren en el mismo bloque de datos, podrán tener el mismo ROWID.

El valor del rowid puede ser consultado de la siguiente manera:

Select rowid From EMP;

Los ROWID poseen las siguientes características:

- Es un identificador unívoco para cada fila en la base de datos.
- Los ROWIDs no se almacenan físicamente como una columna.
- Aunque el ROWID no provee la dirección específica de la fila a nivel de los bloques del sistema de operación, puede ser utilizado para localizar una fila en un bloque de Oracle.
- Pueden ser usados para ver como los datos son almacenados(Rowid físicos).
- Para la tablas organizadas por índice, Oracle provee un rowid lógico el cual se basa en en la clave primaria de la clave-se usa para la construcción de índices secundarios. Los rowid’s lógicos no cambian mientras la clave de la fila no cambie. No pueden ser usados para chequear como la tabla es organizada.

ROWID-Formatos

- RowIds Extendidos: este formato suporta direcciones de datos en los bloques que son relativas al tablespace. Identifica eficientemente filas en cualquier tipo de tabla. Disponible para versiones de Oracle superiores a 8.
- RowIds Restringidos: este formato es disponible para versiones inferiores a 7.

RowIds extendidos usan una base 64 para la codificación de las direcciones físicas de una fila. Los caracteres de codificación son: A-Z, a-z, 0-9, +, and /.

Por ejemplo, la consulta:

SELECT ROWID, last_name FROM employees WHERE department_id = 20;

puede retornar la siguiente información de fila:

ROWID LAST_NAME

AAAAaoAATAAABrXAAA BORTINS

AAAAaoAATAAABrXAAE RUGGLES

AAAAaoAATAAABrXAAG CHEN

AAAAaoAATAAABrXAAN BLUMBERG

ROWID-Formatos

Un rowid extendido tiene el siguiente formato OOOOOOFFFBBBBBBRRR: (18 caracteres codificados en base 64)

- OOOOOO: (6 caracteres) el número del objeto de datos que identifca al segmento donde está el objeto.
- FFF: (3 caracteres) el número del archivo de datos que contiene el blqoue donde está la fila. Este valor es relativo al tablespace.
- BBBBBB: (6 caracteres) bloque de datos que contiene a la fila. Valor relativo al archivo de datos.
- RRR: (3 caracteres) fila en el bloque de datos.

```
Select SUBSTR(ROWID,1,6) ”DON”,  SUBSTR(ROWID,7,3) “RFN”,
        SUBSTR(ROWID,10,6) “BN”, SUBSTR(ROWID,16,3) “RN”
From EMP
Where nro_emp=1;
```

Si la respuesta es: AAAAAS AAB AAAGF1AAV, entonces:

- AAAAAS: $(0*64^5 + 0*64^4 + 0*64^3 + 0*64^2 + 0*64^1 + 18*64^0)$objeto con identificador decimal 18 en decimal.
- AAB: $(0*64^2 + 0*64^1 + 1*64^0)$la fila se encuentra en el archivo de datos 1 decimal.
- AAAGF1$(0*64^5 + 0*64^4 + 0*64^3 + 6*64^2 + 5*64^1 + 53*64^0)$: la fila se encuentra en el bloque 8565 decimal.
- AAV$(0*64^2 + 0*64^1 + 21*64^0)$: la fila 21-ima del bloque.

ROWID-Formatos

Se puede chequear los números de los objetos en las vistas:

USER_OBJECTS, DBA_OBJECTS, and ALL_OBJECTS

Por ejemplo, la siguiente consulta retorna el número de objeto de datos para las filas en la tabla Emp cuyo propietario es SCOTT.

```
SELECT DATA_OBJECT_ID FROM DBA_OBJECTS
   WHERE OWNER = 'SCOTT' AND OBJECT_NAME = 'Emp';
```