



Servicio Andaluz de Salud
CONSEJERÍA DE SALUD

*Oficina Técnica para la Gestión y Supervisión de
Servicios TIC
Subdirección de Tecnologías de la Información*

Particionamiento en Oracle Conceptos y Buenas Prácticas para Administradores

*Referencia documento:
InfV5_JASAS_Partitioning_System_BestPractices_V920.doc
Fecha: 13 de noviembre de 2018
Versión: 9.2.0*

Registro de Cambios

Fecha	Autor	Versión	Notas
17 de Octubre de 2013	Isidro Granados	4.3.0	Versión inicial del documento
08 de Julio de 2015	Isidro Granados	6.1.0	Ampliación cálculo estadísticas en tablas particionadas.
16 de Diciembre de 2015	Isidro Granados	6.2.0	Revisión de Diciembre de 2015, contrato 2014-2016
16 de Junio de 2016	Isidro Granados	7.1.0	Revisión de Junio de 2016, contrato 2014-2016
14 de octubre de 2016	Isidro Granados	7.2.0	Revisión de Noviembre de 2016, contrato 2014-2016
14 de junio de 2017	Isidro Granados	8.1.0	Revisión de Junio de 2017, contrato 2016-2018
16 de noviembre de 2017	Isidro Granados	8.2.0	Revisión de Noviembre de 2017, contrato 2016-2018
16 de junio de 2018	Isidro Granados	9.1.0	Revisión de junio de 2018, contrato 2016-2018
13 de noviembre de 2018	Isidro Granados	9.2.0	Revisión de noviembre de 2018, contrato 2016-2018

Revisiones

Nombre	Role
Jonathan Ortiz	Advanced Support Engineer
Gregorio Adame	Advanced Support Engineer
Jose María Gómez	TAM

Distribución

Copia	Nombre	Empresa
1	Subdirección de Tecnologías de la Información	Servicio Andaluz de Salud, Junta de Andalucía
2	Servicio de Coordinación de Informática de la Consejería de Innovación	Consejería de Innovación, Junta de Andalucía

Índice de Contenidos

CONTROL DE CAMBIOS	4
INTRODUCCIÓN	5
OBJETIVOS DE ESTE DOCUMENTO	7
CONCEPTOS DE PARTICIONAMIENTO.....	8
<i>Particionamiento como opción de Base de datos. Licencia. Instalación.</i>	8
<i>Visión general particionamiento</i>	9
<i>Beneficios del particionamiento.</i>	10
<i>Estrategias de Particionamiento de Tablas.</i>	11
<i>Indices Particionados. Tipos.</i>	20
<i>Algunas restricciones con el uso de particionamiento</i>	23
ADMINISTRANDO BASE DE DATOS CON PARTICIONAMIENTO.	24
<i>Creando Particiones</i>	24
<i>Compresión en tablas e índices particionados.</i>	24
<i>Creación diferida de segmentos en particiones</i>	26
<i>Operaciones de Mantenimiento en Tablas e Índices particionados.</i>	27
<i>Vistas del Data Dictionary para el manejo del Particionamiento</i>	43
<i>Métodos para convertir una tabla no particionada en particionada</i>	46
CÁLCULO DE ESTADÍSTICAS PARA EL OPTIMIZADOR EN TABLAS PARTICIONADAS.....	50
<i>Buenas prácticas en el cálculo de estadísticas con tablas particionadas</i>	51
<i>Estadísticas Incrementales</i>	51
OTROS USOS DE PARTICIONAMIENTO.....	66
<i>SQL Loader y Objetos Particionados</i>	66
<i>Datapump y Particionamiento</i>	66
<i>Particionamiento y Tablespace Transportables</i>	66
<i>Particiones read-only</i>	67

Control de cambios

Cambio	Descripción	Página
1	No se realizan cambios en esta versión del documento.	N/A

Introducción

Una base de datos Oracle puede ser realmente grande y no es difícil encontrar tablas que son capaces de ocupar varios gigabytes o incluso varios terabytes de datos. Particionamiento es la opción que ofrece el RDBMS Oracle para dividir lógicamente en trozos una tabla y un índice para facilitar el procesamiento y rendimiento de consultas y operaciones DML así como el manejo y mantenimiento de una base de datos con estas circunstancias.

Particionamiento es una opción que se ofrece como opción licenciable para Enterprise Edition. El particionamiento aparece por primera vez en la versión 8 del servidor de Oracle y ha ido evolucionando con cada nueva versión.

Esta opción nació pensada fundamentalmente para entornos Data Warehouse, en los que se manejan tablas de un tamaño considerable. Pero cualquier tipo de base de datos que contenga tablas de tamaño elevado se puede beneficiar de la utilización del particionamiento de Oracle. Estos entornos que manejan tan grandes cantidades de datos presentan serios problemas tanto de rendimiento como de mantenimiento, la opción de particionamiento de Oracle es la solución más adecuada para estos dos problemas.

Sin embargo, no todas las tablas y situaciones se pueden beneficiar del particionamiento. Un particionamiento inadecuado o innecesario puede suponer un empeoramiento en el rendimiento, además de un consumo innecesario de tiempo de trabajo del administrador de la base de datos.

La opción de Particionamiento viene ligada a concepto de Base de datos tipo VLDB (very large database). Una VLDB no tiene un mínimo absoluto de tamaño y aunque no deja de ser una base de datos Oracle hay varios aspectos específicos que aplican principalmente a una VLDB provocados por su gran tamaño y por el coste de realizar operaciones en un sistema de este tamaño.

Varias tendencias han sido responsables del crecimiento constante del tamaño de las base de datos:

- Durante mucho tiempo los sistemas han sido desarrollados de forma independiente. Los departamentos de IT han empezado a ver los beneficios de la consolidación, esto es, combinar estos sistemas y habilitar bases de datos interdepartamentales para reducir los costes de mantenimiento de estos sistemas independientes. La consolidación de base de datos y aplicaciones es un factor clave para el crecimiento continuo del tamaño de las bases de datos.
- Muchas compañías se enfrentan a Regulaciones que obligan a guardar datos durante un mínimo tiempo, cuyo resultado es que muchos datos son guardados durante largos periodos de tiempo.
- La compañías crecen expandiendo ventas u operaciones o a través de fusiones o adquisiciones provocando que la cantidad de datos generados y procesados se incrementa.

El Particionamiento es una característica clave en el manejo de una VLDB. Particionamiento permite implementar la técnica de "Divide y Venceras" para tablas o índices muy grandes, especialmente aquellas en continuo crecimiento. Con esta técnica es posible en una base de datos mantener consistentemente el rendimiento y concurrencia a la vez de que se sigue creciendo en tamaño, esto sin tener que incrementar los recursos hardware o administrativos.

El particionamiento permite dividir en trozos más pequeños tablas, índices y tablas organizadas por índices (IOTs) de forma que puedan ser accedidos a un nivel más granular. Oracle dispone de una gran variedad de técnicas y extensiones de particionamiento de forma que se pueda adaptar a cualquier requerimiento de negocio.

El particionamiento es totalmente transparente, por tanto puede ser implementado en casi cualquier aplicación sin necesidad de realizar cambios en la aplicación.

Objetivos de este documento

La administración del particionamiento es una importante tarea cuando se trabaja con tablas e índices particionados. Para el mantenimiento de objetos particionados, Oracle introduce técnicas adicionales al mantenimiento normal de objetos no particionados. El objetivo principal del informe es mostrar cuales son estas técnicas, e indicar en caso de que sean relevantes cuales son las mejores prácticas en el uso y mantenimiento de objetos particionados dentro de bases de datos Oracle por parte de los administradores de base de datos.

El documento comienza con una visión global de los conceptos mínimos de particionamiento necesarios para poder administrar una base de datos Oracle donde se ha implementado o se va a implementar una estrategia de particionamiento.

Este documento complementa el informe InfV5_JASAS_Partitioning_Development_BestPractices_V820.doc donde se muestran en detalle los conceptos y las técnicas necesarias para implementar de forma efectiva el particionamiento en Oracle, mostrando escenarios típicos de particionamiento y estrategias óptimas de particionamiento.

El documento se centra principalmente en la versión Oracle RDBMS 12cR2 (12.2.0.1), aunque muchos de los conceptos y recomendaciones son igualmente aplicables a las versiones anteriores. El documento irá indicando las versiones en las que los cambios o mejoras que se indican han sido introducidas desde versión 11g a 12cR2.

Conceptos de Particionamiento

Particionamiento como opción de Base de datos. Licencia. Instalación.

Particionamiento está solo disponible en la Enterprise Edition de Oracle Database. No está disponible ni con Standard Edition ni en Express Edition.

El particionamiento en Oracle se divide entre System y User Partitioning. El System Partitioning se refiere al uso que hace Oracle de particionamiento para objetos propios como objetos de diccionario.

El particionamiento tiene un coste Extra de licencia para el User Partitioning.

La instalación de Oracle Partitioning en versiones anteriores a 11g, se hace en la instalación con Universal Installer de Enterprise Edition con la opción Typical, aunque si se elige la opción custom install, se podía seleccionar o no la instalación de Oracle Partitioning.

En versiones superiores a 11g se instalará siempre si se elige Enterprise Edition, esto es, no existe el custom install, aunque en la instalación en *select Options*, podremos habilitar y deshabilitar el componente Oracle Partitioning.

Una vez instalada y habilitada, tendremos la opción de deshabilitarla, en versión 11.2 y 12cR2 fácilmente con el comando [chopt](#).

```
$ chopt disable partitioning  
$ chopt enable partitioning
```

Para saber si está habilitado el particionamiento podremos verlo desde la base de datos o desde el file system donde está el ORACLE_HOME instalado. Desde la bbdd podemos consultar:

```
select * from v$option where parameter='Partitioning';
```

PARAMETER	VALUE
-----	-----
Partitioning	TRUE

Si marca True, estará habilitado, en otro caso, deshabilitado.

Para saber si se está usando particionamiento en nuestra base de datos poder consultar la vista DBA_FEATURE_USAGE_STATISTICS.

Por ejemplo una bbdd creada en 12.2.0.1 podemos ver que no hay uso de Particionamiento:

```
select name, version, last_usage_date, currently_used from  
DBA_FEATURE_USAGE_STATISTICS where upper(name) like '%PARTITION%';
```

NAME	VERSION	LAST_USAGE	CURRE
-----	-----	-----	-----
Partitioning (user)	12.2.0.1.0		FALSE
Partitioning (system)	12.2.0.1.0	06-11-2017	TRUE

Nota: Si se deshabilita la opción de particionamiento, esto afecta sólo a User Partitioning, pero la funcionalidad permanece en el ejecutable Oracle. Las tablas particionadas antes de que se lance el disable de la opción de particionamiento permanecerán particionadas y con el mismo comportamiento. Sin embargo si un usuario intenta crear una nueva tabla o índice particionado generará el error: ORA-00439: feature not enabled: Partitioning.

Visión general particionamiento

El particionamiento consiste básicamente en dividir una tabla, índice o IOT en trozos más pequeños, cada trozo de un objeto particionado se llamará partición, o incluso subpartición.

Desde la perspectiva de un DBA, un objeto particionado podrá ser manejado de forma global o individualmente. Esto nos da una gran flexibilidad a la hora de utilizar el particionamiento y posibilita una serie de operaciones que facilitan notablemente las labores de mantenimiento de las bases de datos

Desde el punto de vista de la aplicación, una tabla particionada es idéntica a una no particionada, y no habrá que hacer cambios en consultas SQL o sentencias DML.

Cada partición tendrá su propio nombre, y se corresponderá físicamente con un segmento. Estos segmentos opcionalmente podrán tener sus propias características de almacenamiento, por ejemplo las particiones de un objeto particionado podrán residir en tablespaces distintos, con la restricción de que todos los tablespaces tengan el mismo tamaño de bloque.

Para tablas normales, una tabla es ambas cosas un objeto y un segmento. Para una tabla particionada, una tabla es sólo un objeto pero sin segmento. La tabla contendrá particiones que serán objetos y segmentos. Si la tabla es particionada compuesta, entonces las particiones solo son objetos sin un segmento asignado y las subparticiones serán los segmentos físicos.

Cada partición en una tabla o índice deberá tener los mismos atributos lógicos, como nombres de columnas, tipos de datos y "constraints". Por ejemplo, todas las particiones de una tabla compartirán la definición de las columnas, aunque cada partición podrá tener atributos físicos diferentes como el "tablespace" o el "pctfree".

Clave de Particionamiento (Partition Key)

El criterio para particionar la tabla se definirá sobre una o más columnas que la forman (esta columna o columnas formarán lo que se denomina como **clave de particionamiento**). Cada fila en una tabla particionada será asignada de forma inequívoca a una única partición. Para determinar en qué partición se guarda una fila se usará esta clave de particionamiento. Oracle automáticamente redirige los comandos insert, update y delete a la partición apropiada usando esta clave de particionamiento.

Por ejemplo en una tabla *ventas*, se podría usar la columna *time_id* como clave de particionamiento por rango. La base de datos asignará automáticamente las filas a las particiones basándose cuando una fecha está en un rango específico.

Beneficios del particionamiento.

El concepto de partición es interno y prácticamente invisible al usuario. El usuario se referirá a la tabla completa como si fuera un único objeto al igual que si la tabla fuera no particionada. Una tabla podría pasar de no particionada a particionada y viceversa sin que los usuarios finales tuvieran conocimiento de este hecho o hubiese que cambiar alguna SQL o aplicaciones. Sin embargo, el hecho de particionar una tabla ofrece diversos beneficios:

Mejoras en el Rendimiento

El particionamiento ofrece una serie de mejoras en cuanto al rendimiento ya que dada una consulta sobre la tabla particionada, el optimizador basado en costes puede eliminar de forma transparente las particiones que no contienen datos que satisfagan la consulta. Este mecanismo llamado “partition pruning” es transparente a la aplicación y mejorará considerablemente los tiempos de respuesta de la base de datos.

Otra ventaja es el hecho de que al tratarse cada partición de un segmento de datos diferenciado, se puede ubicar de forma de se eviten contenciones de E/S.

En algunos sistemas tipo OLTP, las particiones pueden ayudar a descender la contención de un recurso compartido. Por ejemplo, operaciones tipo DML se pueden distribuir sobre muchos segmentos en lugar de uno solo.

En un entorno Data Warehouse, el particionamiento puede incrementar la velocidad del procesamiento de consultas grandes ya que las operaciones podrán paralelizarse y ejecutarse concurrentemente en cada partición. Estas particiones podrán incluso localizarse en físicamente en diferentes sitios mejorando la concurrencia.

Mejoras en el Mantenimiento y Disponibilidad

Cada partición se puede administrar separadamente. Esto significa dos cosas:

- Las operaciones se realizarán sobre un segmento de tamaño considerablemente inferior al de la tabla, por lo que los tiempos de indisponibilidad serán significativamente menores.
- En caso de pérdida de una partición, si se encuentra en un tablespaces separado del resto de particiones, las labores de restauración y recuperación afectarán sólo a esta partición, quedando el resto de la tabla accesible.

Igualmente se mejorará la disponibilidad ya que en caso de que una partición no esté disponible, el optimizador automáticamente borrará particiones que no sean referenciadas en el plan de acceso y así consultas no serán afectadas cuando haya particiones no disponibles.

Se han creado nuevas opciones de mantenimiento en herramientas como SQL*Loader, Datapump, etc. que permiten estas operaciones de mantenimiento sobre particiones individuales sin alterar la disponibilidad de resto de la tabla.

Un beneficio adicional es que al ocuparnos de segmentos de tamaño reducido, minimizamos los tiempos de recuperación y los recursos utilizados para ello.

Oracle dispone de gran variedad de métodos y comandos para el manejo de las particiones. Por ejemplo:

- Una partición podrá ser movida de un tablespace a otro.
- Una partición puede ser dividida en varias particiones
- Una partición puede ser borrada, añadida o truncada.
- Aunque las operaciones `SELECT`, `UPDATE`, `INSERT` y `DELETE` no necesitan cambiarse por el hecho de tener una tabla particionada, se pueden restringir a nivel de partición en lugar de que apliquen a la tabla completa, añadiendo la cláusula `PARTITION` en el `FROM`.

Las operaciones de mantenimiento que se pueden realizar sobre las tablas particionadas son:

- Add table partition
- Modify partition
- Move table partition
- Rename partition
- Drop partition
- Truncate partition
- Split partition

Por ejemplo el `DROP` y el `TRUNCATE` de una partición evitarán la ejecución de muchas sentencias `DML DELETE`.

Estrategias de Particionamiento de Tablas.

El particionamiento en Oracle ofrece varias estrategias para indicar el método en que la base de datos pone en el lugar adecuado los datos en particiones. Dependiendo del método de distribución de datos Oracle ofrece tres modelos fundamentales de particionamiento:

- Particionamiento por rangos (y su extensión por intervalos desde 11g)
- Particionamiento hash (desde Oracle8i)
- Particionamiento por listas (desde Oracle9i)

Además se podrá elegir entre particionamiento de un solo nivel o particionamiento compuesto. Desde 11g se añaden varias extensiones de particionamiento que aumentarán las opciones de elegir una estrategia de particionamiento. También en versión 12c y 12cR2 se añaden nuevas mejoras y opciones de particionamiento que serán indicadas en los siguientes apartados.

Cada estrategia tiene diferentes ventajas y consideraciones de diseño. Así cada estrategia será más adecuada a una situación particular.

Particionamiento de un sólo nivel (Single-Level Partitioning)

En este caso la tabla se crea especificando sólo uno de los siguientes métodos de distribución usando una o más columnas como clave de particionamiento:

- Particionamiento por rangos
- Particionamiento hash
- Particionamiento por listas

Particionamiento por rangos

Este es el primer tipo de particionamiento disponible desde Oracle 8. Es el que más se suele aplicar, fundamentalmente sobre campos de tipo fecha.

Este tipo de particionamiento distribuye los datos entre las particiones en base a rangos continuos de valores de las claves de particionamiento. Los rangos de los valores de las particiones se definen mediante comparaciones de tipo “menos estricto que”.

Se ha de seguir las siguientes reglas:

- Cada partición tiene una cláusula `VALUES LESS THAN`, que especifica límite superior para la partición. Cualquier valor de la clave de partición igual o mayor a este valor límite se incluye en la siguiente partición.
- Todas las particiones, salvo la primera, tienen un límite inferior implícito que es el indicado en la cláusula `VALUES LESS THAN` de la partición que le precede.
- Se puede definir el valor `MAXVALUE` como límite superior de la última partición. `MAXVALUE` representa un valor que es superior al más alto que pueda tomar la clave de partición, incluyendo el valor `NULL`. (se puede asimilar al concepto de valor infinito en matemáticas.)

Ejemplo simple de una tabla particionada por rango:

```
CREATE TABLE time_range_sales
( prod_id      NUMBER(6)
, cust_id      NUMBER
, time_id      DATE
, channel_id   CHAR(1)
, promo_id     NUMBER(6)
, quantity_sold NUMBER(3)
, amount_sold  NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
(PARTITION SALES_1998 VALUES LESS THAN (TO_DATE('01-JAN-1999', 'DD-MON-YYYY')),
 PARTITION SALES_1999 VALUES LESS THAN (TO_DATE('01-JAN-2000', 'DD-MON-YYYY')),
 PARTITION SALES_2000 VALUES LESS THAN (TO_DATE('01-JAN-2001', 'DD-MON-YYYY')),
 PARTITION SALES_2001 VALUES LESS THAN (MAXVALUE)
);
```

Particionamiento por listas

Este tipo de particionamiento está disponible desde Oracle 9i.

El particionamiento por lista permite controlar de forma explícita como se asignan los registros a las particiones. Esto se consigue especificando una lista de valores discretos de la clave de particionamiento.

La ventaja de este método de particionamiento es que permite agrupar y organizar de una forma natural conjuntos de datos desordenados (no existe un criterio de ordenación natural) y no relacionados.

La partición `DEFAULT` permite evitar tener que especificar todos los posibles valores en la definición de una tabla particionada por listas, ya que usando la partición `default` todas las filas que no satisfagan los valores de la lista irán a esta partición sin que se genere error.

Ejemplo simple de una tabla particionada por lista:

```
CREATE TABLE list_sales
( prod_id      NUMBER(6)
, cust_id      NUMBER
, time_id      DATE
, channel_id   CHAR(1)
, promo_id     NUMBER(6)
, quantity_sold NUMBER(3)
, amount_sold  NUMBER(10,2)
)
PARTITION BY LIST (channel_id)
(PARTITION even_channels VALUES (2,4),
PARTITION odd_channels VALUES (3,9)
);
```

Particionamiento hash

Este tipo de particionamiento está disponible desde Oracle 8i.

El particionamiento hash es muy sencillo de implementar ya que su sintaxis es muy elemental. No asigna rangos a las particiones sino que utiliza un algoritmo hash sobre la clave de particionamiento para distribuir equitativamente los registros entre las particiones que se hayan definido. Este particionamiento generará particiones de aproximadamente el mismo tamaño.

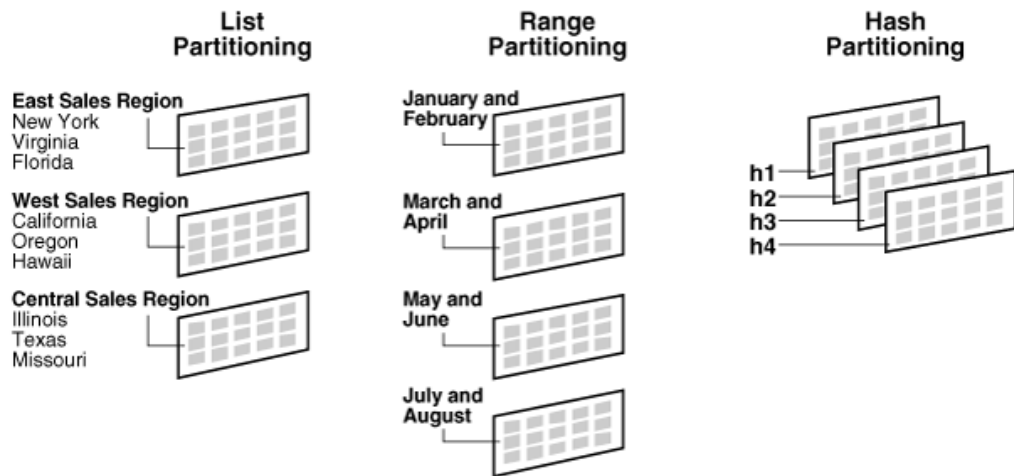
No se podrá manipular los algoritmos hash usados por el particionamiento.

El particionamiento tipo Hash es ideal para distribuir datos entre dispositivos físicos. También es usado como alternativa al particionamiento por rango, especialmente cuando los datos a particionar no son históricos o cuando no se detecta una clave de particionamiento clara.

Ejemplo simple de una tabla particionada por hash con 2 particiones:

```
CREATE TABLE hash_sales
( prod_id      NUMBER(6)
, cust_id      NUMBER
, time_id      DATE
, channel_id   CHAR(1)
, promo_id     NUMBER(6)
, quantity_sold NUMBER(3)
, amount_sold  NUMBER(10,2)
)
PARTITION BY HASH (prod_id)
PARTITIONS 4;
```

La siguiente gráfica muestra una visión física de las estrategias de particionamiento de un solo nivel:

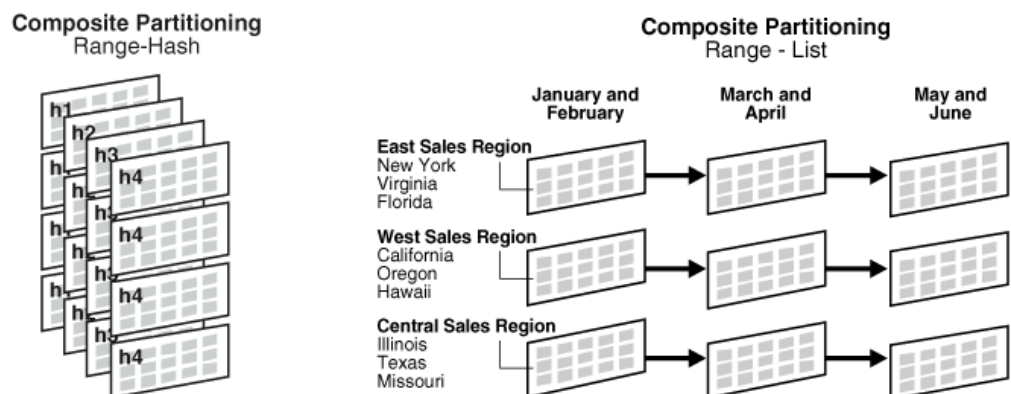


Particionamiento compuesto

El particionamiento compuesto es una combinación de los métodos de distribución fundamentales. Se generará una tabla particionada por un método de distribución que a su vez cada partición será subdividida en subparticiones usando un segundo método de distribución.

Con el particionamiento compuesto se podrá tener un grado más fino de granularidad en la distribución de los datos a través de las subparticiones.

En la siguiente gráfica se muestra una tabla particionada compuesta tipo Rango-Hash y otra tipo Rango-Lista, en este caso la tabla será particionada por rangos en la que cada partición, a su vez, estará particionada en subparticiones por hash o por una lista.



Los tipos de particionamiento compuesto son:

- Particionamiento compuesto rango-rango (Desde 11g)

Particionamiento compuesto rango-rango permite el particionamiento lógico de rangos en dos dimensiones, por ejemplo, particionar por la columna `order_date` y subparticionar por la columna `order_value`

- Particionamiento compuesto rango-hash (Desde 8i)

En el particionamiento compuesto rango-hash se particionará usando el método de rango, y dentro de cada partición, las subparticiones usarán el método hash.

Este particionamiento compuesto rango-hash proporciona la manejabilidad del particionamiento por rango y las mejoras en distribución de los datos y paralelismo del particionamiento hash.

- Particionamiento compuesto rango-lista (Desde 9i)

En el particionamiento compuesto rango-lista se particionará usando el método de rango, y dentro de cada partición, las subparticiones usarán el método lista.

El particionamiento compuesto rango-lista proporciona la manejabilidad del particionamiento por rango y el control explícito del particionamiento por lista para las subparticiones.

- Particionamiento compuesto lista-rango (Desde 11g)

El particionamiento compuesto lista-rango permite subparticionar por rango dentro de una estrategia de particionamiento por lista, por ejemplo, particionar por lista por `country_id` y subparticionar por rango por `order_date`.

- Particionamiento compuesto lista-hash (Desde 11g)

El particionamiento compuesto lista-hash permite subparticionar por hash dentro de una estrategia de particionamiento por lista.

- Particionamiento compuesto lista-lista (Desde 11g)

Particionamiento compuesto lista-lista permite el particionamiento lógico de listas en dos dimensiones, por ejemplo, particionar por `country_id` y subparticionar por `sales-channel`.

- Particionamiento compuesto hash-hash (Desde 11gR2)

Particionamiento compuesto hash-hash permite el particionamiento hash en dos dimensiones. Esto puede ser útil para favorecer el `partition-wise joins` en dos dimensiones.

- Particionamiento compuesto hash-lista (Desde 12c)

El particionamiento compuesto hash-lista permite subparticionar por lista dentro de una estrategia de particionamiento por hash.

- Particionamiento compuesto hash-rango (Desde 12c)

El particionamiento compuesto hash-rango permite subparticionar por rango dentro de una estrategia de particionamiento por hash.

También las extensiones por intervalo que se verán más adelante podrán utilizarse en particionamiento compuesto.

Extensiones de Particionamiento (A partir de 11g)

Además de las estrategias básicas de particionamiento, en versión 11g Oracle añade las siguientes extensiones de Particionamiento que se agrupan por:

- Manageability Extensions. Mejoran el manejo de tablas particionadas:

- Particionamiento por Intervalos
- Partition Advisor
- Partitioning Key Extensions. Añaden más opciones a la hora de elegir la clave de particionamiento:
 - Reference Partitioning
 - Particionamiento sobre columnas virtuales
- System Partitioning

Particionamiento por Intervalos

El particionamiento por intervalos es una extensión del particionamiento por rangos que consiste en indicar a la base de datos automáticamente que cree particiones de un específico intervalo cuando se inserten datos que exceden las particiones por rango existentes, por ejemplo particiones mensuales, que automáticamente se crean el primer día del mes.

Se deberá especificar al menos una partición tipo rango. El valor de la clave de particionamiento determinará el valor mayor del rango de particiones, que se llamará punto de transición, y la base de datos creará particiones de intervalos para datos cuyos valores son superiores al punto de transición.

Cuando se usan particiones tipo intervalo existirán restricciones:

- Sólo se puede especificar una clave de particionamiento y esta deberá ser del tipo NUMBER, FLOAT, DATE o TIMESTAMP.
- El particionamiento por intervalo no está soportado para IOTs.
- No se podrá crear un índice domain en una tabla particionada por intervalos.
- No está soportado el particionamiento por intervalos a nivel de subpartición.

Se pueden crear tablas particionadas por intervalo single-level y las siguientes tablas particionadas compuestas:

Interval-rango

Interval-hash

Interval-lista

Partition Advisor

El Partition Advisor es una funcionalidad incluida en el SQL Access Advisor. El Partition Advisor puede recomendar de forma autónoma una estrategia de particionamiento para una tabla, basándose en el estudio de un conjunto de SQL indicadas y que pueden venir de la SQL Area, un SQL Tuning Set o definidas por el usuario.

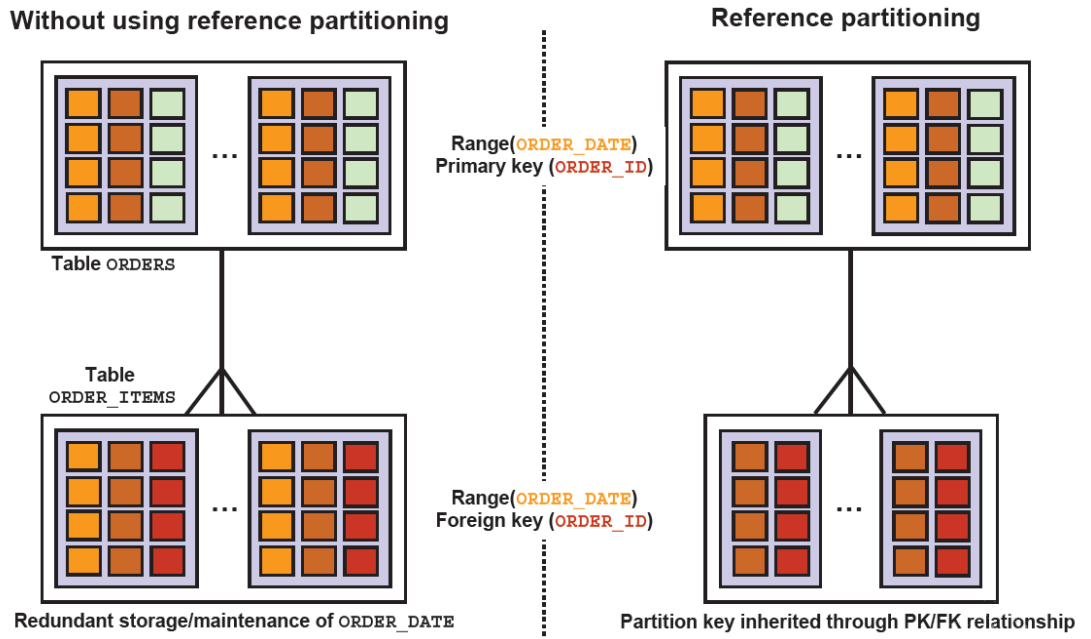
Reference Partitioning

Reference partitioning permite que una tabla sea particionada basándose en el método de particionado de la tabla referenciada por una "constraint". La clave de

particionamiento se resuelve a través de la relación padre-hijo creada a través de las constraints primary key y foreign key.

El gran beneficio de esta extensión es que las tablas con una relación padre-hijo puede estar lógicamente equiparticionadas (esto es, que tienen las mismas particiones) a través de heredar la clave de particionamiento desde la tabla padre sin tener que duplicar columnas.

Un ejemplo de Reference Partitioning se muestra en la gráfica siguiente



En la parte de la derecha se muestra la situación en la que se tienen 2 tablas ORDERS y ORDER_ITEMS que están equiparticionadas por la columna ORDER_DATE. En este caso, ambas tablas necesitan definir la columna ORDER_DATE. Sin embargo, definir ORDER_DATE en la tabla ORDER_ITEMS es redundante porque hay una relación primary key/foreign key entre las dos tablas. En la parte de la derecha se muestra la situación cuando se usa reference partitioning. En este caso no hace falta definir la columna ORDER_DATE en la tabla ORDER_ITEMS. La clave de particionamiento en la tabla ORDER_ITEMS se hereda automáticamente gracias a la relación existente de primary key /foreign key.

Desde versión 12c también es posible usar particionamiento por intervalos en las tablas padre para reference partitioning. Las particiones por intervalo serán creadas también automáticamente en la tabla hija cuando se inserten filas referenciadas que tienen particiones en la tabla padre.

Particionamiento sobre columnas virtuales

Desde versión 11g es posible definir columnas virtuales derivadas de aplicar una función o expresión a otras columnas de la propia tabla. Estas columnas podrán ser utilizadas como clave de particionamiento, esto será el particionamiento sobre columnas virtuales.

Por ejemplo si una tabla tiene una columna ACCOUNT_ID que consiste en 10 dígitos de los cuales los 3 primeros indican el número de oficina. La tabla se podría extender

con una columna virtual `ACCOUNT_BRANCH`, que venga derivada de sacar los 3 primeros dígitos de la columna `ACCOUNT_ID` y usando el Particionamiento sobre columnas virtuales, usar esta columna `ACCOUNT_BRANCH` como clave de particionamiento de la tabla.

Está soportado el uso de todas las estrategias de particionamiento de un solo nivel y compuesto en la definición del particionamiento sobre columnas virtuales.

Ejemplo:

```
create table t (c1 int,
               c2 varchar2(10),
               c3 date,
               c3_v char(1)
               generated always as
               (to_char(c3,'d')) virtual
               )
partition by list (c3_v)
(
  partition p1 values ('1'),
  partition p2 values ('2'),
  partition p3 values ('3'),
  partition p4 values ('4'),
  partition p5 values ('5'),
  partition p6 values ('6'),
  partition p7 values ('7')
);
```

System Partitioning

Permite controlar el particionamiento desde el nivel de aplicación sin que la base de datos controle el lugar donde se localizan los datos. La base de datos simplemente proporciona la habilidad de dividir la tabla en particiones pero sin conocer para que van a ser usadas estas.

Todos los aspectos de particionamiento son controlados desde la aplicación. Por ejemplo si se intenta insertar en una tabla `system partitioned` sin indicar explícitamente la especificación de partición fallará.

`System partitioning` proporciona los beneficios del particionamiento (escalabilidad, disponibilidad, manejabilidad), pero el particionamiento y la localización de los datos serán controlados por la aplicación.

No existirá clave de particionamiento.

Ejemplo de creación de una `System Partition`:

```
CREATE TABLE syspart_example (c1 number, c2 varchar2(10), c3
date)
PARTITION BY SYSTEM
( PARTITION p1 ,
  PARTITION p2 ,
  PARTITION p3 );
```

Si realizáramos una inserción convencional sobre la tabla se producirá un error del tipo:

```
SQL> insert into syspart_example values (1,'A',sysdate);
*
ERROR en línea 1:
ORA-14701: Se debe utilizar el nombre de partición ampliada o la
variable ligada para DML en tablas particionadas
```

Para realizar la inserción deberá por lo tanto utilizarse, una sentencia del tipo siguiente:

```
SQL> insert into syspart_example partition (p3) values  
(1, 'A', sysdate);  
insert into syspart_example values (1, 'A', sysdate);
```

Las sentencias delete y update no requieren la sintaxis de partition.

Este particionado puede ser útil en aplicaciones donde es posible gestionar la forma en la que se realiza el particionado (lógica de aplicación).

Para más información sobre este tipo de particionamiento ver la nota de MOS: 452447.1 11g Partitioning Enhancements.

Nuevas opciones de particionamiento a partir de 12c

Además de las estrategias básicas de particionamiento, en versión 12c Oracle añade las siguientes nuevas opciones en particionamiento:

- Particionamiento en External Tables (Desde 12.2.0.1)

Esta funcionalidad permite mejorar el rendimiento con el uso de partition pruning y partition wise join en sentencias sql sobre tablas externas. Adicionalmente permite que se capturen estadísticas a nivel de partición e incrementales en tablas externas lo que permitirá al optimizador optar por mejores planes de ejecución.

- Range Partitioning for Hash Clusters (Desde 12.1.0.2)

El particionamiento de hash clusters está soportado por rango y sólo para particionamiento single-level.

- Oracle XML DB and Domain Index Support of Hash Partitioned Tables (Desde 12.1.0.2)

Oracle XML DB y otras aplicaciones que usan Domain indexes pueden usar el particionamiento por hash a partir de 12.1.0.2.

En versión 12c para XMLIndex está soportado el uso de particionamiento hash, list, y range. En versión 12.2.0.1 también interval y reference partitioning están soportados para columnas y tablas XMLType y XMLIndex.

- Interval-Reference Partitioning (Desde 12.1.0.1)

Desde versión 12c también es posible usar particionamiento por intervalos en las tablas padre junto con la estrategia de reference partitioning. Las particiones por intervalo serán creadas también automáticamente en la tabla hija cuando se inserten filas referenciadas que tienen particiones en la tabla padre.

- Multi-column List Partitioning (Desde 12.2.0.1)

Desde versión 12cR2 es posible particionar una tabla basándose en una lista de valores para múltiples columnas. Es similar al particionamiento por lista de una sola columna pero en este caso cada partición contendrá como lista un conjunto de valores.

- Automatic List Partitioning (Desde 12.2.0.1)

Desde versión 12cR2 es posible el particionamiento por lista de forma automática. Una tabla particionada auto-list es similar a una tabla particionada por lista, excepto que es más fácil de manejar. Se puede crear una tabla particionada auto-list usando sólo los valores conocidos en ese momento de la clave de particionada. Cuando se vayan cargando datos a la tabla, la base de datos automáticamente irá creando nuevas particiones si la clave de particionamiento insertada no está en la lista definida de particiones existentes. Conceptualmente es similar al método de particionamiento por intervalos ya que las particiones son automáticamente creadas.

Indices Particionados. Tipos.

Un índice particionado es un índice que, como una tabla particionada, se descompone en trozos más pequeños y manejables. Al igual que con tablas particionadas, los índices particionados mejorarán la gestión, la disponibilidad, el rendimiento y la escalabilidad en nuestra base de datos.

Habrán diferentes tipos de índices con el uso de particionamiento:

- Índice noparticionado en una tabla particionada, también llamado índice global noparticionado.
- Índice particionado, que se divide en:
 - o Índice global. Estará particionado independientemente de la tabla donde esté creado.
 - o Índice local. Estará particionado pero las particiones estarán enlazadas con el método de particionado de una tabla.

Además los índices particionados a su vez se categorizarán en:

- Prefixed
- Nonprefixed.

Se pueden crear índices tipo bitmap en tablas particionadas. La única restricción es que solo se permiten índices bitmap de tipo local. No pueden ser índices globales. Los índices bitmap globales sólo se soportan en tablas no particionadas.

En versión 12c se añade la funcionalidad de "Partial Indexes" que permite que los índices locales o globales puedan ser creados en un subconjunto de particiones de la tabla, lo que proporciona más flexibilidad en la creación del índice.

Índices particionados locales

En un índice particionado local, el índice estará particionado por las mismas columnas, con el mismo número de particiones y con los mismos límites de particionamiento que la tabla.

Cada partición del índice se asociará exactamente a una partición de la tabla donde se cree, y todas las claves en una partición del índice apuntarán a sólo filas de una única partición de la tabla. De esta forma, la base de datos sincronizará automáticamente las particiones del índice con su partición en la tabla.

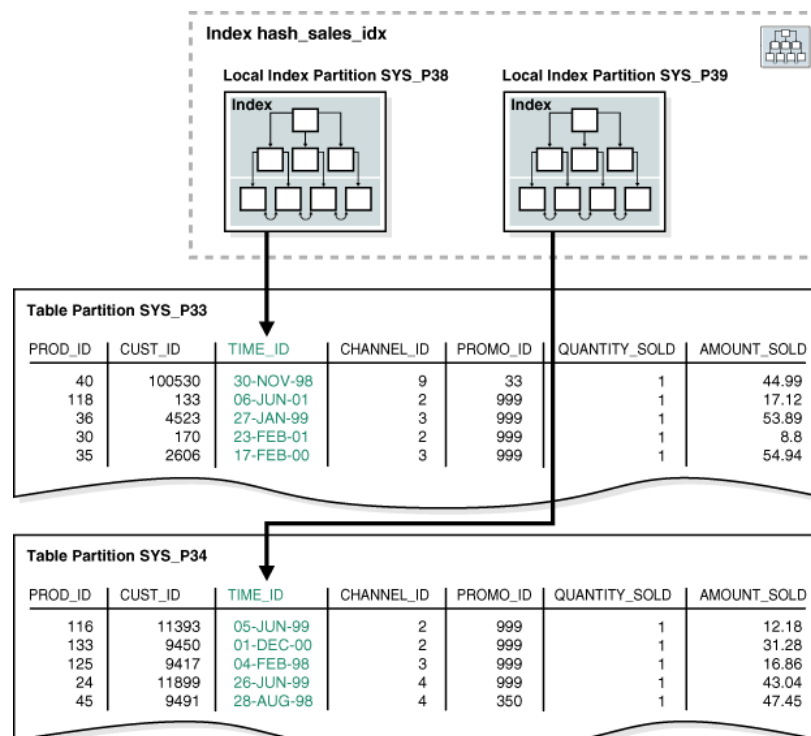
No se podrá explícitamente añadir ni borrar una partición en un índice local, esto se hará automáticamente cuando se añada o se borre en la tabla particionada.

Los índices particionados locales son comunes en entornos Data Warehouse. La disponibilidad es mayor ya que en caso de que no haya datos disponibles en una partición sólo afectará a esa partición.

Imaginemos el siguiente ejemplo de una tabla `hash_sales` particionada en 2 particiones por hash en la columna `product_id`. Podremos crear un índice local en la columna `time_id` con en el siguiente comando:

```
CREATE INDEX hash_sales_idx ON hash_sales(time_id) LOCAL;
```

Gráficamente podremos ver que el índice se divide en dos particiones cada una enlazada con una partición en la tabla. La partición del índice `SYS_P38` indexará las filas de la partición de la tabla `SYS_P33`, y la partición del índice `SYS_P39` indexará las filas de la partición `SYS_P34`:



Los índices particionados locales se dividen a su vez en:

- Local prefixed indexes

En este caso, la clave de particionamiento aparece en la parte inicial de definición del índice. En el ejemplo anterior, un índice local prefixed sería si tiene en la primera columna del índice la columna `product_id`.

- Local nonprefixed indexes

En este caso, la clave de particionamiento no aparece en la parte inicial de definición del índice y no tiene incluso que estar en la definición del índice. En el ejemplo anterior es un caso de índice local nonprefixed.

Ambos tipos tienen la ventaja de poder utilizar la eliminación de particiones (partition pruning), que ocurre cuando el optimizador mejora el rendimiento en el acceso a datos excluyendo particiones que no son necesarias. Esta eliminación de particiones dependerá del predicado de la sentencia sql. Un sentencia sql que usa un índice local prefixed siempre permitirá la eliminación de particiones en cambio un local nonprefixed no siempre.

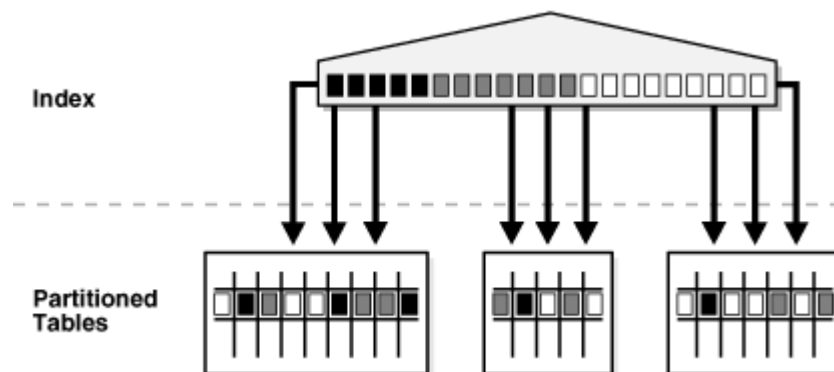
Índices particionados y no particionados globales

Un índice particionado global es un índice B-tree que está particionado independientemente del método de particionamiento de la tabla donde se cree y de su clave de particionamiento. Una única partición del índice podría apuntar a cualquier o todas las particiones de la tabla.

Se puede particionar un índice global por rango y por hash. Por hash mejorará el rendimiento bajando la contención cuando un índice crece de forma continua por un lado, esto es, las inserciones en el índice ocurren solo en el lado derecho del índice.

Un índice noparticionado global es un índice normal creado en una tabla particionada y tiene las mismas características.

En la siguiente imagen muestra una visión gráfica de un índice global noparticionado:



En general, los índices globales se usan en aplicaciones OLTP, donde el acceso rápido, la integridad de los datos y la disponibilidad son importantes. En un sistema OLTP, una tabla podría estar particionados por una clave, por ejemplo, la columna `employees.department_id`, pero una aplicación podría necesitar acceder a los

datos por muchas otras diferentes claves, por ejemplo `employee_id` o `job_id`. Índices globales podrían ser útiles en este escenario.

Algunas restricciones con el uso de particionamiento

Hay una serie de restricciones que se tiene que tener en cuenta a la hora de plantearse el particionamiento de una tabla:

- Tipos de datos. No se puede particionar una tabla que contenga alguna columna de tipo `LONG` o `LONG RAW`. Desde Oracle8i, se admite la presencia de campos de tipo `LOB` (`BLOB`, `CLOB`, `NCLOB` y `BFILE`).
- Aunque las tablas con `LOBs` pueden particionarse, las columnas con tipo `LOBs` no pueden usarse en la clave de particionamiento. Los segmentos de los `LOBs` estarán equiparticionados con la tabla, esto es, habrá tantas particiones de los `LOBs` como particiones en la tabla, además las características de los `LOBs` (`storage in row`, `tablespace`, etc) se podrán especificar a nivel de cada partición.
- No se puede particionar una tabla que forman parte de un cluster.
- Índices Bitmap. Los índices bitmap tienen como única restricción el que tienen que particionarse de forma local.
- Optimizador de costes. El optimizador de reglas no soporta particionamiento. Una aplicación que utilice reglas no se beneficiará de la eliminación de particiones, aunque las ventajas de mantenimiento seguirán estando presentes. En cualquier caso el optimizador basado en reglas no está soportado desde versión 10g.
- Restricciones físicas. Una tabla particionada no puede dispersarse en varias bases de datos. Todas sus particiones deben estar en la misma base de datos.
- Todas las particiones deberán residir en tablespaces con el mismo tamaño de bloque. Los índices u otros segmentos como segmentos `LOBs` podrán residir en tablespaces con otro tamaño de bloque.

Administrando Base de Datos con Particionamiento.

La administración del particionamiento es una importante tarea cuando se trabaja con tablas e índices particionados. En los siguientes apartados se describen diversos aspectos a la hora de crear y mantener tablas e índices particionados.

Creando Particiones

Crear una tabla o índice particionado es muy similar a la creación de una tabla o índice noparticionado, con la salvedad de que hay que incluir una cláusula de particionamiento en la sentencia `CREATE TABLE`. La cláusula y/o subcláusulas de particionamiento que se pueden incluir dependen del tipo de particionamiento que se necesite implantar.

En el informe `InfV5_JASAS_Partitioning_Development_BestPractices_V820.doc` se describen todas las opciones de particionamiento que se pueden elegir, así como la conveniencia de uno u otro tipo dependiendo de la estrategia que se quiera elegir para implantar particionamiento. Igualmente se muestran ejemplos concretos de creación de tablas.

El particionamiento es posible tanto en tablas regulares (heap organized) como en tablas organizadas por índices (IOTs), excepto para tablas que contienen columnas tipo `LONG` o `LONG RAW`.

Cuando se lanza el `create` (o `alter`) de una tabla particionada, se puede especificar la cláusula `row movement` (`ENABLE ROW MOVEMENT` o `DISABLE ROW MOVEMENT`). Esta cláusula habilita y deshabilita la migración de filas de una partición a otra en caso de que la clave de particionamiento sea modificada. Por defecto si no se indica nada, se asumirá `DISABLE ROW MOVEMENT`.

Compresión en tablas e índices particionados

Usando compresión en tablas particionadas.

Para tablas particionadas regulares (heap-organized), se puede comprimir varias o todas las particiones usando el `table compression`. El atributo `compression` puede declararse a nivel de `tablespace`, tabla o partición de una tabla. Si no se especifica el atributo se heredará como cualquier otro atributo.

Ejemplo: Creando una tabla particionada por lista con una partición comprimida.

```
CREATE TABLE costs_demo (  
    prod_id      NUMBER(6),      time_id      DATE,  
    unit_cost    NUMBER(10,2),  unit_price   NUMBER(10,2))  
PARTITION BY RANGE (time_id)  
(PARTITION costs_old
```

```
VALUES LESS THAN (TO_DATE('01-JAN-2003', 'DD-  
MON-YYYY')) COMPRESS,  
PARTITION costs_q1_2003  
VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-  
MON-YYYY')),  
PARTITION costs_q2_2003  
VALUES LESS THAN (TO_DATE('01-JUN-2003', 'DD-  
MON-YYYY')),  
PARTITION costs_recent VALUES LESS THAN (MAXVALUE));
```

Nota:

Este tipo de compresión es la referida a la estándar para operaciones Direct Load recomendable sólo para entorno DataWarehouse donde no hay mucha actividad DML sobre los objetos comprimidos. A partir de Oracle 11g, se puede elegir también la OLTP Table Compression de Oracle11g (parte del Advanced Compression Option), que es válida para cualquier entorno y cualquier tipo de operación.

Usando compresión en índices

El uso de compresión puede reducir el espacio de almacenamiento requerido para los índices. En versión 12c existen dos tipos de compresión:

- Key compression

Si no se tiene licenciado la opción Advanced Compression se puede usar la compresión tradicional de índices "key compression" o también llamada "prefix compression" para comprimir algunas o todas las particiones de un índice B-tree.

El siguiente ejemplo create un índice local particionado en todas las particiones excepto en la más reciente:

```
CREATE INDEX i_cost1 ON costs_demo (prod_id) COMPRESS LOCAL  
(PARTITION costs_old, PARTITION costs_q1_2003,  
PARTITION costs_q2_2003, PARTITION costs_recent NOCOMPRESS);
```

No se puede especificar COMPRESS (o NOCOMPRESS) explícitamente para una subpartición de un índice. Todas las subparticiones de un índice para una partición concreta heredarán la configuración de compresión indicada a la partición a la que pertenecen.

Para modificar el atributo key compression para todas las subparticiones de una partición, primero habrá que lanzar un ALTER INDEX...MODIFY PARTITION y posteriormente hacer un rebuild de todas las subparticiones.

Nota: La clausula MODIFY PARTITION marcará todas las subparticiones del índice como UNUSABLE.

La nota de MyOracleSupport *Note: 179950.1 Handling of compress option on partitioned indexes*, muestra diferentes uso de la opción COMPRESS con índices particionados.

No todos los índices son buenos candidatos a ser comprimidos con key compression, y si el número de columnas indexadas es mayor que 1 habría que añadir el rango más adecuado para que la compresión “key compression” sea efectiva.

- Advanced index compression

Advanced index compression mejora significativamente los ratios de compresión y consigue mantener un eficiente rendimiento en el acceso al índice. Incluso en índices que no son buenos candidatos a la prefix compression pueden ser comprimidos con el advanced compression.

Se puede especificar a nivel de partición el tipo de compresión o a nivel del índice cuyas particiones que no lo especifiquen heredarán los atributos de compresión. Ejemplos:

```
CREATE INDEX my_test_idx ON test(a, b) COMPRESS ADVANCED
HIGH LOCAL
(PARTITION p1 COMPRESS ADVANCED LOW,
PARTITION p2 COMPRESS,
PARTITION p3,
PARTITION p4 NOCOMPRESS);

CREATE INDEX my_test_idx ON test(a, b) NOCOMPRESS LOCAL
(PARTITION p1 COMPRESS ADVANCED LOW,
PARTITION p2 COMPRESS ADVANCED HIGH,
PARTITION p3);
```

NOTA: “Advanced Index Compress” que está disponible a partir de versión 12.1.0.2 y requiere licenciamiento extra de la opción de Advanced Compression Option.

Creación diferida de segmentos en particiones

Creación diferida de segmentos en particiones

Se puede diferir la creación de segmentos cuando se crea una tabla particionada hasta que la primera fila es insertada en la partición. De igual forma, cuando la primera fila se inserta los segmentos son creados tanto para la partición, como para las posibles columnas LOBs y para los posibles índices locales o globales.

Con el uso de esta característica se pueden tener precreadas todas las particiones sin que ocupen espacio físico en el tablespace, hasta que se inserte el primer registro en la partición. Este primer insert tardará más al necesitar crear previamente físicamente la partición.

La creación en diferido de segmentos (Deferred segment creation) se controla con:

- El parámetro de inicialización DEFERRED_SEGMENT_CREATION a TRUE o FALSE en el spfile. Por defecto a TRUE.

- Configurando el parámetro `DEFERRED_SEGMENT_CREATION` a `TRUE` o `FALSE` con un `ALTER SESSION` o `ALTER SYSTEM`.
- Especificando las palabras `SEGMENT CREATION IMMEDIATE` o `SEGMENT CREATION DEFERRED` en la cláusula de particionamiento cuando se lance el `CREATE TABLE`.

Se puede forzar a la creación de segmentos para una partición de una tabla particionada con el comando `ALTER TABLE ... MODIFY PARTITION ... ALLOCATE EXTENT`.

Nota: Esta sentencia creará una extensión más que el `initial extent` que se haya definido en el `CREATE TABLE`.

Se pueden borrar segmentos en tablas con el procedure `DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS`.

Además, si una partición o subpartición tiene un segmento, se puede lanzar un `ALTER TABLE TRUNCATE PARTITION` con la opción `DROP ALL STORAGE`, y se borrará también el segmento.

Operaciones de Mantenimiento en Tablas e Índices particionados.

En esta sección se mostrarán las operaciones más habituales de mantenimiento que se pueden realizar sobre tablas e índices particionados. Para una completa descripción de las opciones de mantenimiento así como comprobar la sintaxis exacta de las cláusulas de modificación de tablas e índices particionados, así como las posibles restricciones en su uso, se recomienda la lectura de las guías:

[Oracle® Database VLDB and Partitioning Guide 12cR2](#)

[Oracle Database SQL Language Reference](#)

Mantenimiento de los índices en tablas particionadas. Estado UNUSABLE

Por defecto, muchas operaciones en tablas particionadas marcarán `UNUSABLE` e invalidarán el índice global o las particiones concretas afectadas en caso de índice local. La mayoría de las operaciones de mantenimiento en la tabla invalidan los índices globales particionados, forzando a que se tenga que hacer un rebuild, por tanto son más difíciles de manejar que los índices particionados locales.

El que una operación marque una partición de un índice como `UNUSABLE` dependerá del tipo de partición (por ejemplo `add partition` no marca `UNUSABLE` para partición por rango, en cambio sí para `HASH`).

NOTA: Sólo los índices y particiones de índices que no están vacíos son candidatos a ser marcados como `UNUSABLE`. Si están vacíos el status `USABLE/UNUSABLE` permanecerá sin cambio.

A partir del 11.2 índices en estado UNUSABLE no consumen espacio.

Cuando un índice está marcado como UNUSABLE, estos son ignorados por el optimizador, si el parámetro SKIP_UNUSABLE_INDEXES = TRUE (por defecto así). Si se configura a FALSE, se generará un error en la ejecución de la sentencia.

Por defecto las siguientes operaciones en tablas normales (no IOTs) marcan los índices globales como unusable: ([Operations that mark global indexes unusable \(VLDB and Partitioning Guide\)](#))

```
ADD (HASH)
COALESCE (HASH)
DROP
EXCHANGE
MERGE
MOVE
SPLIT
TRUNCATE
```

En los siguientes apartados de operaciones de mantenimiento se irá indicando en cada caso cómo se comportan los índices asociados a las tablas.

A partir de versión 10g se puede evitar el que un índice se marque como UNUSABLE, si se utiliza la cláusula **UPDATE INDEXES** en la operación de alter table. Especificando esta cláusula se le dirá a la base de datos que modifique el índice (ambos índices locales y globales) a la vez que se ejecuta el comando DDL. El índice estará disponible y online mientras esté la operación y no se tendrá que reconstruir después de la operación.

Para modificar sólo los índices globales, se usará la cláusula UPDATE GLOBAL INDEXES evitando dejar los índices globales UNUSABLE, aunque sí se dejen los índices locales como UNUSABLE.

Las siguientes operaciones soportan la cláusula UPDATE INDEXES:

```
ADD          PARTITION | SUBPARTITION
COALESCE    PARTITION | SUBPARTITION
DROP        PARTITION | SUBPARTITION
EXCHANGE    PARTITION | SUBPARTITION
MERGE       PARTITION | SUBPARTITION
MOVE        PARTITION | SUBPARTITION
SPLIT       PARTITION | SUBPARTITION
TRUNCATE    PARTITION | SUBPARTITION
```

El uso de UPDATE INDEXES tiene varias implicaciones:

- Los comandos DDL tardarán más en ejecutar, ya que los índices que se deberían marcar como UNUSABLE son ahora modificados. Se debería comparar el tiempo de realizar la operación con esta cláusula respecto a lanzar el DDL sin la cláusula y después hacer un rebuild del índice. Como regla general es más rápido el uso de la cláusula si el tamaño de la partición implicada es menor del 5% del tamaño de la tabla.
- Las operaciones de DROP, TRUNCATE y EXCHANGE dejan de ser operaciones rápidas. (A partir de versión 12c una nueva funcionalidad llamada mantenimiento asíncrono de índices hacen que DROP y TRUNCATE sí sean operaciones rápidas, en el siguiente apartado se detallará la funcionalidad).

- El uso de `UPDATE INDEXES` generará información de undo y redo. En cambio cuando se hace un rebuild de un índice global, se puede usar el modo `NOLOGGING`. Además haciendo un rebuild del índice entero tendremos un índice más compacto y más eficiente.

En caso de que no se use la cláusula `UPDATE INDEXES`, si la operación de mantenimiento deja en estado `UNUSABLE` el índice local o global (particionado o no) para habrá que reconstruir los índices con `rebuild`:

- Para índices globales:
 - o Para índices globales no particionados usando `ALTER INDEX REBUILD` o borrando y creando el índice de nuevo.
 - o Para índices globales particionados usando: `ALTER INDEX ... REBUILD PARTITION` para cada partición o borrando y creando el índice de nuevo que es más eficiente ya que la tabla será escaneada sólo una vez.

- Para índices locales:

- o `ALTER INDEX ... REBUILD PARTITION/SUBPARTITION`

`ALTER INDEX ... REBUILD PARTITION` reconstruye una partición de un índice independientemente de que esté o no `UNUSABLE`. No puede usarse para tablas particionadas compuestas.

`ALTER INDEX ... REBUILD SUBPARTITION` reconstruye una subpartición de un índice en una tabla particionada compuesta.

En ambos casos, se podrán cambiar los atributos de la partición o subpartición del índice (`tablespace`, `storage`, `compression`, etc).

- o `ALTER TABLE ... MODIFY PARTITION/SUBPARTITION ... REBUILD UNUSABLE LOCAL INDEXES`

Este comando buscará todos los índices en estado `UNUSABLE` para una partición o subpartición y los reconstruye. Sólo reconstruirá aquellos que estén marcados como `UNUSABLE`.

No permite cambiar ningún atributo de la partición del índice a reconstruir.

A parte de que cuando esté marcado como `UNUSABLE` el índice o una partición de un índice, otras razones que pueden hacer necesario hacer un rebuild de un índice o particiones de un índice:

- o Recuperar espacio y mejorar el rendimiento
- o Reparar un índice o partición dañada por un fallo físico o corrupción.
- o Para reconstruir el índice de una partición tras cargar datos en la partición con `SQL*Loader` o alguna utilidad de `import`
- o Para habilitar el key `compression` de los índices `B*Tree`.

La mayoría de las operaciones de mantenimiento en la tabla invalidan los índices globales particionados, forzando a que se tenga que hacer un rebuild, por tanto son más difíciles de manejar que los índices particionados locales.

Mantenimiento asíncrono de los índices globales en operaciones de drop y truncate partitions

En versiones anteriores a 12c se podía indicar la cláusula `UPDATE INDEXES`, y de forma síncrona mantenía los índices globales en operaciones de `drop/truncate partition` pero la operación de dejaba de ser rápida.

En versión 12c se introduce una la funcionalidad (Asynchronous global index maintenance) que permite el mantenimiento del índice de forma asíncrona optimizando el `drop/truncate`.

Oracle internamente mantiene la lista de entradas del índice global asociadas a las particiones que han sido borradas y las marca como entradas `ORPHAN`. Estas entradas `ORPHAN` en el índice son limpiadas por defecto regularmente a través del job del scheduler llamado `SYS.PMO_DEFERRED_GIDX_MAINT_JOB` que limpiará todos los índices. Por defecto se ejecutará cada día las 2:00 am.

```
SQL> select owner,job_name,program_name,next_run_date,state,enabled
from dba_scheduler_jobs where job_name='PMO_DEFERRED_GIDX_MAINT_JOB';
```

JOB_NAME	STATE	ENABL	NEXT_RUN_DATE
PMO_DEFERRED_GIDX_MAINT_JOB	SCHEDULED	TRUE	16-NOV-17 02.00.00.183032 AM UTC

Se recomienda no borrar ni deshabilitar este job.

También pueden ser limpiadas manualmente de forma proactiva ejecutando manualmente el job con `DBMS_SCHEDULER.RUN_JOB`, o utilizando el procedure `DBMS_PART.CLEANUP_GIDX`, también haciendo un rebuild o un coalesce del índice. En la vista `DBA_INDEXES` hay una nueva columna `orphaned_entries` indicando si el índice tiene alguna entrada `ORPHAN`.

Añadiendo Particiones a Tablas Particionadas

Se pueden añadir nuevas particiones a una tabla particionada usando el comando

```
ALTER TABLE ... ADD PARTITION
```

Dependiendo del tipo de particionamiento usado habrá que especificar una sintaxis u otra.

- **Añadiendo particiones en una Tabla particionada por Rango:**

Se añadirá una nueva partición después de la última partición, añadiendo un valor mayor del último valor del rango indicado. Para añadir una partición al principio o mitad de la tabla habrá que usar la cláusula `SPLIT PARTITION`.

Por ejemplo, considerar la tabla, `sales`, particionada por meses, y a la que queremos añadir una nueva partición para el siguiente mes de enero:

```
ALTER TABLE sales
    ADD PARTITION jan99 VALUES LESS THAN ( '01-FEB-1999'
) TABLESPACE tsx;
```

Implicaciones:

Los índices locales y globales asociados con la tabla particionada por rango se mantienen "USABLE".

Ejemplos:

En la siguiente nota de MyOracleSupport se puede ver un ejemplo de un script para automatizar los comandos de mantenimiento en tablas particionadas por rango, no sólo para añadir particiones, también para borrar, exchange, renombrar, etc: Note:166652.1 Example of Script to Maintain Range Partitioned Table

- **Añadiendo particiones en una Tabla particionada por Hash:**

Cuando se añade una partición a una tabla particionada por hash, la base de datos tiene que completar la nueva partición moviendo filas desde particiones existentes. Por tanto, esta operación puede tardar un tiempo en completarse.

Implicaciones:

La operación puede tardar un tiempo en completarse.

En tablas regulares (HEAP) todos los índices globales serán marcados como **UNUSABLE**, a menos que se indique la opción `UPDATE INDEXES`, para índices locales, los correspondientes a la nueva partición y a las particiones cuyas filas se han redistribuido, también serán marcados como **UNUSABLE**.

Ejemplos:

En los siguientes ejemplos se muestra dos formas de añadir una partición hash a una tabla particionada, en el segundo caso, eligiendo el nombre de la partición y el tablespace donde se quiere guardar.

```
ALTER TABLE scubagear ADD PARTITION;

ALTER TABLE scubagear
    ADD PARTITION p_named TABLESPACE gear5;
```

- **Añadiendo particiones en una Tabla particionada por Lista:**

Se puede añadir una nueva partición indicando valores literales que no existan en cualquiera de las otras particiones de la tabla.

Implicaciones:

No se puede añadir una partición en una tabla particionada por lista que tenga una partición default. En este caso, se tendrá que hacer un split de la default partition, y se obtendrá una nueva partición con los valores que se indiquen y una segunda partición que continuará siendo la default partition.

Los índices locales y globales asociados con una tabla particionada por lista se mantendrán **USABLE**.

Ejemplo:

```
ALTER TABLE q1_sales_by_region
  ADD PARTITION q1_nonmainland VALUES ('HI', 'PR')
  STORAGE (INITIAL 20K NEXT 20K) TABLESPACE tbs_3
  NOLOGGING;
```

Para otras situaciones de necesidad de añadir particiones en tablas donde configuración de particionamiento es más complejo (interval, particiones compuestas, subparticiones, etc) se recomienda revisar la guía:

[Oracle® Database VLDB and Partitioning Guide](#)

Añadiendo particiones a índices particionados

No se puede explícitamente añadir una partición a un índice local, esto se hará automáticamente cuando se añada una partición a la tabla a la que pertenece el índice.

La base de datos asignará los nombres y los atributos por defecto de almacenamiento a las nuevas particiones del índice, aunque se podrán renombrar después de que la operación de `ADD PARTITION` se haya completado. Antes de añadir el índice se puede cambiar los atributos por defecto del índice, por ejemplo el default tablespace. Si no, una vez añadida si se quiere cambiar, se tendrá que hacer un rebuild de la partición del índice al tablespace deseado, por ejemplo:

```
ALTER INDEX q1_sales_by_region_locix
  REBUILD PARTITION q1_nonmainland TABLESPACE tbs_4;
```

Para índices globales, se puede añadir particiones para índices globales particionados por hash. No se puede añadir particiones a un índice global particionado por rango, ya que siempre existe la partición con un límite MAXVALUE (infinito), en este caso, habría que hacer `ALTER INDEX ... SPLIT PARTITION` para añadir una partición por encima.

Nota:

En lo siguiente cada sección se indicará con el nombre en inglés de la operación en el `ALTER TABLE`, para evitar la confusión con una traducción del comando.

Coalescing partitions

“Coalescing” es una forma de reducir el número de particiones en una tabla o índice particionado por hash, o el número de subparticiones en una tabla particionada *-hash, donde * puede ser lista, rango, intervalo o hash.

Cuando se lanza un `ALTER TABLE ... COALESCE` sobre una partición hash, su contenido es redistribuido entre una o más particiones, redistribución determinada por la función hash. Una vez que el contenido de la partición es redistribuido, esta partición es borrada.

Por ejemplo, esta sentencia reduce en uno el número de particiones en una tabla:

```
ALTER TABLE ouu1  
    COALESCE PARTITION;
```

Implicaciones:

Los índices en tablas particionadas regulares por hash tras lanzar el `coalesce` se comportan así:

- Todos los índices globales son marcados como `UNUSABLE` a menos que se utilice el `UPDATE INDEXES` con el `ALTER TABLE`
- La partición de los índices locales en la que se ha hecho el `coalesce` se borrará, y las particiones que hayan sido requeridas para reorganizar la tabla se marcarán como `UNUSABLE`.

Ejemplo:

Coalesce partition | subpartition option of the ALTER TABLE command (Doc ID 93767.1)

Drop partitions

Se pueden borrar con un `ALTER TABLE ... DROP PARTITION` y un `ALTER TABLE ... DROP SUBPARTITION` una partición y una subpartición de una tabla particionada por rango, intervalo, lista o compuesta de *-rango y *-lista.

Para tablas particionadas por intervalo, sólo se puede borrar particiones que han sido ya creadas.

Implicaciones:

Para tablas particionadas por hash o subparticiones hash de particionamiento compuestos *-hash, se deberá utilizar el comando `COALESCE`.

Si se tienen índices locales definidos en la tabla, el comando de `DROP PARTITION`, borrará la partición o subparticiones del índice que correspondan con la partición o subparticiones de la tabla que se está borrando.

Para índices globales (particionados o no), el índice o todas las particiones serán marcados como **UNUSABLE** a menos que:

- Se especifique `UPDATE INDEXES`
- La partición que está siendo borrada o sus subparticiones estuvieran vacías

Como se indicó anteriormente, en versión 12c con la introducción de la funcionalidad `asynchronous global index maintenance`, una operación de `drop partition update indexes` es sólo una operación que actúa en los metadatos y es mucho más óptima que en versiones anteriores manteniendo el índice global disponible en estado `USABLE`.

Nota:

- o Si una tabla contiene solo una partición, no se puede borrar la partición, se tendrá que borrar la tabla completa.
- o No se puede borrar la partición más alta de una tabla particionada por intervalo o una particionada compuesta intervalo-*

Cuando se quieran borrar particiones, y existan índices globales se podrán seguir distintos métodos:

1. Borrar la partición y reconstruir el índice global a continuación:

```
ALTER TABLE sales DROP PARTITION dec98;  
ALTER INDEX sales_area_ix REBUILD;
```

En este caso, si una la aplicación requiere este índice habrá que hacer una parada de servicio para realizar la operación de mantenimiento.

2. Borrar datos, y borrar la partición:

```
DELETE FROM sales partition (dec98);  
ALTER TABLE sales DROP PARTITION dec98;
```

Este método será válido cuando las tablas no sean muy grandes, ya que el `DELETE` actualizará el índice global. También dependiendo del tamaño generará gran cantidad de `UNDO` y `REDO`.

3. Especificar el `UPDATE INDEXES`

```
ALTER TABLE sales DROP PARTITION dec98  
UPDATE INDEXES;
```

Esto indicará que actúe el nuevo mecanismo de mantenimiento del índice global `asynchronous global index maintenance` y los índices permanecerán válidos.

Exchanging partitions

La cláusula `EXCHANGE PARTITION / EXCHANGE SUBPARTITION` es usada para intercambiar los segmentos de datos e índices entre:

- o Una tabla no particionada con:

- Una partición por rango, lista o hash
- Una subpartición por rango, lista o hash
- Una tabla particionada por rango con las subparticiones rango de una tabla particionada compuesta por rango-rango o lista-rango
- Una tabla particionada por hash con las subparticiones hash de una tabla particionada compuesta por rango-hash o lista-hash
- Una tabla particionada por lista con las subparticiones por lista de una tabla particionada compuesta por rango-lista o hash-lista

El `exchange partition` es muy usado cuando se tiene una aplicación que usa tablas no particionadas que se quieren convertir en particiones de una tabla particionada. Por ejemplo en un data warehouse, esta opción facilita una forma rápida de cargar datos a una tabla particionada.

Más adelante en el documento en el apartado “Métodos para convertir una tabla no particionada en particionada”, el `Exchange Partition` es uno de los métodos descritos, aunque como se indicará en este punto, existen otros métodos para convertir una tabla no particionada en particionada más recomendados en versión 12cR2.

La estructura de la tabla y la partición o subpartición que está siendo intercambiada, incluyendo sus claves de particionamiento, deberán ser idénticas. En caso de particiones y subparticiones por lista, los correspondientes valores de la lista también tienen que coincidir.

Implicaciones:

Se puede especificar la cláusula `WITHOUT VALIDATION` en la operación de `exchange`, que hace que la operación sea muy rápida ya que sólo se harán modificaciones en el diccionario de datos. Por defecto, la opción usada será `WITH VALIDATION` que hará que Oracle de un error si alguna fila de la tabla que se quiere intercambiar no pueda incluirse en las particiones o subparticiones donde se quiere intercambiar.

Si la tabla o tabla particionada tiene un clave primaria o clave única habilitada, el `exchange` será siempre realizado como si se hubiera especificado `WITH VALIDATION`. Para evitar esta comprobación, se puede deshabilitar la `constraint` antes de la operación de `exchange`. Por ejemplo:

```
ALTER TABLE table_name  
DISABLE CONSTRAINT constraint_name KEEP INDEX
```

Y tras el `exchange` volver a habilitarla.

También se puede especificar la cláusula `INCLUDING INDEXES`, para que los índices locales también sean intercambiados con el correspondiente índice de la tabla. Si no se indica la cláusula se asumirá la opción `EXCLUDING INDEXES` que hará que estos índices se marquen como `UNUSABLE`.

A menos que se indique el `UPDATE INDEXES`, la base de datos marcará como `UNUSABLE` los índices globales de la tabla cuyas particiones han sido intercambiadas.

Las estadísticas de la tabla serán intercambiadas con las de la partición que está siendo intercambiada. Sin embargo, las estadísticas globales de la tabla particionada no serán cambiadas. En versiones anteriores a 12c habría que recalculas las estadísticas globales con `DBMS_STATS.GATHER_TABLE_STATS`.

En versión 12c, se puede usar las estadísticas incrementales para evitar tener que recalculas estadísticas globales tras el uso de `partition exchange`. Cuando una tabla no particionada pasa con `exchange` a ser una partición de una tabla, Oracle automáticamente calcula las estadísticas globales usando las estadísticas de la tabla no particionada y las estadísticas existentes a nivel de partición. Esto será posible si las estadísticas fueron capturadas en el la tabla no particionada con `DBMS_STATS` con la preferencia `INCREMENTAL` puesta a `true`, y el `INCREMENTAL_LEVEL` configurado a `TABLE`.

Más adelante en el informe en el punto "Cálculo de Estadísticas para el Optimizador en tablas particionadas" se indicarán más detalles sobre cómo recalculas las estadísticas globales.

Para ver más ejemplos y opciones de `EXCHANGE` se recomienda revisar:

[Oracle® Database VLDB and Partitioning Guide](#)

Note:166652.1 Example of Script to Maintain Range Partitioned Table

Merging partitions

El comando `ALTER TABLE...MERGE PARTITIONS` une el contenido de 2 particiones en una partición.

Para el caso de tablas particionadas por rango las particiones que se unen tienen que ser adyacentes, en este caso el resultado heredará el límite superior más alto de las dos particiones. Para tablas particionadas por lista, las particiones no tienen que ser adyacentes.

Implicaciones:

Las dos particiones originales son borradas, así como los índices locales.

No se puede usar este comando para tablas particionadas por hash, en este caso habrá que usar el comando `COALESCE` visto más arriba.

Cualquier índice global en la tabla será marcado como `UNUSABLE`.

Los índices locales serán marcados como `UNUSABLE` en las particiones o subparticiones involucradas, a menos que las particiones estén vacías o que se use el `UPDATE INDEXES`.

Moving partitions/subpartitions

Se puede usar la cláusula `MOVE PARTITION` en el `ALTER TABLE` para:

- Reagrupar y reducir la fragmentación de un objeto cuyos datos han sido muy modificados.
- Mover una partición a otro tablespace
- Cambiar atributos indicados en la creación
- Guardar los datos en un formato comprimido usando el table compression

Otros atributos como algunos de almacenamiento físico de una partición se pueden modificar con el comando `ALTER TABLE/INDEX ... MODIFY PARTITION`, sin embargo algunos atributos como el `TABLESPACE`, no se puede modificar usando `MODIFY PARTITION`, habrá que usar en este caso la cláusula `MOVE PARTITION`.

Por otro lado, la modificación con `MODIFY PARTITION` de atributos, por ejemplo `table compression`, afecta a los nuevos datos, no a los existentes.

Ejemplos:

```
ALTER TABLE parts MOVE PARTITION depot2
TABLESPACE ts094 NOLOGGING COMPRESS;
```

En el ejemplo la sentencia borra el segmento de la partición y crea uno nuevo en un nuevo tablespace, además los datos los guardará en formato comprimido y no generará información de redolog.

Aunque no se indicara la opción de tablespace, se borraría y crearía un nuevo segmento en el mismo tablespace.

El siguiente ejemplo muestra cómo mover datos en una subpartición de una tabla.

En este caso, también se añade la cláusula de `PARALLEL` para paralelizar el trabajo:

```
ALTER TABLE scuba_gear MOVE SUBPARTITION bcd_types
TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

Implicaciones:

En versiones anteriores a 12c, a menos que se especifique `UPDATE INDEXES`, cuando la partición que se mueve tiene datos, el comando `MOVE PARTITION` marcará la partición del índice local correspondiente, y todos los índices globales como `UNUSABLE`. Se deberá hacer un rebuild de las particiones de los índices locales implicados con `ALTER INDEX ... REBUILD PARTITION` y de todos los índices globales, o también simplemente borrar y crear el índice tras el `MOVE`.

En versiones anteriores a 12c la cláusula `MOVE PARTITION` implica que se produzca un bloqueo en la partición implicada, evitando que se puedan realizar operaciones DML (`update`, `insert`, `delete`) en esta partición mientras se realiza el `MOVE PARTITION`, y si se quiere hacer `ONLINE` habría que implementar un mecanismo de redefinición online con el package `DBMS_REDEFINITION`.

A partir de versión 12c el `ALTER TABLE MOVE PARTITION ONLINE` es una operación sin parada de servicio ya que las operaciones DML pueden continuar sin interrupción mientras la partición está siendo movida, además los índices globales usando `ONLINE` son mantenidos durante el `move` independientemente de poner `UPDATE INDEXES`, así que no es requerido hacer el `REBUILD`.

No se puede hacer un `MOVE` de una tabla particionada completa (tanto para tablas regulares como IOTs). Se tiene que mover individualmente particiones o subparticiones.

Igualmente no se puede hacer un `move` de una partición que sea parte de una tabla particionada compuesta, se tendrá que mover cada subpartición de forma separada con la cláusula `MOVE SUBPARTITION`.

Renaming partitions/subpartitions

Con los comandos:

```
ALTER TABLE ... RENAME PARTITION  
ALTER TABLE ... RENAME SUBPARTITION  
ALTER INDEX ... RENAME PARTITION  
ALTER INDEX ... RENAME SUBPARTITION
```

Es posible renombrar los nombres de las particiones y subparticiones de tablas e índices.

La razón habitual para renombrar una partición es quererle asignarle un nombre más identificativo que los asignados automáticamente por el sistema en alguna operación de mantenimiento.

Splitting partitions

La cláusula `SPLIT PARTITION` en el `ALTER TABLE` o `ALTER INDEX` se usa para redistribuir el contenido de una partición en dos nuevas particiones.

Esta opción es usada por ejemplo, si la partición se hace muy grande, y es complicado su manejo en operaciones de `backup/recovery`, o para operaciones de mantenimiento, o por rendimiento para redistribuir la carga de I/O.

Implicaciones:

Esta clausula no puede usarse con particiones hash ni subparticiones.

Si la partición que se está dividiendo contiene datos, a menos que se use `UPDATE INDEXES`, tanto las nuevas 2 particiones de los índices locales como todos los índices globales (particionados o no) serán marcados como `UNUSABLE`.

Para una tabla particionada por lista, se pueden especificar una lista de valores que serán los que se muevan a una partición, las filas con valores que no se indiquen irán a la otra partición.

En versión Oracle 12cR2 se introduce a opción de poder realizar la cláusula `ONLINE` el `SPLIT PARTITION` y `SPLIT SUBPARTITION`, lo que permitirá operaciones DML mientras la operación de mantenimiento se realiza. Además cuando se indica `ONLINE`, aunque no se especifique `UPDATE INDEXES`, los índices globales y locales serán automáticamente mantenidos y no serán puestos en estado `UNUSABLE`.

Ejemplos:

```
ALTER TABLE sales_by_region
  SPLIT PARTITION region_east VALUES ('CT', 'MA', 'MD')
  INTO
  ( PARTITION region_east_1
    TABLESPACE tbs2,
    PARTITION region_east_2
    STORAGE (INITIAL 8M))
  PARALLEL 5;
```

En este ejemplo, la partición `region_east` se divide en 2 particiones, la primera con las filas con los valores `'CT'`, `'MA'`, `'MD'` y la segunda con las filas con los valores restantes de la definición inicial de la partición inicial.

Cuando se quiere añadir una partición a la partición por defecto en una tabla particionada por lista, se usará la cláusula `split`. Cuando se hace el `split` de la partición por defecto, se creará una nueva partición definida por los valores que se indiquen, y una segunda partición que seguirá siendo la partición por defecto.

El siguiente ejemplo muestra cómo dividir la partición por defecto de la tabla `sales_by_region`, creando una nueva partición:

```
ALTER TABLE sales_by_region
  SPLIT PARTITION region_unknown VALUES ('MT', 'WY', 'ID')
  INTO
  ( PARTITION region_wildwest,
    PARTITION region_unknown);
```

El siguiente ejemplo muestra cómo dividir la partición `sales_q4_2016` en particiones por mes, además se añade la cláusula `ONLINE` que permitirá operaciones DMLs concurrentes y el mantenimiento automático de los índices.

```
CREATE TABLE orders
  (prod_id      NUMBER(6),
   cust_id      NUMBER,
   time_id      DATE,
   channel_id   CHAR(1),
   promo_id     NUMBER(6),
   quantity_sold NUMBER(3),
   amount_sold  NUMBER(10,2)
  )
  PARTITION BY RANGE (time_id)
  (PARTITION sales_q1_2016 VALUES LESS THAN (TO_DATE('01-APR-2016','dd-MON-yyyy')),
   PARTITION sales_q2_2016 VALUES LESS THAN (TO_DATE('01-JUL-2016','dd-MON-yyyy')),
   PARTITION sales_q3_2016 VALUES LESS THAN (TO_DATE('01-OCT-2016','dd-MON-yyyy')),
   PARTITION sales_q4_2016 VALUES LESS THAN (TO_DATE('01-JAN-2017','dd-MON-yyyy')));
```

```
);  
  
ALTER TABLE orders  
  SPLIT PARTITION sales_q4_2016 INTO  
  (PARTITION sales_oct_2016 VALUES LESS THAN (TO_DATE('01-  
NOV-2016', 'dd-MON-yyyy'))),  
  PARTITION sales_nov_2016 VALUES LESS THAN (TO_DATE('01-  
DEC-2016', 'dd-MON-yyyy'))),  
  PARTITION sales_dec_2016  
  )  
  ONLINE;
```

Más ejemplos de uso de SPLIT:

Note:166652.1 Example of Script to Maintain Range Partitioned Table

<https://docs.oracle.com/database/122/VLDBG/maintenance-partition-tables-indexes.htm#GUID-C27329F1-00C6-4952-A411-4B9A278E4DE8>

Truncating partitions/subpartitions

Con el ALTER TABLE ... TRUNCATE PARTITION se borrarán todas las filas de la partición, de forma similar a si se borrara la partición, excepto que sólo borra los datos pero no es borrada la partición físicamente. Si la tabla es particionada compuesta se borrarán todas las subparticiones de la partición, o si se usa TRUNCATE SUBPARTITION se podrá limitar a sólo una subpartición.

Implicaciones:

Si la tabla contiene columnas LOBS, tanto los segmentos de datos e índices de LOB para esa partición también serán truncados. Si la tabla es particionada compuesta, todas las subparticiones de los segmentos de LOB para la partición serán truncadas.

Si existen constraints para asegurar la integridad referencial, habrá que deshabilitar las constraints antes de lanzar el truncate, o de forma alternativa borrar los datos con delete y posteriormente lanzar el truncate:

```
DELETE FROM sales partition (dec94);  
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

Esta segunda alternativa es más adecuada para tablas pequeñas, ya que el delete generará mucha información de redo y undo.

Por cada partición o subpartición que sea truncada, Oracle también hará truncate de su correspondiente partición del índice local y subparticiones, y cambiará la etiqueta UNUSABLE a VALID. Para índices globales el comportamiento será que si la partición o subpartición que está siendo truncada tiene datos, los índices globales serán marcados UNUSABLE a menos que se indique la cláusula UPDATE INDEXES.

Como se indicó anteriormente, en versión 12c con la introducción de la funcionalidad asynchronous global index maintenance, una operación de "truncate partition update indexes" es sólo una operación que actúa en los metadatos y es mucho más óptima que en versiones anteriores manteniendo el índice global disponible en estado USABLE.

En caso de que tengamos índices globales tendremos los siguientes métodos para hacer truncate:

- Método 1: Lanzar el truncate y posteriormente un rebuild del índice global

Ejemplo:

```
ALTER TABLE sales TRUNCATE PARTITION dec98;  
ALTER INDEX sales_area_ix REBUILD;
```

Este es el método más apropiado en versiones anteriores a 12c para tablas grandes cuando la partición que está siendo truncada contiene un porcentaje importante del total de datos de la tabla.

A partir de 12c si la disponibilidad de la aplicación depende del tiempo en que se hace el REBUILD del índice se recomienda el método 3.

- Método 2: Lanzar el delete de todas las filas antes de hacer el TRUNCATE .

Ejemplo:

```
DELETE FROM sales PARTITION (dec98);  
ALTER TABLE sales TRUNCATE PARTITION dec98;
```

En este caso, el delete mantendrá el índice global, ejecutará triggers y generará undo y redo. Este método es más apropiado para tablas pequeñas, o tablas grandes cuando la partición contiene un porcentaje pequeño del total de datos de la tabla.

- Método 3: Lanzar el truncate con la opción UPDATE INDEXES

```
ALTER TABLE sales TRUNCATE PARTITION dec98  
UPDATE INDEXES;
```

En versiones anteriores a 12c actuará de forma bastante similar al método 2, en ese caso como regla general es útil usar esta cláusula si el tamaño de la partición implicada es menor del 5% del tamaño de la tabla.

En versiones superiores a 12c gracias a la funcionalidad asynchronous global index maintenance esta operación será sólo a nivel de metadatos y será mucho más óptima evitando el rebuild del índice.

Opcionalmente se puede especificar la cláusula DROP STORAGE para que se libere el espacio físico ocupado por las filas borradas excepto el espacio reservado indicado por el parámetro MINEXTENTS .

Con la opción DROP ALL STORAGE (a partir de 11.2.0.2) se liberará también el espacio indicado por MINEXTENTS (opción Deferred Segment Creation).

Operaciones en múltiples particiones

En versión 12c se añade la funcionalidad de poder ejecutar operaciones de mantenimiento en múltiples particiones/subparticiones simplificando las operaciones a realizar y haciendo más eficiente el mantenimiento.

Esta funcionalidad permite añadir múltiples particiones a una tabla, borrar múltiples particiones, realizar merge múltiples particiones en una partición, hacer split un única partición en múltiples particiones y hacer un truncate de múltiples particiones con sólo un comando SQL. Las mismas operaciones sobre subparticiones.

El mantenimiento de los índices asociados se comportará de la misma forma de cuando la operación de mantenimiento sólo actúa sobre una partición.

Por ejemplo se pueden añadir múltiples particiones por rango indicando el límite superior de cada partición (después de la última partición existente) en una tabla particionada por rango con este comando:

```
ALTER TABLE sales ADD
PARTITION sales_q1_2007 VALUES LESS THAN (TO_DATE('01-APR-
2007','dd-MON-yyyy')),
PARTITION sales_q2_2007 VALUES LESS THAN (TO_DATE('01-JUL-
2007','dd-MON-yyyy')),
PARTITION sales_q3_2007 VALUES LESS THAN (TO_DATE('01-OCT-
2007','dd-MON-yyyy')),
PARTITION sales_q4_2007 VALUES LESS THAN (TO_DATE('01-JAN-
2008','dd-MON-yyyy'))
;
```

O por ejemplo borrar múltiples particiones con un único comando:

```
ALTER TABLE sales DROP PARTITION sales_q1_2008,
sales_q2_2008,sales_q3_2008, sales_q4_2008;
```

Filtering

En versión 12cR2 se añade la funcionalidad de filtro de datos en operaciones de mantenimiento de particionamiento que permite además de realizar tareas de mantenimiento de datos filtrar los datos que queremos que permanezcan en la tabla, borrando los que no cumplan el filtro.

La funcionalidad de filtro de datos aplicará a operaciones de MOVE PARTITION, MERGE PARTITION, y SPLIT PARTITION.

Si se especifica la opción ONLINE, operaciones DMLs serán permitidas mientras se realiza la operación, además el filtro sólo aplicará a los datos al principio de lanzar el comando de mantenimiento, los datos insertados por DMLs posteriores ignorarán el filtro.

Ejemplo:

Suponemos una tabla particionada, donde insertamos datos:

```
SQL> CREATE TABLE test_part_tab
(
  Col_Num    NUMBER(5)    NOT NULL
  ,Col_Chr   CHAR(5)      NOT NULL
```

```
),Col_Dat DATE
)
PARTITION BY RANGE (Col_Num)
(
    PARTITION P_LESS20 VALUES LESS THAN (20)
    ,PARTITION P_LESS40 VALUES LESS THAN (40)
) ;
```

```
SQL> INSERT INTO test_part_tab VALUES (1,'A1', SYSDATE) ;
SQL> INSERT INTO test_part_tab VALUES (2,'A2', SYSDATE) ;
SQL> INSERT INTO test_part_tab VALUES (3,'A2', SYSDATE) ;
SQL> INSERT INTO test_part_tab VALUES (21,'A2', SYSDATE) ;
SQL> commit;
```

```
SQL> select col_num from test_part_tab;
COL_NUM
-----
      1
      2
      3
     21
```

Podemos hacer un `move` de la partición indicando que sólo mantenga ciertas filas en la partición con un filtro:

```
SQL> ALTER TABLE test_part_tab MOVE PARTITION P_LESS20 ONLINE
TABLESPACE users
INCLUDING ROWS WHERE Col_Num <=2;

SQL> select col_num from test_part_tab;

COL_NUM
-----
      1
      2
     21
```

Vistas del Data Dictionary para el manejo del Particionamiento

Podemos confirmar la estructura del objeto particionado examinando varias vistas de diccionario, las principales:

- `DBA/ALL/USER_PART_TABLES` - Esta vista mostrará como la tabla está particionada, cuantas particiones existen y cuáles son las columnas que componen la clave de particionamiento.
- `DBA/ALL/USER_PART_KEY_COLUMNS` - Esta vista mostrará detalles sobre la clave de particionamiento.
- `DBA/ALL/USER_TAB_PARTITIONS`- Esta vista muestra las particiones individualmente y los detalles de la configuración de cada partición dependiendo del tipo de particionamiento elegido.

- DBA/ALL/USER_TAB_SUBPARTITIONS. Mostrará información a nivel de subparticionamiento, los parámetros de almacenamiento de cada subpartición y las estadísticas de subpartición generadas por el cálculo de estadísticas (DBMS_STATS).

Ejemplos:

```
SELECT table_name, partitioning_type "Type", partition_count "Count",
partitioning_key_count "Key count" from user_part_tables WHERE
table_name='EMPLEADOS';
```

TABLE_NAME	Type	Count	Key count
EMPLEADOS	RANGE	4	1

```
SELECT * FROM user_part_key_columns WHERE name='EMPLEADOS';
NAME OBJEC COLUMN_NAM COLUMN_POSITION
EMPLEADOS TABLE EMPID 1
```

```
SELECT table_name, partition_name, high_value, partition_position
FROM user_tab_partitions
WHERE table_name='EMPLEADOS'
ORDER BY 2;
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE	PARTITION_POSITION
EMPLEADOS	P_EMPLEADO1	250	1
EMPLEADOS	P_EMPLEADO2	500	2
EMPLEADOS	P_EMPLEADO3	750	3
EMPLEADOS	P_EMPLEADO4	1000	4

```
select table_name, partition_name, subpartition_name, tablespace_name
from dba_tab_subpartitions
where table_name='TAB2';
```

TABLE_NAME	PARTITION_NAME	SUBPARTITION_NAME	TABLESPACE_NAME
TAB2	Q1	Q1_H1	DATOS1
TAB2	Q1	Q1_H2	DATOS2
TAB2	Q1	Q1_H3	DATOS3
TAB2	Q1	Q1_H4	DATOS4
TAB2	Q2	Q2_H5	DATOS5
TAB2	Q2	Q2_H6	DATOS6
TAB2	Q2	Q2_H7	DATOS7
TAB2	Q2	Q2_H8	DATOS8
TAB2	Q3	Q3_H1	DATOS1
TAB2	Q3	Q3_H2	DATOS2
TAB2	Q3	Q3_H3	DATOS3
TAB2	Q3	Q3_H4	DATOS4
TAB2	Q4	Q4_H5	DATOS5
TAB2	Q4	Q4_H6	DATOS6
TAB2	Q4	Q4_H7	DATOS7
TAB2	Q4	Q4_H8	DATOS8

En la siguiente tabla tenemos todas las vistas del diccionario que contienen la información más relevante del particionamiento de tablas e índices:

View	Description
DBA_PART_TABLES ALL_PART_TABLES USER_PART_TABLES	DBA view displays partitioning information for all partitioned tables in the database. ALL view displays partitioning information for all partitioned tables accessible to the user. USER view is restricted to partitioning information for partitioned tables owned by the user.
DBA_TAB_PARTITIONS ALL_TAB_PARTITIONS USER_TAB_PARTITIONS	Display partition-level partitioning information, partition storage parameters, and partition statistics generated by the DBMS_STATS package or the ANALYZE statement.
DBA_TAB_SUBPARTITIONS ALL_TAB_SUBPARTITIONS USER_TAB_SUBPARTITIONS	Display subpartition-level partitioning information, subpartition storage parameters, and subpartition statistics generated by the DBMS_STATS package or the ANALYZE statement.
DBA_PART_KEY_COLUMNS ALL_PART_KEY_COLUMNS USER_PART_KEY_COLUMNS	Display the partitioning key columns for partitioned tables.
DBA_SUBPART_KEY_COLUMNS ALL_SUBPART_KEY_COLUMNS USER_SUBPART_KEY_COLUMNS	Display the subpartitioning key columns for composite-partitioned tables (and local indexes on composite-partitioned tables).
DBA_PART_COL_STATISTICS ALL_PART_COL_STATISTICS USER_PART_COL_STATISTICS	Display column statistics and histogram information for the partitions of tables.
DBA_SUBPART_COL_STATISTICS ALL_SUBPART_COL_STATISTICS USER_SUBPART_COL_STATISTICS	Display column statistics and histogram information for subpartitions of tables.
DBA_PART_HISTOGRAMS ALL_PART_HISTOGRAMS USER_PART_HISTOGRAMS	Display the histogram data (end-points for each histogram) for histograms on table partitions.
DBA_SUBPART_HISTOGRAMS ALL_SUBPART_HISTOGRAMS USER_SUBPART_HISTOGRAMS	Display the histogram data (end-points for each histogram) for histograms on table subpartitions.
DBA_PART_INDEXES ALL_PART_INDEXES	Display partitioning information for partitioned indexes.

View	Description
USER_PART_INDEXES	
DBA_IND_PARTITIONS	Display the following for index partitions: partition-level partitioning information, storage parameters for the partition, statistics collected by the DBMS_STATS package or the ANALYZE statement.
ALL_IND_PARTITIONS	
USER_IND_PARTITIONS	
DBA_IND_SUBPARTITIONS	Display the following information for index subpartitions: partition-level partitioning information, storage parameters for the partition, statistics collected by the DBMS_STATS package or the ANALYZE statement.
ALL_IND_SUBPARTITIONS	
USER_IND_SUBPARTITIONS	
DBA_SUBPARTITION_TEMPLATES	Display information about existing subpartition templates.
ALL_SUBPARTITION_TEMPLATES	
USER_SUBPARTITION_TEMPLATES	

Métodos para convertir una tabla no particionada en particionada

Cuando tenemos una tabla noparticionada que quiere ser particionada, esta tendrá que reconstruirse. A continuación se describen varios métodos para llevar a cabo esta tarea, indicando en cada caso las ventajas e inconvenientes más importantes:

1. Export/Import (Datapump o export/import tradicional)

Pasos básicos:

- Exportar la tabla
- Borrar la tabla origen
- Recrear la tabla con la estructura particionada
- Importar los datos por ejemplo con import:

```
$ imp usr/pswd file=exp.dmp ignore=y
```
- Recrear los objetos dependientes (índices, constraints, triggers)
- Calcular estadísticas

Ventajas/Inconvenientes:

Es un método simple, pero el tiempo de parada de aplicación será alto directamente proporcional al tamaño de la tabla, ya que las operaciones de export/import pueden tardar.

2. Insertar datos con una subquery sobre la tabla

Pasos básicos:

- Crear la tabla particionada con otro nombre
- Insertar datos en la nueva tabla con una subquery de la tabla no particionada

```
insert into partbl (qty, name) select * from  
origtbl;
```

- Borrar la tabla original y renombrar la tabla particionada con su nombre.
- Recrear índices, constraints, triggers, etc en la nueva tabla. (Este paso podría hacerse previamente con otros nombres).
- Calcular estadísticas

Ventajas/Inconvenientes:

Es un método simple, pero el tiempo de parada de aplicación será también alto si la aplicación necesita hacer operaciones DML en la tabla origen. Si sólo son consultas, el tiempo de parada se limitaría al tiempo de renombrar la tabla y objetos relacionados. Además el uso de UNDO y REDO será alto en las operaciones de insert select.

3. Usando SQL*Loader

Pasos básicos:

- Descargar los datos de la tabla origen a ficheros planos uno por partición
- Borrar la tabla origen
- Recrear la tabla con la estructura particionada
- Cargar los datos con Sqlloader desde cada fichero a la partición correspondiente.
- Recrear los objetos dependientes (índices, constraints, triggers)
- Calcular estadísticas

Ventajas/Inconvenientes:

Es un método más complejo y manual. Habrá tiempo de parada dependiendo del tiempo que tarden las cargas de datos. La ventaja principal es que se pueden usar opciones de NOLOGGING, DIRECT, PARALLEL de SQL*Loader para minimizar el uso de buffer cache, redo, etc y mejorar el rendimiento.

4. Usando la opción Partition Exchange del ALTER TABLE

Pasos básicos:

- Crear la tabla particionada

Ejemplo:

```
SQL> CREATE TABLE p_emp  
  (sal NUMBER(7,2))  
  PARTITION BY RANGE(sal)  
  (partition emp_p1 VALUES LESS THAN (2000),  
   partition emp_p2 VALUES LESS THAN (4000));
```

- Crear tablas puente con los datos de cada partición desde la tabla origen

```
SQL> CREATE TABLE dummy_y as SELECT sal FROM emp WHERE  
sal<2000;
```

```
SQL> CREATE TABLE dummy_z as SELECT sal FROM emp WHERE sal  
BETWEEN 2000 AND 3999;
```

- Hacer exchange desde cada tabla puente a cada partición de la nueva tabla

```
SQL> alter table p_emp exchange partition emp_p1 with  
table dummy_y;
```

```
SQL> alter table p_emp exchange partition emp_p2 with  
table dummy_z;
```

- Borrar la tabla origen y renombrar la tabla particionada con su nombre original
- Recrear índices, constraints, triggers, etc en la nueva tabla.
- Calcular estadísticas

Ventajas/Inconvenientes:

Es un método simple, pero el tiempo de parada de aplicación será alto si la aplicación necesita hacer operaciones DML en la tabla origen. Si sólo son consultas, el tiempo de parada se limitaría al tiempo de renombrar la tabla y objetos relacionados.

5. Usando DBMS_REDEFINITION

Pasos básicos:

- Se crea la tabla particionada con otro nombre con un nombre provisional.
- Se ejecuta el procedure `DBMS_REDEFINITION.can_redef_table` para ver si es posible el proceso.
- Se ejecuta el procedure `DBMS_REDEFINITION.start_redef_table` en el que se creará una vista materializada basada en un `CREATE TABLE AS SELECT` sobre la tabla particionada, dependiendo del tamaño de la tabla origen este proceso tardará.
- Se ejecuta el procedure `DBMS_REDEFINITION.sync_interim_table` para sincronizar las tablas con las transacciones que estén ocurriendo (como un `MVIEW FAST REFRESH`)
- Aquí se pueden crear índices/constraints sobre la tabla particionada. (Los nombres tendrán que ser diferentes a los originales)
- Calcular estadísticas para el optimizador
- Se ejecuta `DBMS_REDEFINITION.finish_redef_table` para hacer el cambio de roles entre las tablas, la tabla particionada es ahora nombrada con el nombre de la tabla original. Durante este proceso habrá un bloqueo breve sobre la tabla original.
- Borrar la tabla original
- De forma opcional renombrar los nombres de los índices/constraints con los nombres originales.

Ventajas/Inconvenientes:

La ventaja principal es que permite minimizar el tiempo de parada de una aplicación, proporcionando un mecanismo de traspaso de datos y sincronización entre tablas que puede ser realizado de forma concurrente con operaciones sobre la tabla. El tiempo de parada es mínimo.

El inconveniente principal es que dependiendo de la versión existen distintas restricciones en el uso de `DBMS_REDEFINITION`.

Se recomienda la revisión de las notas de MOS:

How To Partition Existing Table Using `DBMS_Redefinition` (Doc ID 472449.1)

`DBMS_REDEFINITION`: Case Study for a Large Non-Partition Table to a Partition Table with Online Transactions occurring (Doc ID 1481558.1)

6. Usando cláusula `MODIFY` en el `ALTER TABLE`

Desde versión Oracle 12cR2 se puede usar la cláusula `MODIFY` en el `ALTER TABLE` para convertir una tabla no particionada en particionada y con la opción de hacerlo `ONLINE` permitiendo de forma concurrente sentencias DML sobre la tabla y manteniendo los índices en modo `USABLE`.

Además se puede usar la cláusula `UPDATE INDEXES` para cambiar el estado de particionamiento de los índices o las propiedades de almacenamiento en el momento de ser convertida la tabla, aunque no pueden ser cambiadas las columnas indexadas. Si no se especifica nada en el `UPDATE INDEXES`, dependiendo de los índices previos estos serán convertidos a un índice global no particionado non-prefixed, a un índice local prefixed, o a índice global particionado. El comportamiento por defecto en cada caso está descrito en la guía: [Oracle® Database VLDB and Partitioning Guide 12cR2](#)

Ventajas/Inconvenientes:

Este método permite con un solo comando y sin pérdida de servicio convertir una tabla no particionada en particionada, evitando las múltiples etapas y complejidad de los métodos `EXCHANGE PARTION` y `DBMS_REDEFINITION`

Es el método recomendado en versión 12cR2.

Algunas restricciones asociadas a esta funcionalidad:

- No se puede usar para particionar una index-organized table (IOT).
- No se puede usar si la tabla tiene un domain index.
- Sólo se puede usar el modo offline del método para la conversión de una tabla a una tabla particionada con `reference`.

Cálculo de Estadísticas para el Optimizador en tablas particionadas.

El optimizador de consultas de Oracle para seleccionar un buen plan de ejecución de una sentencia SQL se basa en las estadísticas que haya calculadas, tanto en las estadísticas de toda la tabla (las estadísticas globales) como en las estadísticas de las particiones individuales (las estadísticas de partición).

Cuando una consulta accede a una única partición con estadísticas a nivel de partición obsoletas o inexistentes el plan de ejecución puede no resultar todo lo eficiente que se necesita debido en gran parte a que los valores de cada partición no están delimitados.

Si la consulta tiene que acceder a una única partición (partition pruning), el optimizador utiliza sólo las estadísticas de acceso a la partición. Si el acceso es a más de una partición, se utiliza una combinación de las estadísticas globales y de la partición.

La recopilación de estadísticas globales es extremadamente costosa en términos de tiempo y recursos del sistema.

Es recomendable mantener las estadísticas de cada una de las particiones de las tablas. En entornos de DW es muy común realizar cargas de datos directamente sobre las particiones, si el volumen de datos es muy grande el proceso de recreación de las estadísticas puede resultar muy costoso. Tradicionalmente el proceso de recolección de las estadísticas se realiza en dos pasos:

- En el primero, se recrean las estadísticas globales de toda la tabla.
- En un segundo paso se recrean las estadísticas de aquellas particiones que han sufrido cambios.

El proceso de análisis de la tabla varía en función del tamaño de la tabla. Ha de tenerse en cuenta que se realiza una exploración de toda la tabla incluso aunque el cambio en los datos sea mínimo.

Para determinar el tipo de estadísticas de particionamiento que son recogidas se usa el argumento `GRANULARITY` del procedure `DBMS_STATS`. Oracle recomienda configurar el parámetro a `AUTO` (por defecto), que hará que dependiendo del tipo de particionamiento, se calculen estadísticas para subparticiones, particiones y estadísticas globales. Otras opciones del argumento en versión 12cR2:

'ALL' - gathers all (subpartition, partition, and global) statistics

'AUTO' - determines the granularity based on the partitioning type. This is the default value.

'DEFAULT' - gathers global and partition-level statistics. This option is obsolete, and while currently supported, it is included in the documentation for legacy reasons only. You should use the 'GLOBAL AND PARTITION' for this functionality. Note that the default value is now 'AUTO'.

'GLOBAL' - gathers global statistics

'GLOBAL AND PARTITION' - gathers the global and partition level statistics. No subpartition level statistics are gathered even if it is a composite partitioned object.

'PARTITION' - gathers partition-level statistics

'SUBPARTITION' - *gathers subpartition-level statistics*

Buenas prácticas en el cálculo de estadísticas con tablas particionadas

- Asegurar que se están capturando estadísticas para las tablas particionadas (bien con el proceso automático o con método personalizado). Dynamic sampling no es una buena solución para grandes volúmenes de datos típicas en tablas sobre entornos de DW.
- Si es viable, una buena práctica es ejecutar las consultas sobre una tabla vacía (antes de la carga de datos) de esa forma se obtiene información útil sobre la utilización de las columnas de la tabla, que será usada para calcular de forma más efectiva estadísticas después.
- Las estadísticas deberían recrearse después de la carga de datos en una tabla.
- Si se hace un rebuild o un create de un índice no falta calcularlas para los índices (Oracle automáticamente genera estadísticas para los índices cuando se lleva a cabo el proceso de creación o rebuild de estos últimos).
- A partir de versión 11g debería utilizarse “auto sample size”, esta opción incorpora un nuevo algoritmo hash muy adecuado para obtener el Número de Valores Distintos (NDV), este modo de generación de estadísticas no emplea grandes cantidades de tiempo. En versión anterior sobre 10g este método no era eficiente y se recomendaba si era viable por la ventana de mantenimiento utilizar como “sample size” el 100%, o al menos el 30%.
- Como se ha comentado, las tablas particionadas se benefician de tener buenas estadísticas globales calculadas, pero las estadísticas globales son costosas de calcular. A partir de 11g (y por regresión en últimos patchsets de 10gR2) existe la opción de generación de estadísticas incrementales en una tabla particionada, que permite tener actualizadas las estadísticas globales de una tabla escaneando en el proceso de captura de estadísticas sólo las particiones que han sido modificadas y evitando tener que escanear toda la tabla.

Estadísticas Incrementales

Con tablas particionadas, típicamente lo que se hace es insertar nuevos datos en una nueva partición. Como se ha recomendado, se debería calcular estadísticas para la nueva partición y tener actualizadas las estadísticas globales.

Para mejorar el tiempo en el cálculo de estadísticas, se puede usar la opción INCREMENTAL y evitar tener que hacer un *full table scan* de toda la tabla para mantener las estadísticas globales.

Para poder usar la opción de estadísticas incrementales varias condiciones tienen que cumplirse:

- El valor para la preferencia INCREMENTAL de DBMS_STATS para la tabla particionada es true (por defecto es false).
- El valor para la preferencia PUBLISH de DBMS_STATS para la tabla particionada es true (por defecto es true).
- Las estadísticas en la tabla son recogidas con AUTO_SAMPLE_SIZE para el ESTIMATE_PERCENT y AUTO para GRANULARITY.

Cuando todos estos criterios se cumplen, las estadísticas globales se calculan incrementalmente escaneando sólo las particiones que han sido modificadas.

Para comprobar la configuración de las preferencias INCREMENTAL y PUBLISH:

```
SQL> select
DBMS_STATS.GET_PREFS('INCREMENTAL','<schema_name>','<table_name>')
from dual;
```

```
SQL> select
DBMS_STATS.GET_PREFS('PUBLISH','<schema_name>','<table_name>') from
dual;
```

Para cambiarlos usar:

```
exec DBMS_STATS.SET_TABLE_PREFS ('<owner_name>',
'<table_name>', 'INCREMENTAL', 'TRUE');
```

```
exec DBMS_STATS.SET_TABLE_PREFS ('<owner_name>',
'<table_name>', 'PUBLISH', 'TRUE');
```

Si se deja el valor INCREMENTAL para la tabla particionada a FALSE (valor por defecto), entonces se usará full table scan para mantener las estadísticas globales, que puede ser una operación muy costosa en tablas grandes.

Cuando las estadísticas se calculan con la opción incremental hay varias consecuencias:

- El tablespace SYSAUX consume espacio adicional para mantener estas estadísticas globales para las tablas particionadas
- Si la tabla es particionada compuesta, la base de datos sólo calcula estadísticas para las subparticiones modificadas, esto es, a nivel de subpartición no se calcularán estadísticas en subparticiones que no se han modificado, reduciendo el trabajo ignorando estas particiones.
- Si la tabla usa estadísticas incrementales, y la tabla tiene un índice local particionado, la bdd calculará estadísticas globales para el índice y para las particiones del índice modificadas. La base de datos no generará las estadísticas globales del índice desde las estadísticas a nivel de partición del índice, en lugar de esto la base de datos calcula las estadísticas globales del índice haciendo un full index scan.

Desde versión 12c, se pueden usar estadísticas incrementales para calcular estadísticas globales en una tabla particionada incluso cuando las estadísticas de una partición o subpartición son "stale" o bloqueadas.

En versiones anteriores a 12c, en cuanto una DML ocurre en una partición, el optimizador considera las estadísticas de esa partición como "stale". Tras esto DBMS_STATS tiene que generar estadísticas otra vez en la partición y actualizar las estadísticas globales. Además, si la DML ocurre en una partición cuyas estadísticas están bloqueadas, DBMS_STATS no recalculará estadísticas en la partición, pero para calcular estadísticas globales no usará estadísticas incrementales y tendrá que realizar un full table scan.

En versión 12c, se añade preferencia INCREMENTAL_STALENESS que controla cómo la base de datos determinará cuando las estadísticas de una partición o subpartición son puestas como stale.

Los valores que puede tomar esta preferencia son:

- USE_STALE_PERCENT
Una partición o subpartición no es considerada "stale" si los cambios DML son menos que la preferencia STALE_PERCENT a nivel de tabla. Por defecto el STALE_PERCENT es 10, lo que significa que si sentencias DML causan más de un 10% de cambios en filas, la tabla es considerada "stale".
- USE_LOCKED_STATS
Las estadísticas bloqueadas en particiones o subparticiones no serán consideradas "stale", independientemente de las sentencias DML que sufran.
- NULL (default)
Una partición o subpartición se considerará stale si sufre cualquier cambio DML. Es el mismo comportamiento que en versión 11g.

Con el valor por defecto, las estadísticas capturadas en modo incremental se garantizan que son las mismas que en modo noincremental. Cuando se usa otro valor, las estadísticas incrementales podrían ser menos precisas que si no se usan pero el coste en el proceso de recolección de estadísticas se reducirá por el uso de estadísticas incrementales.

Se puede especificar juntos USE_STALE_PERCENT y USE_LOCKED_STATS. Por ejemplo, se puede cambiar la preferencia "incremental_staleness" para una tabla de esta forma:

```
BEGIN
  DBMS_STATS.SET_TABLE_PREFS (
    ownname    => null
  ,   table_name => 't'
  ,   pname     => 'incremental_staleness'
  ,   pvalue    => 'use_stale_percent,use_locked_stats'
  );
END;
```

También a partir de versión 12c las estadísticas incrementales pueden ser usadas para evitar tener que recalcularse estadísticas globales tras el uso de partition exchange.

Cuando una tabla no particionada pasa con exchange a ser una partición de una tabla, Oracle automáticamente calcula las estadísticas globales usando las estadísticas de la tabla no particionada y las estadísticas existentes a nivel de

partición. Esto será posible si las estadísticas fueron capturadas en el la tabla no particionada con DBMS_STATS con la preferencia INCREMENTAL puesta a true, y el INCREMENTAL_LEVEL configurado a TABLE.

Ejemplos recolección estadísticas en tablas particionadas

Veamos un sencillo ejemplo para mostrar cómo actúa la recolección de estadísticas:

```
CREATE TABLE test_part_tab
(
  Col_Num    NUMBER(5)    NOT NULL
, Col_Chr    CHAR(5)      NOT NULL
, Col_Dat    DATE
)
PARTITION BY RANGE (Col_Num)
(
  PARTITION P_LESS02 VALUES LESS THAN (2)
, PARTITION P_LESS04 VALUES LESS THAN (4)
, PARTITION P_LESS06 VALUES LESS THAN (6)
) ;
```

Verificamos las estadísticas de la tabla creada, se verifica que no existen datos sobre la tabla.

```
alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
  table_name || NVL2(partition_name, ': ' || partition_name, '')
  Object_Name
  , Last_Analyzed
  , num_rows
  , GLOBAL_STATS
FROM
  user_tab_statistics
WHERE table_name = 'TEST_PART_TAB'
;
```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST_PART_TAB			NO
TEST_PART_TAB:P_LESS02			NO
TEST_PART_TAB:P_LESS04			NO
TEST_PART_TAB:P_LESS06			NO

El escenario inicial es el siguiente: Tenemos una nueva tabla sin registros y sin estadísticas, en primer lugar generemos estadísticas sobre la tabla.

```
EXEC DBMS_STATS.GATHER_TABLE_STATS (OwnName=>'OSS' ,
TabName=>'TEST_PART_TAB', Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
  table_name || NVL2(partition_name, ': ' || partition_name, '')
```

```

Object_Name
, Last_Analyzed
, num_rows
, GLOBAL_STATS
FROM
  user_tab_statistics
WHERE table_name = 'TEST_PART_TAB'
;

```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST_PART_TAB	15-11-2017 07:50:08	0	YES
TEST_PART_TAB:P_LESS04	15-11-2017 07:50:08	0	YES
TEST_PART_TAB:P_LESS02	15-11-2017 07:50:08	0	YES
TEST_PART_TAB:P_LESS06	15-11-2017 07:50:08	0	YES

Vemos que se han calculado estadísticas para todas las particiones y globales. Insertamos un registro y volvemos a lanzar estadísticas sobre la tabla y el resultado es el siguiente:

```

SQL> INSERT INTO test part tab VALUES (1,'A1', SYSDATE) ;
COMMIT ;
1 fila creada.

EXEC DBMS_STATS.GATHER_TABLE_STATS (OwnName=>'OSS', TabName=>'TEST_PART_TAB',
Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls date format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
  table_name || NVL2(partition_name,':'||partition_name,'')
  Object Name
  , Last Analyzed
  , num_rows
  , GLOBAL_STATS
FROM
  user_tab_statistics
WHERE table name = 'TEST PART TAB'
;

```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST PART TAB	15-11-2017 07:51:36	1	YES
TEST PART TAB:P LESS04	15-11-2017 07:51:36	0	YES
TEST PART TAB:P LESS02	15-11-2017 07:51:36	1	YES
TEST_PART_TAB:P_LESS06	15-11-2017 07:51:36	0	YES

Se han actualizado todas las estadísticas a nivel global y de partición aunque sólo una partición ha variado. En este ejemplo el volumen de los datos es muy pequeño y el tiempo invertido en actualizar las estadísticas es mínimo, deberemos pensar en el tiempo que puede necesitarse para una tabla con un número elevado de registros y particiones.

Utilizando las estadísticas incrementales pueden mantenerse las estadísticas globales de una tabla particionada sin realizar un escaneo completo de tabla. Con las tablas particionadas, es muy común cargar nuevos datos en una nueva partición. Tras la carga de los datos a medida que se añaden nuevas particiones, o se actualizan datos en las particiones existentes, las estadísticas de las tablas globales deben mantenerse actualizados.

Cambiamos la configuración para poder utilizar estadísticas incrementales.

```
SQL> SELECT
dbms_stats.get_prefs('INCREMENTAL', tabname=>'TEST PART TAB') "INCREMENTAL"
FROM dual ;

INCREMENTAL
-----
FALSE

EXEC DBMS_STATS.SET_TABLE_PREFS (OwnName=>'OSS', TabName=>'TEST_PART_TAB',
pname=>'INCREMENTAL', pvalue=>'TRUE');

SQL> SELECT
dbms_stats.get_prefs('INCREMENTAL', tabname=>'TEST PART TAB') "INCREMENTAL"
FROM dual ;

INCREMENTAL
-----
TRUE
```

Insertamos algún registro y calculamos estadísticas:

```
INSERT INTO test_part_tab VALUES (1,'A1', SYSDATE);
COMMIT ;

EXEC DBMS_STATS.GATHER TABLE STATS (OwnName=>'OSS', TabName=>'TEST PART TAB',
Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
    table name || NVL2(partition name,':'||partition name,'')
    Object Name
    , Last Analyzed
    , num_rows
    , GLOBAL_STATS
FROM
    user tab statistics
WHERE table name = 'TEST PART TAB'
;
```

OBJECT_NAME	LAST ANALYZED	NUM_ROWS	GLO
TEST PART TAB	15-11-2017 07:54:56	2	YES
TEST PART TAB:P_LESS04	15-11-2017 07:54:56	0	YES
TEST_PART_TAB:P_LESS02	15-11-2017 07:54:56	2	YES
TEST_PART_TAB:P_LESS06	15-11-2017 07:54:56	0	YES

Vemos que ha actualizado los datos de todas las particiones pero la siguiente vez que se calculen, sólo se actualizarán las que hayan sido modificadas así como las globales. Si insertamos un registro en una de las particiones y volvemos a calcularlas:

```
INSERT INTO test_part_tab VALUES (1,'A1', SYSDATE);
COMMIT ;

EXEC DBMS_STATS.GATHER TABLE STATS (OwnName=>'OSS', TabName=>'TEST PART TAB',
Granularity=>'AUTO', Cascade=>TRUE);
```

```
alter session set nls date format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
    table_name || NVL2(partition_name,':'||partition_name,'')
    Object_Name
    , Last Analyzed
    , num rows
    , GLOBAL_STATS
FROM
    user_tab_statistics
WHERE table name = 'TEST PART TAB'
;
```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST_PART_TAB	15-11-2017 07:56:05	3	YES
TEST PART TAB:P LESS04	15-11-2017 07:54:56	0	YES
TEST PART TAB:P LESS02	15-11-2017 07:56:05	3	YES
TEST PART TAB:P LESS06	15-11-2017 07:54:56	0	YES

Vemos que sólo han cambiado las estadísticas a nivel global y las de la partición que fue cambiada. Lo mismo ocurre si se hacen un delete/update de registros de particiones:

```
INSERT INTO test_part_tab VALUES (-1,'A', SYSDATE);
INSERT INTO test_part_tab VALUES (2,'A', SYSDATE);
INSERT INTO test_part_tab VALUES (3,'A', SYSDATE);
INSERT INTO test_part_tab VALUES (3,'A', SYSDATE);
update test part tab set Col Chr='dsd' where col num=2;
COMMIT ;

EXEC DBMS_STATS.GATHER_TABLE_STATS(OwnName=>'OSS', TabName=>'TEST_PART_TAB',
Granularity=>'AUTO', Cascade=>TRUE);

column object name format a30
SELECT
    table_name || NVL2(partition_name,':'||partition_name,'')
    Object_Name
    , Last Analyzed
    , num rows
    , GLOBAL_STATS
FROM
    user_tab_statistics
WHERE table_name = 'TEST_PART_TAB'
;
```

OBJECT NAME	LAST ANALYZED	NUM ROWS	GLO
TEST_PART_TAB	15-11-2017 07:58:01	7	YES
TEST_PART_TAB:P_LESS04	15-11-2017 07:58:00	3	YES
TEST PART TAB:P LESS02	15-11-2017 07:58:00	4	YES
TEST PART TAB:P LESS06	15-11-2017 07:54:56	0	YES

```
update test_part_tab set Col_Chr='dsd' where col_num=3;
commit;

EXEC DBMS_STATS.GATHER_TABLE_STATS(OwnName=>'OSS', TabName=>'TEST PART TAB',
Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';

column object name format a30
SELECT
    table name || NVL2(partition name,':'||partition name,'')
    Object_Name
    , Last Analyzed
    , num_rows
```

```

, GLOBAL STATS
FROM
  user_tab_statistics
WHERE table_name = 'TEST_PART_TAB'
;

OBJECT NAME                                LAST ANALYZED                            NUM ROWS GLO
-----
TEST_PART_TAB                              15-11-2017 07:59:18                      7 YES
TEST_PART_TAB:P_LESS04                     15-11-2017 07:59:18                      3 YES
TEST PART TAB:P LESS02                     15-11-2017 07:58:00                      4 YES
TEST_PART_TAB:P_LESS06                     15-11-2017 07:54:56                      0 YES

```

Sólo las partición modificada y las globales son actualizadas.

A continuación se muestran los casos cuando se realizan operaciones sobre la tabla particionada split, exchange, etc. .

Ejemplo con split partition

Cuando se hace un split partition, las estadísticas de las particiones que son divididas se anulan, y las estadísticas globales no cambian. Por tanto sería recomendado calcula estadísticas sobre la tabla. Si se calculan estadísticas y son incrementales sólo las globales y las particiones involucradas se calcularán, si no fueran incrementales recalcularía estadísticas en todas las particiones y requerirá un FTS completo.

En el ejemplo anterior, se simula un *split* de 2 particiones:

```

ALTER TABLE TEST_PART_TAB SPLIT PARTITION P_LESS02 AT (0) INTO (PARTITION
P_LESS00,PARTITION P_LESS02) ;

ALTER TABLE TEST PART TAB SPLIT PARTITION P_LESS04 AT (3) INTO (PARTITION
P_LESS03,PARTITION P_LESS04) ;

```

Sin calcular estadísticas vemos qué datos de estadísticas contiene ahora la tabla:

```

SELECT
table_name||NVL2(partition_name,':'||partition_name,') Object_Name
, Last Analyzed
, num rows
FROM
user_tab_statistics WHERE table_name = 'TEST_PART_TAB' ;

OBJECT NAME                                LAST ANALYZED                            NUM ROWS
-----
TEST_PART_TAB                              15-11-2017 07:59:18                      7
TEST_PART_TAB:P_LESS02
TEST_PART_TAB:P_LESS04
TEST PART TAB:P LESS00
TEST PART TAB:P LESS03
TEST PART TAB:P LESS06                     15-11-2017 07:54:56                      0

6 rows selected.

```

Mantiene estadísticas en las particiones que no han sido tocadas, también las globales.

Calculamos de nuevo estadísticas:

```
EXEC DBMS_STATS.GATHER TABLE STATS(OwnName=>'OSS', TabName=>'TEST PART TAB',
Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls date format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
    table_name || NVL2(partition_name,':'||partition_name,'')
    Object Name
    , Last Analyzed
    , num_rows
    , GLOBAL_STATS
FROM
    user_tab_statistics
WHERE table_name = 'TEST PART TAB'
;
```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST PART TAB	15-11-2017 08:01:19	7	YES
TEST PART TAB:P LESS04	15-11-2017 08:01:19	2	YES
TEST PART TAB:P LESS02	15-11-2017 08:01:19	3	YES
TEST PART TAB:P LESS00	15-11-2017 08:01:19	1	YES
TEST PART TAB:P LESS03	15-11-2017 08:01:19	1	YES
TEST PART TAB:P LESS06	15-11-2017 07:54:56	0	YES

Vemos que se actualizan sólo las involucradas en el Split y las globales.

Si modifico alguna fila en una partición, vemos que el cálculo de estadísticas sólo afectaría a esa partición y a las globales:

```
update test_part_tab set Col_Chr='dsd' where col_num=3;
commit;

EXEC DBMS_STATS.GATHER TABLE STATS(OwnName=>'OSS', TabName=>'TEST PART TAB',
Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';

column object_name format a30
SELECT
    table_name || NVL2(partition_name,':'||partition_name,'')
    Object Name
    , Last Analyzed
    , num_rows
    , GLOBAL_STATS
FROM
    user_tab_statistics
WHERE table_name = 'TEST PART TAB'
;
```

OBJECT NAME	LAST ANALYZED	NUM ROWS	GLO
TEST PART TAB	15-11-2017 08:02:12	7	YES
TEST PART TAB:P LESS04	15-11-2017 08:02:12	2	YES
TEST PART TAB:P LESS02	15-11-2017 08:01:19	3	YES
TEST PART TAB:P LESS00	15-11-2017 08:01:19	1	YES
TEST PART TAB:P LESS03	15-11-2017 08:01:19	1	YES
TEST PART TAB:P LESS06	15-11-2017 07:54:56	0	YES

6 rows selected.

Ejemplo con exchange partition

Cuando se hace un exchange partition, si la tabla auxiliar tiene estadísticas estas serán automáticamente incluidas como estadísticas locales para esta partición pero no se actualizan las estadísticas globales. Sería recomendado actualizar las estadísticas globales, si no se usa estadísticas incrementales se actualizan todas las particiones y se hará un FTS, si hay incrementales sólo las globales y la nueva partición sin necesidad de un FTS.

Siguiendo con el mismo ejemplo, vamos a utilizar la table TMP_EXCH como table auxiliar que contendrá los datos que serán “absorbidos” por la tabla particionada.

```
SQL> CREATE TABLE tmp_exch AS SELECT * FROM test_part_tab PARTITION (p_less06) ;
;
Tabla creada.
SQL> INSERT INTO tmp_exch VALUES (5,'E2', SYSDATE) ;
1 fila creada.
SQL> commit;
Confirmación terminada.
```

Después de pasar estadísticas sobre la nueva tabla:

```
EXEC DBMS_STATS.GATHER_TABLE_STATS (OwnName=>'OSS', TabName=>'TMP_EXCH');

OBJECT_NAME                                LAST_ANALYZED                                NUM_ROWS
-----
TMP_EXCH                                    06-07-2015 10:08:02                            1
```

Realizamos el “Exchange Partition” y confirmamos que las estadísticas globales y a nivel de partición se han actualizado.

```
SQL> ALTER TABLE
      TEST_PART_TAB
EXCHANGE PARTITION P_LESS06
WITH TABLE TMP_EXCH
INCLUDING INDEXES
WITH VALIDATION
UPDATE GLOBAL INDEXES;

Tabla modificada.

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object name format a30
SELECT
      table_name || NVL2(partition_name,':'||partition_name,'')
      Object_Name
      , Last_Analyzed
      , num_rows
```

```

, GLOBAL_STATS
FROM
  user_tab_statistics
WHERE table_name = 'TEST_PART_TAB'
;

```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST_PART_TAB	15-11-2017 08:02:12	7	YES
TEST_PART_TAB:P_LESS06	15-11-2017 08:03:49	1	YES
TEST_PART_TAB:P_LESS04	15-11-2017 08:02:12	2	YES
TEST_PART_TAB:P_LESS02	15-11-2017 08:01:19	3	YES
TEST_PART_TAB:P_LESS00	15-11-2017 08:01:19	1	YES
TEST_PART_TAB:P_LESS03	15-11-2017 08:01:19	1	YES

6 rows selected.

Vemos que se añaden los datos de estadísticas existentes en la tabla origen a los datos de estadísticas de la partición implicada. No se actualizan automáticamente las estadísticas globales.

Habría que calcularlas:

```

EXEC DBMS_STATS.GATHER_TABLE_STATS (OwnName=>'OSS', TabName=>'TEST_PART_TAB',
Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
  table name || NVL2(partition name, ': ' || partition name, '')
  Object Name
  , Last Analyzed
  , num_rows
  , GLOBAL_STATS
FROM
  user tab statistics
WHERE table name = 'TEST PART TAB'
;

```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST PART TAB	15-11-2017 08:06:41	8	YES
TEST_PART_TAB:P_LESS04	15-11-2017 08:02:12	2	YES
TEST_PART_TAB:P_LESS02	15-11-2017 08:01:19	3	YES
TEST_PART_TAB:P_LESS00	15-11-2017 08:01:19	1	YES
TEST_PART_TAB:P_LESS03	15-11-2017 08:01:19	1	YES
TEST_PART_TAB:P_LESS06	15-11-2017 08:06:41	1	YES

Vemos que ha actualizado las estadísticas globales y las de la partición implicada.

Ejemplo añadiendo particiones en una tabla particionada

Esto será lo habitual en entornos DW, se añadirán particiones y se insertarán nuevos datos. En este caso, será recomendado recalcular estadísticas una vez cargados los nuevos datos y si son incrementales sólo se actualizarán las estadísticas globales y las de la nueva partición evitando un FTS.

Siguiendo con el mismo ejemplo, se añade una nueva partición:

```
ALTER TABLE test_part_tab
  ADD PARTITION P LESS08 VALUES LESS THAN (8);

alter session set nls date format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
  table_name || NVL2(partition_name,':'||partition_name,'')
  Object Name
  , Last Analyzed
  , num_rows
  , GLOBAL_STATS
FROM
  user_tab_statistics
WHERE table_name = 'TEST PART TAB'
;
```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST PART TAB	15-11-2017 10:35:51	8	YES
TEST PART TAB:P LESS04	15-11-2017 10:35:51	2	YES
TEST PART TAB:P LESS02	15-11-2017 10:35:51	3	YES
TEST PART TAB:P LESS00	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS03	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS06	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS08			NO

7 rows selected.

Se insertan datos en la nueva partición y se calculan estadísticas.

```
INSERT INTO test_part_tab VALUES (7,'A1', SYSDATE) ;
COMMIT ;

EXEC DBMS_STATS.GATHER TABLE STATS(OwnName=>'OSS', TabName=>'TEST PART TAB',
Granularity=>'AUTO', Cascade=>TRUE);

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
  table name || NVL2(partition name,':'||partition name,'')
  Object Name
  , Last Analyzed
  , num_rows
  , GLOBAL_STATS
FROM
  user tab statistics
WHERE table_name = 'TEST_PART_TAB'
;
```

OBJECT NAME	LAST ANALYZED	NUM ROWS	GLO
TEST PART TAB	15-11-2017 10:38:35	9	YES
TEST PART TAB:P LESS04	15-11-2017 10:35:51	2	YES
TEST PART TAB:P LESS02	15-11-2017 10:35:51	3	YES
TEST PART TAB:P LESS00	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS03	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS06	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS08	15-11-2017 10:38:35	1	YES

7 rows selected.
7 rows selected.

Sólo las estadísticas de la nueva partición y las globales son actualizadas.

Ejemplo merging partitions

Cuando se hace un merge de particiones adyacentes, la nueva partición resultante requerirá el recalcu de estadísticas. Si está configurada la opción de INCREMENTAL, sólo la partición resultante será actualizada, así como las estadísticas globales.

Siguiendo con el mismo ejemplo,;

```
alter table test part tab MERGE PARTITIONS P LESS06, P LESS08;

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object name format a30
SELECT
  table name || NVL2(partition name,':'||partition name,'')
  Object Name
  , Last_Analyzed
  , num_rows
  , GLOBAL_STATS
FROM
  user tab statistics
WHERE table name = 'TEST PART TAB'
;
```

OBJECT NAME	LAST ANALYZED	NUM ROWS	GLO
TEST_PART_TAB	15-11-2017 10:38:35	9	YES
TEST_PART_TAB:P_LESS04	15-11-2017 10:35:51	2	YES
TEST_PART_TAB:P_LESS02	15-11-2017 10:35:51	3	YES
TEST_PART_TAB:P_LESS00	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS03	15-11-2017 10:35:51	1	YES
TEST PART TAB:SYS P811			NO

6 rows selected.

```
EXEC DBMS_STATS.GATHER TABLE STATS (OwnName=>'OSS', TabName=>'TEST PART TAB',
Granularity=>'AUTO', Cascade=>TRUE);
```

```
alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
  table name || NVL2(partition name,':'||partition name,'')
  Object Name
  , Last_Analyzed
  , num_rows
  , GLOBAL_STATS
FROM
  user tab statistics
WHERE table_name = 'TEST_PART_TAB'
;
```

OBJECT NAME	LAST ANALYZED	NUM ROWS	GLO
TEST PART TAB	15-11-2017 10:39:38	9	YES
TEST_PART_TAB:P_LESS04	15-11-2017 10:35:51	2	YES
TEST_PART_TAB:P_LESS02	15-11-2017 10:35:51	3	YES
TEST_PART_TAB:P_LESS00	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS03	15-11-2017 10:35:51	1	YES
TEST PART TAB:SYS P811	15-11-2017 10:39:38	2	YES

6 rows selected.

Ejemplo moving partitions

Cuando se hace un move de un partición por ejemplo para cambiarlo de tablespace o cambiarle algunas características de la tabla (compresión, logging, etc) las estadísticas no son modificadas. Si se actualizan las estadísticas y está configurada la opción de INCREMENTAL, sólo la partición movida será actualizada, así como las estadísticas globales.

Nota: De forma independiente al cálculo de estadísticas, tanto en esta operación como en algunas de las comentadas anteriormente, hay que tener en cuenta las tareas que requieren el mantenimiento de los posibles índices locales y globales, ya que por ejemplo con un MOVE PARTITION, a menos que se especifique UPDATE INDEXES, cuando la partición que se mueve tiene datos, el comando MOVE PARTITION marcará la partición del índice local correspondiente, y todos los índices globales como UNUSABLE.

Siguiendo con el mismo ejemplo:

```
alter table test_part_tab MOVE PARTITION SYS_P811 TABLESPACE SYSTEM NOLOGGING
COMPRESS;

alter session set nls date format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
    table_name || NVL2(partition_name,':'||partition_name,'')
    Object Name
    , Last Analyzed
    , num_rows
    , GLOBAL_STATS
FROM
    user tab statistics
WHERE table name = 'TEST PART TAB'
;
```

OBJECT_NAME	LAST_ANALYZED	NUM_ROWS	GLO
TEST PART TAB	15-11-2017 10:39:38	9	YES
TEST PART TAB:SYS P811	15-11-2017 10:39:38	2	YES
TEST PART TAB:P_LESS04	15-11-2017 10:35:51	2	YES
TEST PART TAB:P_LESS02	15-11-2017 10:35:51	3	YES
TEST PART TAB:P_LESS00	15-11-2017 10:35:51	1	YES
TEST PART TAB:P_LESS03	15-11-2017 10:35:51	1	YES

6 rows selected.

Ejemplo truncate partitions

Con el ALTER TABLE ... TRUNCATE PARTITION se borrarán todas las filas de la partición, de forma similar a si se borrara la partición, excepto que sólo borra los datos pero no es borrada la partición físicamente. A nivel de estadísticas actuarán como un DELETE, esto es, tras el truncate no se actualizarán las estadísticas y será

recomendado recalcularlas. Si está configurada la opción de INCREMENTAL, sólo la partición truncada será actualizada, así como las estadísticas globales.

Siguiendo con el mismo ejemplo:

```
alter table test part tab TRUNCATE PARTITION SYS P811;

alter session set nls_date_format='dd-mm-yyyy hh24:mi:ss';
column object_name format a30
SELECT
    table_name || NVL2(partition_name, ':' || partition_name, '')
    Object Name
    , Last_Analyzed
    , num_rows
    , GLOBAL_STATS
FROM
    user tab statistics
WHERE table name = 'TEST PART TAB'
;
```

OBJECT NAME	LAST ANALYZED	NUM ROWS	GLO
TEST PART TAB	15-11-2017 10:41:15	9	YES
TEST PART TAB:SYS P811	15-11-2017 10:41:15	2	YES
TEST_PART_TAB:P_LESS04	15-11-2017 10:35:51	2	YES
TEST_PART_TAB:P_LESS02	15-11-2017 10:35:51	3	YES
TEST PART TAB:P LESS00	15-11-2017 10:35:51	1	YES
TEST PART TAB:P LESS03	15-11-2017 10:35:51	1	YES

6 rows selected.

Otros usos de Particionamiento

SQL Loader y Objetos Particionados

La utilidad de SQL*Loader que viene con Oracle database server es muy común para cargar datos externos a una base de datos Oracle.

SQL*Loader soporta la carga de datos sobre objetos particionados. Se podrá cargar sobre una única partición de una tabla particionada o sobre todas las particiones, pudiendo utilizar tanto direct path como conventional path, así como en paralelo.

Los índices locales y globales son mantenidos automáticamente cuando se cargan datos con SQL*Loader, excepto cuando se insertan con la opción Direct Path y se especifica la opción de SKIP_INDEX_MAINTENANCE=true (por defecto es false). Si se especifica la opción, los índices son marcados como UNUSABLE y listados en el fichero de log de SQL*Loader.

Datapump y Particionamiento

Desde 10g, con Datapump se puede exportar o importar una o más particiones o subparticiones en una tabla usando el nombre de la partición o subpartición. Para esto, se puede especificar el nivel que queremos export/importar en la cláusula TABLES, especificando con una coma todas las particiones o subparticiones de una tabla a las que se quiere limitar la tarea:

```
TABLES=[schema_name.]table_name[:partition_name]
```

A la hora de importar una tabla particionada se importará como una tabla particionada, o a partir de 11g con la opción PARTITION_OPTIONS=MERGE indicarle que se importe como una tabla no particionada. Por defecto se importará como una tabla particionada.

También se podrá indicar PARTITION_OPTIONS=DEPARTITION, que hará que se creen las particiones o subparticiones como tablas individuales. El nombre por defecto de la nueva tabla será la concatenación del nombre de la tabla y el nombre de la partición o el de la tabla y la subpartición.

El uso de la opción de PARTITION_OPTIONS=MERGE es un forma simple de convertir una tabla particionada en no particionada.

Particionamiento y Tablespaces Transportables

Transportable tablespaces es la forma más rápida de mover grandes volúmenes de datos entre dos bases de datos Oracle, moviendo físicamente el datafile o datafiles donde estén guardados los datos. Por ejemplo puede ser muy útil para mover datos históricos a otras bases de datos de forma rápida.

Hasta versión 11g, la tabla tiene que estar contenida totalmente (todas las particiones) en el conjunto de transportable tablespaces. Esto provoca que si se quiere transportar sólo una partición habría que hacer EXCHANGE de la tabla a una independiente o create as select, y poder de esta forma hacer un transportable tablespace de una tabla convencional.

En la nota [How to Move or Transport Table Partitions Using Transportable Table Space \(TTS\) Option?](#) (Doc ID 731559.1) muestra un ejemplo de cómo realizar esta tarea.

Desde versión 11g, está soportado el transporte de un único tablespace donde está una partición concreta y copiarlo a otra base de datos a nivel de datafile, sin tener que copiar todos los datafiles de todos los tablespaces que involucrados en todas las particiones de una tabla. Para esto, en el paso de exportación del metadata con datapump, se especificará en TABLES sólo la partición que se quiere copiar y se añade la opción TRANSPORTABLE=ALWAYS. Sigue siendo requerido tener el tablespace en cuestión en modo READ-ONLY en el momento de la copia física de los datafiles.

Se puede ver un ejemplo simple del procedimiento de copiar una única partición entre bases de datos en 11g en la nota de MOS: [Tablespace Transport for a Single Partition](#) (Doc ID 1063299.1)

Particiones read-only

Desde versión 12cR2 se pueden marcar particiones y subparticiones read-only de esta forma de forma simple proteger los datos de estas particiones de posibles operaciones DML lanzadas por un usuario o por un trigger que cambien datos.

Por ejemplo:

```
CREATE TABLE orders_read_write_only (
  order_id NUMBER (12),
  order_date DATE CONSTRAINT order_date_nn NOT NULL,
  state VARCHAR2(2)
) READ WRITE
PARTITION BY RANGE (order_date)
SUBPARTITION BY LIST (state)
( PARTITION order_p1 VALUES LESS THAN (TO_DATE ('01-DEC-
2015', 'DD-MON-YYYY')) READ ONLY
( SUBPARTITION order_p1_northwest VALUES ('OR', 'WA'),
  SUBPARTITION order_p1_southwest VALUES ('AZ', 'UT', 'NM')
),
PARTITION order_p2 VALUES LESS THAN (TO_DATE ('01-MAR-
2016', 'DD-MON-YYYY'))
( SUBPARTITION order_p2_northwest VALUES ('OR', 'WA'),
  SUBPARTITION order_p2_southwest VALUES ('AZ', 'UT', 'NM')
) READ ONLY
),
PARTITION order_p3 VALUES LESS THAN (TO_DATE ('01-JUL-
2016', 'DD-MON-YYYY'))
(
SUBPARTITION order_p3_northwest VALUES ('OR', 'WA') READ
ONLY,
SUBPARTITION order_p3_southwest VALUES ('AZ', 'UT', 'NM')
)
```

);

En este ejemplo la tabla particionada que es definida como READ WRITE, tiene algunas particiones y subparticones READ ONLY y otras READ WRITE.

Se puede chequear el estado de cada partición con:

```
SQL> SELECT PARTITION_NAME, READ_ONLY FROM USER_TAB_PARTITIONS
WHERE TABLE_NAME = 'ORDERS_READ_WRITE_ONLY';
PARTITION_NAME          READ
-----
ORDER_P1                YES
ORDER_P2                NONE
ORDER_P3                NONE

SQL> SELECT PARTITION_NAME, SUBPARTITION_NAME, READ_ONLY FROM
USER_TAB_SUBPARTITIONS WHERE TABLE_NAME
='ORDERS_READ_WRITE_ONLY';
PARTITION_NAME          SUBPARTITION_NAME          REA
-----
ORDER_P1                ORDER_P1_NORTHWEST        YES
ORDER_P1                ORDER_P1_SOUTHWEST        YES
ORDER_P2                ORDER_P2_NORTHWEST        NO
ORDER_P2                ORDER_P2_SOUTHWEST        YES
ORDER_P3                ORDER_P3_NORTHWEST        YES
ORDER_P3                ORDER_P3_SOUTHWEST        NO
```

Igualmente se podría especificar que la tabla fuera READ ONLY y que ciertas particiones sean READ WRITE. O cambiarlas posteriormente con un ALTER TABLE MODIFY PARTITION.

Ejemplo:

Se crea una tabla particionada READ ONLY a nivel de tabla, las particiones heredan esta configuración:

```
CREATE TABLE test_part_tab
(
  Col_Num    NUMBER(5)    NOT NULL
  ,Col_Chr   CHAR(5)     NOT NULL
  ,Col_Dat   DATE
)
READ ONLY
PARTITION BY RANGE (Col_Num)
(
  PARTITION P_LESS20 VALUES LESS THAN (20)
  ,PARTITION P_LESS40 VALUES LESS THAN (40)
) ;

SET LINESIZE 120
COLUMN table_name FORMAT A20
COLUMN partition_name FORMAT A20
COLUMN subpartition_name FORMAT A20
COLUMN read_only FORMAT A9

SELECT table_name,
       partition_name,
       read_only
FROM   user_tab_partitions
WHERE  table_name = 'TEST_PART_TAB'
ORDER BY 1, 2;
```

TABLE_NAME	PARTITION_NAME	READ_ONLY
TEST_PART_TAB	P_LESS20	YES
TEST_PART_TAB	P_LESS40	YES

Si se intenta insertar en la partición generará error:

```
INSERT INTO test_part_tab VALUES (1,'A1', SYSDATE) ;
```

*

ERROR at line 1:

ORA-14466: Data in a read-only partition or subpartition cannot be modified.

Se modifica para que la partición sea read write y se comprueba que se puede insertar:

```
ALTER TABLE TEST_PART_TAB MODIFY PARTITION P_LESS20 READ WRITE;
```

```
SET LINESIZE 120
COLUMN table_name FORMAT A20
COLUMN partition_name FORMAT A20
COLUMN subpartition_name FORMAT A20
COLUMN read_only FORMAT A9
```

```
SELECT table_name,
       partition_name,
       read_only
FROM   user_tab_partitions
WHERE  table_name = 'TEST_PART_TAB'
ORDER BY 1, 2;
```

```
SQL> SQL> SQL> SQL> SQL> 2 3 4 5 6
TABLE_NAME PARTITION_NAME READ_ONLY
-----
TEST_PART_TAB P_LESS20 NO
TEST_PART_TAB P_LESS40 YES
```

```
SQL> INSERT INTO test_part_tab VALUES (1,'A1', SYSDATE) ;
```

1 row created.

```
SQL> COMMIT;
```