Práctica 5: Utilización de SQL: 99 en Oracle

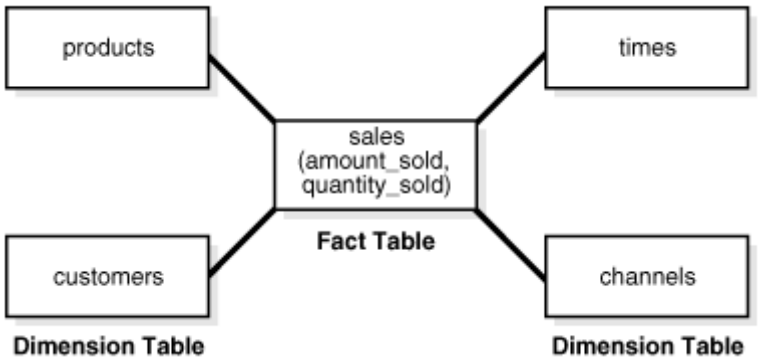
1. INTRODUCCIÓN Y OBJETIVOS

En esta práctica el alumno se familiarizará con algunas de las herramientas que proporciona Oracle para trabajar con almacenes de datos y en concreto con las extensiones de SQL para realizar consultas de agregados sobre varias dimensiones y sobre la creación de índices especiales.

Trabajaremos con un esquema en estrella, que es un esquema típico en los almacenes de datos.

ESQUEMA EN ESTRELLA

En esta práctica vamos a trabajar con un esquema de BDS en estrella compuesto por una tabla de hechos (SALES) y otras tablas que contienen las dimensiones (TIMES, PRODUCTS, CUSTOMERS, CHANNELS).



Estas tablas pertenecen al esquema ejemplo **SH** proporcionado por ORACLE. Estas tablas que ya están creadas y pobladas de datos están definidas de la siguiente manera:

SALES(PROD_ID,CUST_ID,TIME_ID,CHANNEL_ID,PROMO_ID,QUANTITY_SOLD,AMOUNT_SOLD)

PRODUCTS(**PROD_ID**,PROD_NAME,PROD_DESC,PROD_SUBCATEGORY,PROD_SUBCATEGORY_ID,PROD_CATEGORY,PROD_CATEGORY_ID, PROD_LIST_PRICE)

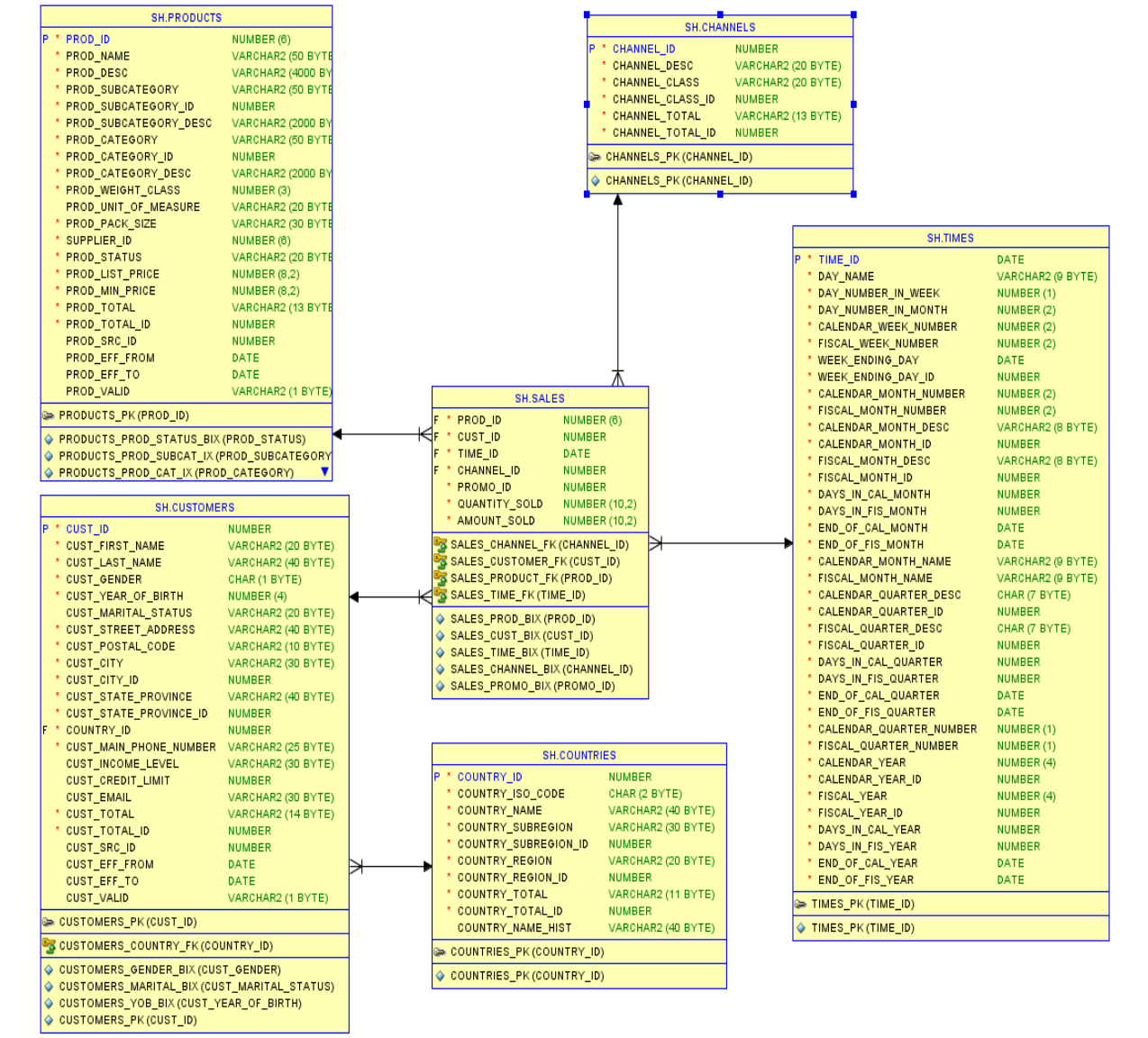
TIMES(**TIME_ID**,DAY_NAME,DAY_NUMBER_IN_WEEK,DAY_NUMBER_IN_MONTH,CALENDAR_WEEK_NUMBER,CALENDAR_MONTH_NAME,CALENDAR_YEAR,DAYS_IN_CAL_YEAR)

CUSTOMERS(**CUST_ID**,CUST_FIRST_NAME, CUST_LAST_NAME, CUST_GENDER, CUST_YEAR_OF_BIRTH, CUST_MARITAL_STATUS, CUST_STREET_ADDRESS,CUST_POSTAL_CODE, CUST_CITY,CUST_CITY_ID,CUST_STATE_PROVINCE,CUST_STATE_PROVINCE_ID, COUNTRY_ID, CUST_MAIN_PHONE_NUMBER,CUST_INCOME_LEVEL,CUST_CREDIT_LIMIT,CUST_EMAIL)

CHANNELS(**CHANNEL_ID**,CHANNEL_DESC, CHANNEL_CLASS, CHANNEL_CLASS_ID,CHANNEL_TOTAL,CHANNEL_TOTAL_ID)

También disponemos de una tabla extra que relaciona a cada cliente con su país de origen, para poder hacer consultas por zonas (si se añade esta tabla, el esquema es en realidad un esquema en copos).

COUNTRIES(**COUNTRY_ID**,COUTNRY_ISO_CODE,COUNTRY_NAME,COUNTRY_SUBREGION,COUNTRY_SUBREGION_ID,COUNTRY_REGION,COUNTRY_REGION_ID,COUNTRY_TOTAL,COUNTRY_TOTAL_ID)



Introducción a los operadores de agregación en Almacenes de Datos

La posibilidad de realizar agregaciones es una parte fundamental de los almacenes de datos. Oracle proporciona las siguientes funcionalidades:

- CUBE y RULLUP, extensiones de la cláusula GROUP BY.
- 3 funciones GROUPING
- La expresión GROUPING SETS
- Operaciones de pivotaje.

Para ilustrar el uso de estos operadores vamos a utilizar el esquema demostración de Oracle ***SH***, que contiene tablas con datos útiles para realizar operaciones de agregación.

Ejemplo 1: La empresa del ejemplo tiene ventas alrededor del mundo. Imaginemos que queremos extraer información, tipo una tabla de doble entrada mostrando las ventas por país y tipo de canal de distribución (country_id y channel_desc), filtrando solo para dos países (Francia y US) y sólo para 2 tipos de canal de distribución ('Internet', 'Direct Sales') :

Channel	Country		
	France	US	Total
Internet	9,597	124,224	133,821
Direct Sales	61,202	638,201	699,403
Total	70,799	762,425	833,224

La consulta que generaría esta información, vista de forma tabular sería la siguiente:

```
SELECT sh.channels.channel_desc, sh.countries.country_iso_code,

  TO_CHAR(SUM(sh.sales.amount_sold), '9,999,999,999') SALES$

FROM sh.sales, sh.customers, sh.times, sh.channels, SH.COUNTRIES

WHERE sh.sales.time_id=sh.times.time_id AND sh.sales.cust_id=customers.cust_id
AND

  sh.sales.channel_id= sh.channels.channel_id AND sh.channels.channel_desc IN

  ('Direct Sales', 'Internet') AND sh.times.calendar_month_desc='2000-09'

  AND sh.customers.country_id=sh.countries.country_id

  AND sh.countries.country_iso_code IN ('US','FR')

GROUP BY CUBE(sh.channels.channel_desc, sh.countries.country_iso_code);
```

Ejercicio 1: Piensa que consultas básicas **tipo group by** necesitarías ejecutar para obtener el resultado anterior.

Utilizando el operador CUBE, genera una tabla de doble entrada que muestre las ventas por tipo de canal (SH.CHANNELS.CHANNEL_DESC) y por categoría de producto (atributo PRODUCTS.PROD_CATEGORY).

En vista de estos datos, dí cuantos productos se han vendido en total por el canal 'Internet', y cuantos por venta directa.

Ejemplo 2: ROLLUP permite una instrucción SELECT para calcular múltiples niveles de subtotales a través de un grupo de dimensiones especificado. También calcula un total general.

ROLLUP es una extensión de la cláusula GROUP BY, por lo que su sintaxis es muy fácil de usar. La acción de ROLLUP es sencilla: crea subtotales que van desde el nivel más detallado hasta un gran total, siguiendo una lista de agrupación especificada en la cláusula ROLLUP.

```
SELECT sh.channels.channel_desc, sh.countries.country_iso_code,

  TO_CHAR(SUM(sh.sales.amount_sold), '9,999,999,999') VENTAS

FROM sh.sales, sh.customers, sh.times, sh.channels, SH.COUNTRIES

WHERE sh.sales.time_id=sh.times.time_id AND sh.sales.cust_id=customers.cust_id
AND SH.CUSTOMERS.COUNTRY_ID=SH.COUNTRIES.COUNTRY_ID AND

  sh.sales.channel_id= sh.channels.channel_id AND sh.channels.channel_desc IN

  ('Direct Sales', 'Internet') AND sh.times.calendar_month_desc='2000-09'

  AND  sh.channels.channel_desc IN ('Direct Sales', 'Internet')

GROUP BY ROLLUP(sh.channels.channel_desc, sh.countries.country_iso_code)
```

Ejercicio 2: Ejecuta la consulta del ejemplo anterior e interpreta los valores nulos que aparecen en algunas de las columnas.

Utiliza el operador ROLLUP para obtener las ventas de clientes agrupando dentro de la jerarquía de las direcciones de los clientes (country, country_region, country_subregion).

Mirando los resultados, di que regiones están formadas únicamente por una sub-región, por lo que hay varias filas en las coinciden los subtotales

EJEMPLO 3: GROUPING FUNCTION

El uso de ROLLUP y CUBE conlleva dos problemas.

En primer lugar, ¿Cómo se puede determinar mediante programación que filas del conjunto son subtotales, y ¿cómo encontrar el nivel exacto de agregación para un subtotal dado?

En segundo lugar, ¿qué sucede si los resultados de la consulta contienen tanto valores NULL almacenados como valores "NULL" creados por un ROLLUP o cubo? ¿Cómo se puede diferenciar entre los dos?

La función GROUPING maneja estos problemas: se le pasa una columna como argumento y GROUPING devuelve 1 cuando se encuentra un valor NULL creado por una operación ROLLUP o CUBE. Es decir, si el valor NULL corresponde a un subtotal GROUPING devuelve un 1. Cualquier otro tipo de valor, incluyendo un NULL almacenado, devuelve un 0.

La siguiente columna muestra un ejemplo de uso de la citada función, junto con el uso de la función **DECODE:**

```
SELECT DECODE(GROUPING(channel_desc), 1, 'Suma multi-canal', channel_desc) AS
Channel, DECODE (GROUPING (country_iso_code), 1, 'Suma multi-
pais',country_iso_code) AS Country, TO_CHAR(SUM(amount_sold), '9,999,999,999')
VENTAS FROM SH.SALES, SH.CUSTOMERS, SH.TIMES, SH.CHANNELS, SH.COUNTRIES WHERE
sh.sales.time_id=sh.times.time_id AND sh.sales.cust_id=customers.cust_id AND
sh.sales.channel_id= sh.channels.channel_id AND sh.channels.channel_desc IN
('Direct Sales', 'Internet') AND sh.times.calendar_month_desc='2000-09' AND
sh.channels.channel_desc IN ('Direct Sales', 'Internet') GROUP BY
CUBE(sh.channels.channel_desc, sh.countries.country_iso_code);
```

En este ejemplo se cambia los valores nulos de la consulta, devueltos por las funciones de agregación por texto asociado, de forma que la consulta resulta más sencilla de entender.

Ejercicio 3: Repite la consulta del ejercicio 2, pero mejorando su legibilidad, es decir utiliza la función GROUPING junto con DECODE. Construye un resultado que cambie los valores nulos que genera el GROUP BY ROLLUP por etiquetas apropiadas

EJEMPLO 4: GROUPING SETS

Se puede especificar selectivamente el conjunto de grupos que se desea crear utilizando un GROUPING SETS expresión dentro de una cláusula GROUP BY. Esto permite la especificación precisa a través de múltiples dimensiones sin calcular todo el cubo. Por ejemplo:

SELECT CHA.channel_desc, TI.calendar_month_desc, COU.country_iso_code, TO_CHAR(SUM(SA.amount_sold), '9,999,999,999') VENTAS

FROM SH.sales SA, SH.customers CU, SH.times TI, SH.channels CHA, SH.countries COU

WHERE

SA.time_id=TI.time_id AND SA.cust_id=CU.cust_id AND SA.channel_id= CHA.channel_id AND

CU.COUNTRY_ID = COU.COUNTRY_ID

AND

CHA.channel_desc IN ('Direct Sales', 'Internet') AND TI.calendar_month_desc IN ('2000-09', '2000-10') AND COU.country_iso_code IN ('GB', 'US')

GROUP BY GROUPING SETS((CHA.channel_desc, TI.calendar_month_desc, COU.country_iso_code),

```
(CHA.channel_desc, COU.country_iso_code), (TI.calendar_month_desc, COU.country_iso_code));
```

Ejercicio 4: Realiza una consulta que calcule el total de ventas agrupando por:

-mes de venta y país de venta,

-mes de venta y canal de distribución

-canal de distribución y categoría de producto.

INTRODUCCIÓN AL SQL PARA ANÁLISIS Y REALIZACIÓN DE INFORMES.

Oracle introduce varias funciones de cálculo analítico:

- Órdenes y percentiles
- Calculo de ventanas móviles.
- Análisis del Primero y último
- Estadísticas con modelos de regresión lineal
- OTROS…

Los conceptos esenciales que se utilizan en las funciones analíticas son:

- **Orden de procesamiento.** El procesamiento de consultas mediante funciones analíticas se realiza en tres etapas. En primer lugar, se realizan los JOIN, WHERE, GROUP BY y HAVING. En segundo lugar, el conjunto de resultados está a disposición de las funciones analíticas. En tercer lugar, si la consulta tiene una cláusula ORDER BY, se ordena el resultado y se devuelve con el orden especificado.
- **Particiones del conjunto de resultados.** Las funciones analíticas permiten a los usuarios dividir los resultados de la consulta en grupos de filas que se denominan particiones. Las particiones se crean después de los grupos definidos con las cláusulas GROUP BY, para que estén disponibles para todos los resultados agregados, tales como sumas y promedios. Las divisiones de particiones pueden basarse en las columnas o expresiones que se desee. Un conjunto de resultados de consulta puede dividirse en una sola partición que contiene todas las filas, unas cuantas particiones grandes, o muchas particiones pequeñas conteniendo pocas filas cada una.
- **Definición de ventanas.** Para cada fila de una partición, se puede definir una ventana deslizante. Esta ventana determina el rango de filas que se utilizan para realizar los cálculos para la fila actual. Los tamaños de ventana pueden basarse en un número físico de filas o un intervalo lógico tal como el tiempo. La ventana tiene una fila de partida y una fila de fin. Dependiendo de su definición, la ventana puede moverse en uno o ambos extremos. Por ejemplo, una ventana definida por una función suma acumulada tendría

su fila de partida fijado en la primera fila de su partición, y su fila de fin se deslizaría desde el punto de partida hasta el final a la última fila de la partición. Por otro lado, una ventana definida para una media móvil tendría tanto su punto inicial y final cambiantes, referidos siempre a la fila actual.

- **Fila actual.** Cada operación realizada con las funciones analíticas se basa en una fila concreta dentro de una partición (fila actual). La fila actual sirve de referencia para definir la ventana deslizante.

EJEMPLO 5: FUNCIONES RANK y DENSE_RANK

Estas funciones sirven para ordenar artículos en un grupo, por ejemplo para encontrar los 3 primeros productos vendidos en Valencia el año pasado. Estas dos funciones tienen la siguiente sintaxis:

RANK ( ) OVER ([query partition_clause] order_by_clause**)**

DENSE_RANK OVER ([query partition_clause] order_by_clause**)**

La diferencia entre RANK Y DENSE_RANK radica en que DENSE_RANK no deja huecos en el orden, mientras que RANK sí. Por ejemplo, si el resultado de una competición es ordenado mediante la función DENSE_RANK y hay tres personas que optan al 2º puesto (con la misma puntuación) la función diría que el orden es [1,2,2,2,3, 4..], mientras que RANK devolvería [1,2,2,2,5,6]

Puntos de interés sobre estas dos funciones:

- Si se especifica una partición, el orden se realiza para cada una de las particiones.
- Si no se especifica la partición, el orden se calcula sobre todos los datos.
- La cláusula ORDER BY especifica la medida sobre la que obtener el ranking.
- La cláusula NULLS FIRST | NULLS LAST indica la posición donde se colocarán los valores nulos.

La siguiente consulta sobre el esquema de prueba muestra el uso de las funciones descritas:

```
SELECT channel_desc,  SUM(amount_sold) VENTAS,

 RANK() OVER (ORDER BY SUM(amount_sold)) AS ORDEN_DEFECTO,

 RANK() OVER (ORDER BY SUM(amount_sold) DESC NULLS LAST) AS ORDEN_ESP

FROM  SH.sales,  SH.products,  SH.customers,  SH.times,  SH.channels,
SH.countries

WHERE            SH.sales.prod_id=SH.products.prod_id            AND
SH.sales.cust_id=SH.customers.cust_id

  AND   SH.customers.country_id   =   SH.countries.country_id   AND
SH.sales.time_id=times.time_id

  AND SH.sales.channel_id=channels.channel_id

  AND SH.times.calendar_month_desc IN ('2000-09', '2000-10')
```

```
  AND country_iso_code='US'

GROUP BY channel_desc;
```

Esta otra consulta muestra las diferencias entre las dos funciones explicadas, se ordena de acuerdo a las ventas realizadas, redondeando hasta las decenas de millar (-5), para obtener datos repetidos.

```
SELECT channel_desc, calendar_month_desc, TRUNC(SUM(amount_sold),-
5) VENTAS_R,

RANK() OVER (ORDER BY TRUNC(SUM(amount_sold),-5) DESC) AS
ORDEN,DENSE_RANK() OVER (ORDER BY TRUNC(SUM(amount_sold),-5) DESC)
AS ORDEN_DENSO

FROM SH.sales, SH.products, SH.customers, SH.times,
SH.channelsWHERE SH.sales.prod_id=SH.products.prod_id AND
SH.sales.cust_id=SH.customers.cust_id AND
SH.sales.time_id=SH.times.time_id AND
SH.sales.channel_id=SH.channels.channel_id AND
SH.times.calendar_month_desc IN ('2000-09', '2000-10') AND
SH.channels.channel_desc<>'Tele Sales'

group by channel_desc, calendar_month_desc;
```

Ejercicio 5: Sabiendo que las funciones RANK y DENSE_RANK pueden ordenar por particiones, utiliza la claúsula correspondiente para hacer una consulta que ordene las ventas realizadas para los distintos canales de distribución, en los meses comprendidos entre agosto y diciembre del año 2000.

Nota : La clausula **[**QUERY PARTITION_CLAUSE], opcional, tiene la forma:

PARTITION BY columna |expresión

En base al resultado di en que mes se realizaron más ventas para cada canal de distribución existente.

Ejemplo 6: Función CUME_DIST , PERCENT_RANK, NTILE

Otras funciones muy útiles son la función CUME_DIST y PERCENT_RANK y N_TILE

La primera de estas funciones calcula la posición de un valor especificado dentro de un conjunto de valores. Los valores devueltos están en el intervalo entre]0,1]. Se calcula como el número de valores del conjunto que están antes que x (incluyendo al propio valor), dividido por el cardinal del conjunto.

La segunda es similar pero se calcula utilizando la posición de la fila dentro de su partición menos 1 dividido por el número total de filas en la partición -1. Devuelve un valor entre [0,1].

La función NTILE permite realizar cálculos de cuartiles, tercios, etc. Esta función divide de forma ordenada el conjunto en cestas y asigna a cada una de estas cestas un valor numérico. Cada fila dentro de la cesta se le asigna ese mismo número.

Ejemplo de la última función en una consulta que ordena el grupo de tuplas devuelto por una consulta sobre las ventas realizadas en el año 2000, agrupadas por mes de calendario para los productos de categoría 'Electronics'. NTILE(4) significa que se divide en 4 grupos.

```
SELECT calendar_month_desc AS MES, SUM(amount_sold)VENTAS,

 NTILE(4) OVER (ORDER BY SUM(amount_sold)) AS CUARTIL

FROM sales, products, customers, times, channels

WHERE sales.prod_id=products.prod_id AND
sales.cust_id=customers.cust_id

  AND sales.time_id=times.time_id AND
sales.channel_id=channels.channel_id

  AND times.calendar_year=2000 AND prod_category= 'Electronics'

GROUP BY calendar_month_desc;
```

Ejercicio 6: Realiza una consulta que calcule la posición PERCENT_RANK de los resultados de una consulta que calcula las ventas totales de los clientes agrupados por país y por tipo de canal de venta. Realiza la misma operación utilizando la función CUME_DIST.

Ejemplo 7:

La Funciones de ventana puede calcular funciones de agregados acumuladas, y sobre ventanas deslizantes, utilizando una fila como referencia. Devuelven un valor para cada fila procesada (a diferencia de las funciones de agregado). Con estas funciones se pueden calcular versiones acumuladas de las funciones SUM, AVERAGE, COUNT, MAX, MIN. Sólo se pueden utilizar en las clausulas SELECT y ORDER BY. Se dispone de las funciones FIRST VALUE Y LAST_VALUE. Estas funciones permiten acceso a distintas filas de la misma tabla sin necesitar hacer reuniones reflexivas.

```
analytic_function([ arguments ])
   OVER (analytic_clause)

 where analytic_clause =
     [ query_partition_clause ]
     [ order_by_clause [ windowing_clause ] ]

and query_partition_clause =
    PARTITION BY
      { value_expr[, value_expr ]...
      }

 and windowing_clause =
     { ROWS | RANGE }
     { BETWEEN
       { UNBOUNDED PRECEDING
     | CURRENT ROW
     | value_expr { PRECEDING | FOLLOWING }
     }
     AND
     { UNBOUNDED FOLLOWING
     | CURRENT ROW
     | value_expr { PRECEDING | FOLLOWING }
     }
   | { UNBOUNDED PRECEDING
     | CURRENT ROW
     | value_expr PRECEDING
     }
   }
```

Por ejemplo, la siguiente consulta realiza una suma acumulada de ventas para cada cliente, y por mes. La primera función de agregado calcula el total de ventas de un cliente para cada mes (en el año 2000) y la siguiente función de agregada realiza la acumulación de esas ventanas dentro de los límites definidos por la cláusula windowing:

SELECT c.cust_id, t.calendar_quarter_desc, SUM(amount_sold) AS Q_SALES, **TO_CHAR(SUM(SUM(amount_sold))**

OVER (**PARTITION BY** c.cust_id ORDER BY c.cust_id, t.calendar_quarter_desc

ROWS UNBOUNDED

PRECEDING), '9,999,999,999.99') AS CUM_SALES

Clausula ventana: toma desde la primera fila dentro de la partición, hasta la fila actual

FROM sales s, times t, customers c

WHERE s.time_id=t.time_id AND s.cust_id=c.cust_id AND t.calendar_year=2000

GROUP BY c.cust_id, t.calendar_quarter_desc

ORDER BY c.cust_id, t.calendar_quarter_desc;

Ejercicio 7: Realiza una consulta que calcule las compras realizadas por el cliente 6510 para cada uno de los meses en el año 1999.

Una vez tengas esta consulta, utiliza una ventana móvil para calcular el promedio de ventas utilizando la consulta anterior como base.

INTRODUCCIÓN A LAS VISTAS MATERIALIZADAS EN ORACLE

La motivación del uso de vistas materializadas es mejorar el rendimiento, pero la sobrecarga asociada a la materialización de vistas puede convertirse en un problema importante de gestión del sistema. Algunas de las tareas relacionadas con la gestión de vistas son las siguientes.

- Identificar las vistas materializadas para crear inicialmente.
- La indexación de las vistas materializadas.
- Asegurar que todas las vistas materializadas y los índices asociados se actualizan correctamente cada vez que la base de datos se actualiza.
- Comprobar qué las vistas materializadas se han utilizado.
- Determinar la eficacia de cada vista materializada en la mejora del rendimiento.
- Medir el espacio que está siendo utilizado por las vistas materializadas.
- Determinar si se deben crear más vistas materializadas.
- Determinar si alguna de las ya creadas deben borrarse.

Tipos de vistas materializadas en Oracle:

En Oracle podemos hablar de tres tipos básicos de vistas materializadas:

- Vistas materializadas con agregados.
- Vistas materializadas con sólo operaciones JOIN
- Vistas materializadas anidadas.

Aquí voy a hacer una prueba de creación de varias vistas materializadas, y una muestra de cómo se agilizan las consultas con su creación, cuando se tienen los índices apropiados.

Ejemplo 8: Creación de vistas materializadas

Para poder crear vistas materializadas se utiliza la sentencia

CREATE MATERIALIZED VIEW

Por ejemplo la siguiente sentencia crea una vista materializada que calcula el total de dinero gastado de la tabla ventas, para cada producto:

```
CREATE MATERIALIZED VIEW product_sales_mv
REFRESH COMPLETE ON DEMAND
AS
SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales
FROM sales s, products p WHERE s.prod_id = p.prod_id
```

```
GROUP BY p.prod_name;
```

La consulta anterior actualiza la vista completamente cuando se solicita explícitamente. Otras opciones válidas para la creación de la vista son:

REFRESH COMPLETE ON COMMIT

REFRESH FAST ON COMMIT

REFRESH COMPLETE ON DEMAND

REFRESH FAST ON DEMAND

REFRESH FORCE ON DEMAND

Otra opción muy interesante de las vistas es el habilitar la reescritura de consultas: ENABLE QUERY REWRITE.

Está opción hace que en consultas que no utilizan directamente la consulta se busca si se agilizaría la respuesta utilizándola y si ese es el caso, se reescribe la consulta usando la vista materializada.

Para crear vistas materializadas de tablas fuentes situadas en otros esquemas se necesita disponer de los siguientes privilegios del sistema:

CREATE MATERIALIZED VIEW system privilege

CREATE ANY MATERIALIZAED VIEW

GLOBAL QUERY REWRITE system privilege

La tabla siguiente muestra las restricciones existentes sobre el uso de las funciones agregado cuando se utiliza la opción REFRESH FAST. En esta tabla x es la función de agregado que se desea utilizar, Y es en ese caso la función obligatoria a utilizar también y Z es opcional, pero recomendable.

X	Y	Z
COUNT(expr)	-	-
MIN(expr)	-	-
MAX(expr)	-	-
SUM(expr)	COUNT(expr)	-
SUM(col), col has NOT NULL constraint	-	-
AVG(expr)	COUNT(expr)	SUM(expr)
STDDEV(expr)	COUNT(expr) SUM(expr)	SUM(expr * expr)
VARIANCE(expr)	COUNT(expr) SUM(expr)	SUM(expr * expr)

Ejercicio 8: Realiza una consulta básica que calcule la suma de cantidad invertida (gastada) para cada producto y la cantidad de productos. Se debe mostrar el nombre del producto, el dinero invertido, y la cantidad de ese producto comprada (para todos los países, y clientes, y tiempo).

Anota el tiempo que tarde en resolver la consulta.

Ejercicio 9: Crea ahora una vista materializada que guarde para cada **country_id**, nombre de producto y tipo de canal de distribución la cantidad de dinero invertida y el número de productos vendidos.

Revisa la vista y utiliza dicha vista para resolver el ejercicio 8. Anota el tiempo que tarda en resolver la consulta.

Crea después un índice sobre la vista materializada que creas que mejore el rendimiento.

Ejercicio 10: Realiza la siguiente consulta y mira el tiempo de ejecución:

```
SELECT t.calendar_month_desc, SUM(s.amount_sold)
FROM sh.sales s, sh.times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

Ahora crea la siguiente vista con la opción de QUERY REWRITE:

```
CREATE MATERIALIZED VIEW ventas_por_mes
ENABLE QUERY REWRITE AS
SELECT t.calendar_month_desc, SUM(s.amount_sold) AS dollars
FROM sh.sales s, sh.times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

Ejecuta ahora la consulta primera de nuevo y revisa el tiempo de ejecución. ¿Ha tardado menos?? ¿¿Qué es lo que ha cambiado??