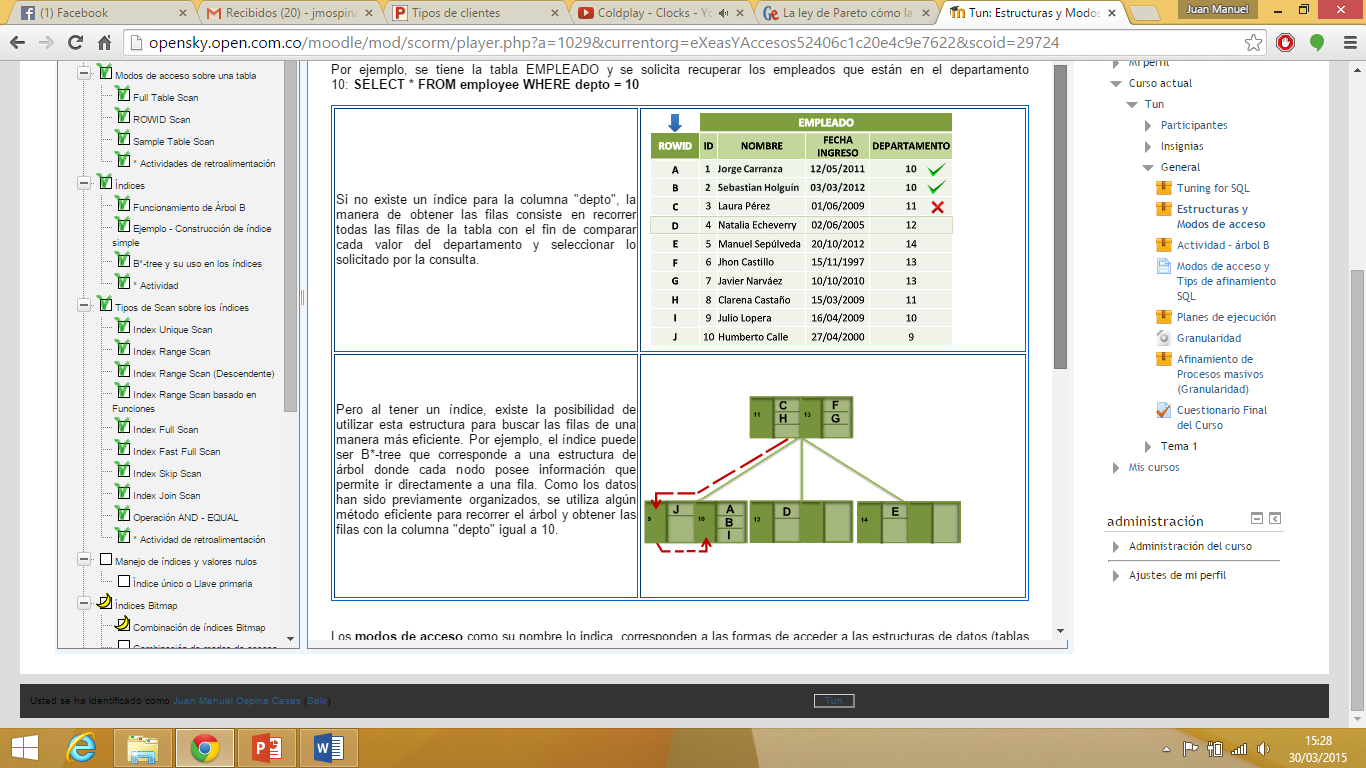
TUNING

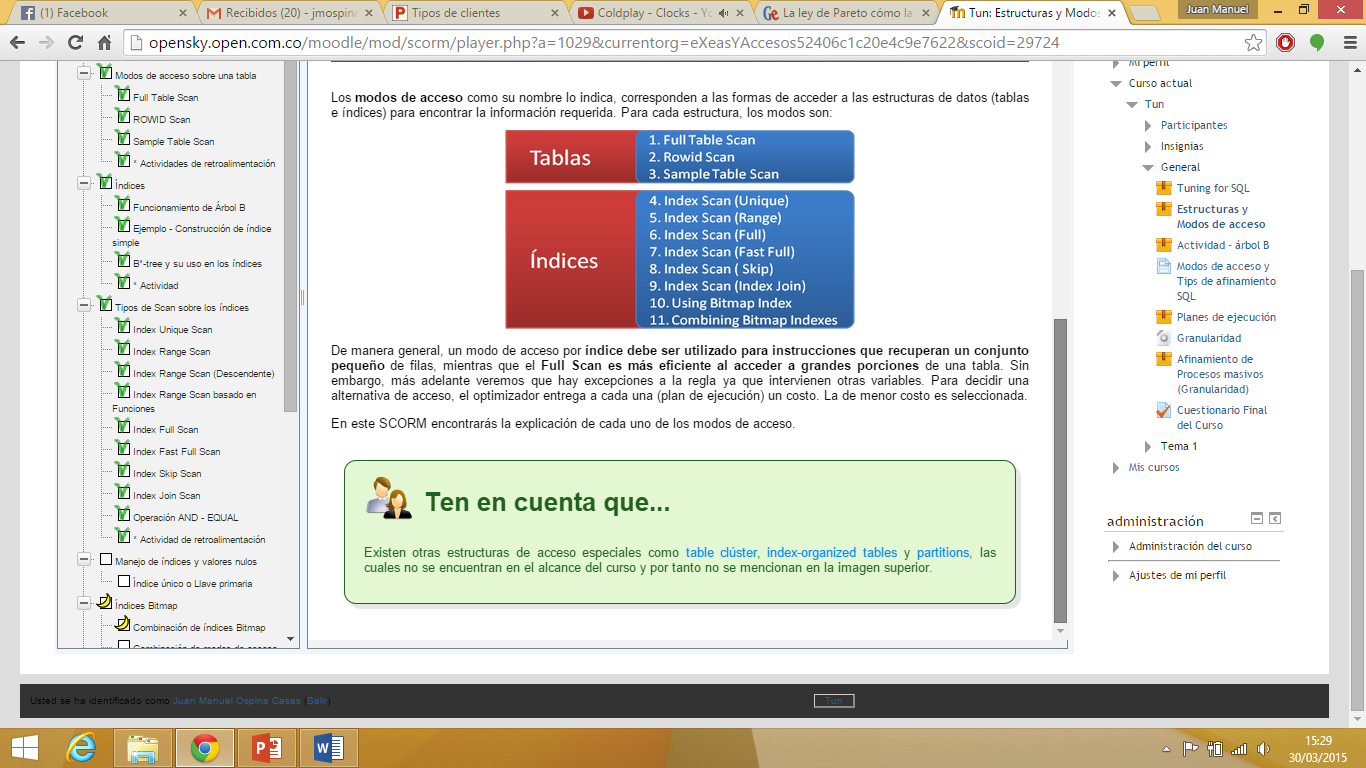
**Estructuras de datos y Modos de acceso**

Entre las estructuras de datos que conforman una Base de Datos Oracle se encuentran las **tablas** y los **índices**, ambas caracterizadas porque almacenan **lógica y físicamente** los datos. Una tabla es la estructura física donde se almacenan las filas de determinada información, mientras que un índice es una estructura física secundaria que permite el acceso directo a las filas de una tabla. Puede verse como un subconjunto de información de la tabla, estructurada de una manera ordenada que permite recuperar una o varias filas de manera más eficiente.

Por ejemplo, se tiene la tabla EMPLEADO y se solicita recuperar los empleados que están en el departamento 10: **SELECT \* FROM employee WHERE depto = 10**



Los **modos de acceso** como su nombre lo indica, corresponden a las formas de acceder a las estructuras de datos (tablas e índices) para encontrar la información requerida. Para cada estructura, los modos son :



De manera general, un modo de acceso por **índice debe ser utilizado para instrucciones que recuperan un conjunto pequeño**de filas, mientras que el **Full Scan es más eficiente al acceder a grandes porciones** de una tabla. Sin embargo, más adelante veremos que hay excepciones a la regla ya que intervienen otras variables. Para decidir una alternativa de acceso, el optimizador entrega a cada una (plan de ejecución) un costo. La de menor costo es seleccionada.

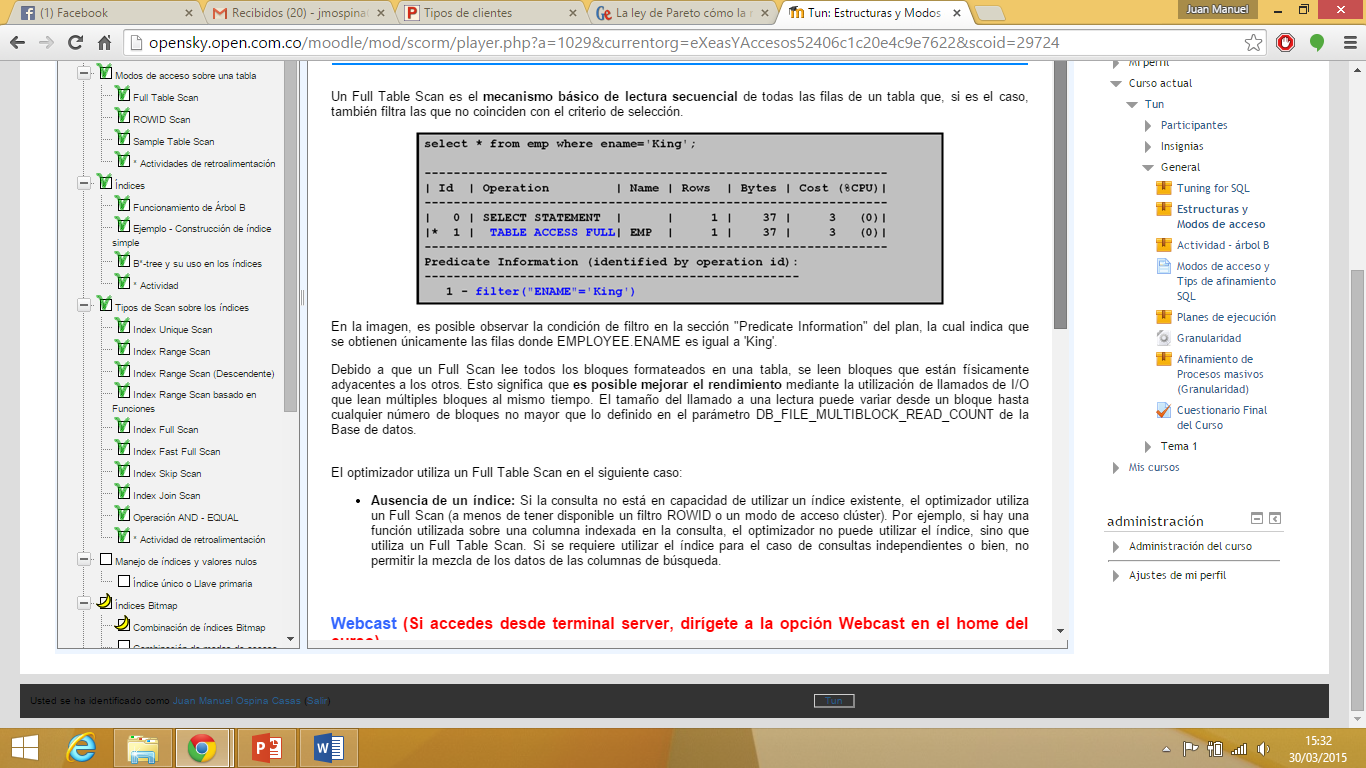
## Ten en cuenta que...

Existen otras estructuras de acceso especiales como [table clúster](http://docs.oracle.com/cd/E11882_01/server.112/e25789/tablecls.htm" \l "i25478" \t "_blank), [index-organized tables](http://docs.oracle.com/cd/E11882_01/server.112/e25789/indexiot.htm" \l "CBBJEBIH" \t "_blank)y [partitions](http://docs.oracle.com/cd/E11882_01/server.112/e25789/schemaob.htm" \l "CFAGCECI" \t "_blank), las cuales no se encuentran en el alcance del curso y por tanto no se mencionan en la imagen superior.

# Modos de acceso sobre una tabla

Haga clic en cada modo para ver la explicación:

Un Full Table Scan es el **mecanismo básico de lectura secuencial** de todas las filas de un tabla que, si es el caso, también filtra las que no coinciden con el criterio de selección.



En la imagen, es posible observar la condición de filtro en la sección "Predicate Information" del plan, la cual indica que se obtienen únicamente las filas donde EMPLOYEE.ENAME es igual a 'King'.

Debido a que un Full Scan lee todos los bloques formateados en una tabla, se leen bloques que están físicamente adyacentes a los otros. Esto significa que **es posible mejorar el rendimiento** mediante la utilización de llamados de I/O que lean múltiples bloques al mismo tiempo. El tamaño del llamado a una lectura puede variar desde un bloque hasta cualquier número de bloques no mayor que lo definido en el parámetro DB\_FILE\_MULTIBLOCK\_READ\_COUNT de la Base de datos.

El optimizador utiliza un Full Table Scan en el siguiente caso:

* **Ausencia de un índice:** Si la consulta no está en capacidad de utilizar un índice existente, el optimizador utiliza un Full Scan (a menos de tener disponible un filtro ROWID o un modo de acceso clúster). Por ejemplo, si hay una función utilizada sobre una columna indexada en la consulta, el optimizador no puede utilizar el índice, sino que utiliza un Full Table Scan. Si se requiere utilizar el índice para el caso de consultas independientes o bien, no permitir la mezcla de los datos de las columnas de búsqueda.

# ROWID Scan

El ROWID refiere a un identificador único de una fila, que indica su ubicación (Datafile y Datablock donde se encuentra). Utilizando este mecanismo se logra rapidéz ya que **especifica la posición exacta sin realizar ningún tipo de recorrido en una tabla**. Para acceder una tabla por ROWID, el sistema primero obtiene los Rowids de las filas seleccionadas por la cláusula WHERE o mediante un Índex Scan de una o más de las tablas indexadas. Posteriormente el sistema ubica cada fila seleccionada en la tabla, basado en sus rowids. Frecuentemente, el optimizador utiliza Rowids después de recuperar las filas con un índice:

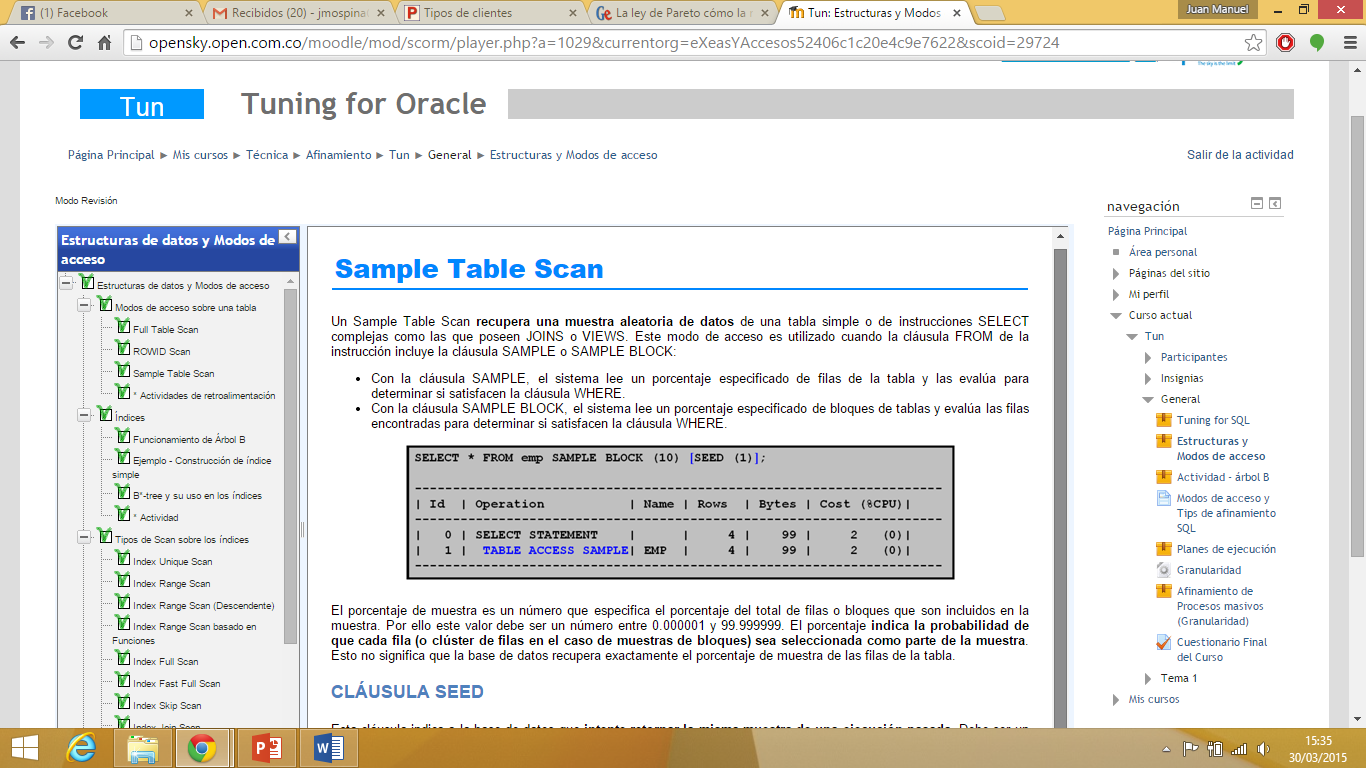
Si el índice contiene la (o las) columna requerida por la instrucción, no es necesario acceder a la tabla. Pero si el índice no incluye todas las columnas requeridas, es necesario acceder a la tabla utilizando el ROWID:

Tenga en cuenta que no es recomendado acceder a los datos por medio de su posición ya que las filas pueden moverse como producto de migraciones, encadenamientos, exportaciones e importaciones.

# Sample Table Scan

Un Sample Table Scan **recupera una muestra aleatoria de datos** de una tabla simple o de instrucciones SELECT complejas como las que poseen JOINS o VIEWS. Este modo de acceso es utilizado cuando la cláusula FROM de la instrucción incluye la cláusula SAMPLE o SAMPLE BLOCK:

* Con la cláusula SAMPLE, el sistema lee un porcentaje especificado de filas de la tabla y las evalúa para determinar si satisfacen la cláusula WHERE.
* Con la cláusula SAMPLE BLOCK, el sistema lee un porcentaje especificado de bloques de tablas y evalúa las filas encontradas para determinar si satisfacen la cláusula WHERE.



El porcentaje de muestra es un número que especifica el porcentaje del total de filas o bloques que son incluidos en la muestra. Por ello este valor debe ser un número entre 0.000001 y 99.999999. El porcentaje **indica la probabilidad de que cada fila (o clúster de filas en el caso de muestras de bloques) sea seleccionada como parte de la muestra**. Esto no significa que la base de datos recupera exactamente el porcentaje de muestra de las filas de la tabla.

## **CLÁUSULA SEED**

Esta cláusula indica a la base de datos que **intente retornar la misma muestra de una ejecución pasada**. Debe ser un entero entre 0 y 4294969295. Si se omite esta cláusula, la muestra resultante cambia de una ejecución a otra. Con este modo de acceso, más bloques necesitan ser accedidos, pero el resultado es usualmente más exacto. Es menos costoso pero puede ser inexacto principalmente con muestras pequeñas.

## Ten en cuenta que...

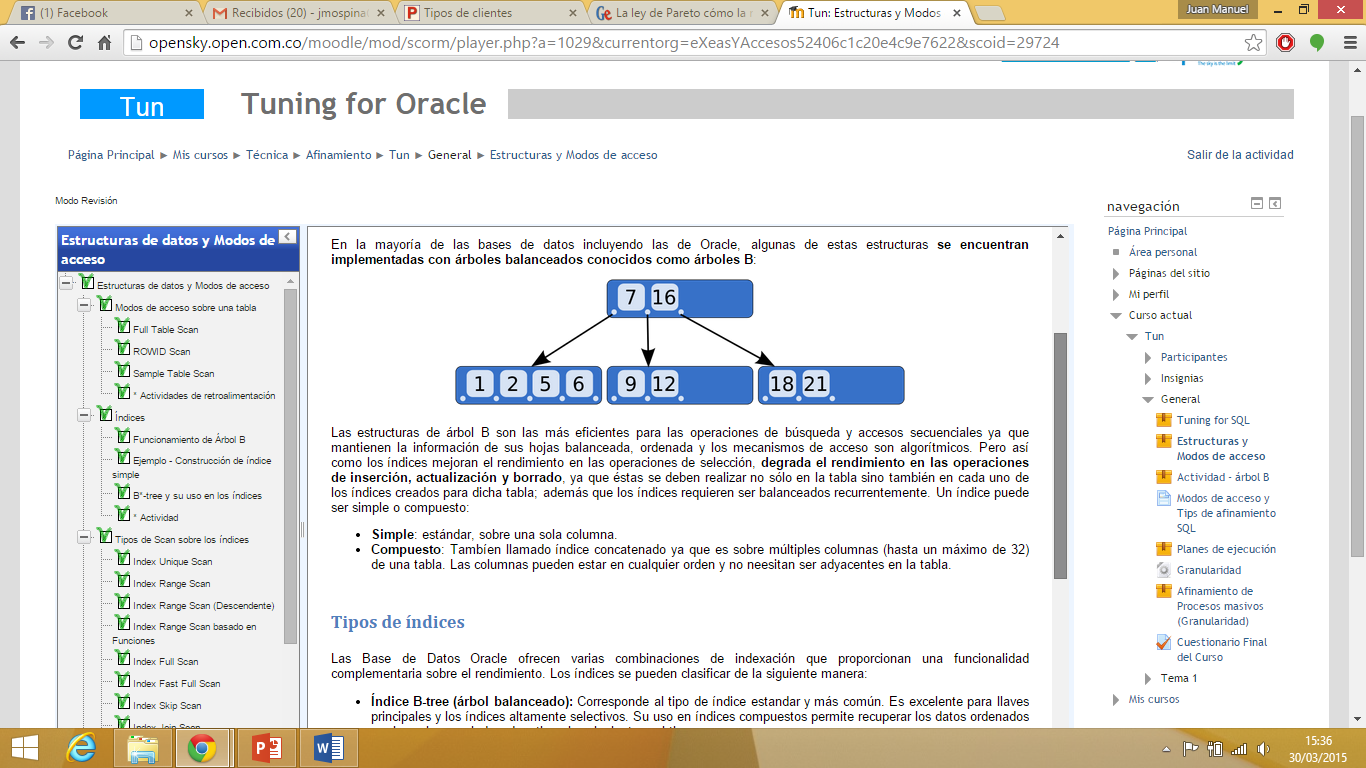
Un bloque de muestra es posible sólo durante un Full Table Scan o Index Fast Full Scan. Si existe un modo de acceso más eficiente, Oracle no lleva a cabo el block sampling. Pero si se quiere garantizar la ejecución de block sampling para una tabla o índice en particular, es posible utilizar las palaras reservadas FULL o INDEX\_FFS.

# Índices

Un índice es una estructura opcional asociada con una tabla, que en algunos casos acelera el acceso a los datos. Creando un índice sobre una o más columnas de una tabla, se mejora en muchas ocasiones la obtención de filas distribuidas aleatoriamente en una tabla. Es decir que los índices son uno de los elementos que ayudan a la eficiencia de operaciones I/O.

Debido a que un índice es una estructura de datos lógica y físicamente independiente de los datos de una tabla, requiere de espacio de almacenamiento. Funciona exactamente igual al índice de un libro, el cual ayuda a ubicar rápidamente la información. En una Base de Datos de Oracle, **el índice provee un modo de acceso rápido a los datos de una tabla.**

En la mayoría de las bases de datos incluyendo las de Oracle, algunas de estas estructuras **se encuentran implementadas con árboles balanceados conocidos como árboles B**:



Las estructuras de árbol B son las más eficientes para las operaciones de búsqueda y accesos secuenciales ya que mantienen la información de sus hojas balanceada, ordenada y los mecanismos de acceso son algorítmicos. Pero así como los índices mejoran el rendimiento en las operaciones de selección, **degrada el rendimiento en las operaciones de inserción, actualización y borrado**, ya que éstas se deben realizar no sólo en la tabla sino también en cada uno de los índices creados para dicha tabla; además que los índices requieren ser balanceados recurrentemente. Un índice puede ser simple o compuesto:

* **Simple**: estándar, sobre una sola columna.
* **Compuesto**: Tambíen llamado índice concatenado ya que es sobre múltiples columnas (hasta un máximo de 32) de una tabla. Las columnas pueden estar en cualquier orden y no neesitan ser adyacentes en la tabla.

## Tipos de índices

Las Base de Datos Oracle ofrecen varias combinaciones de indexación que proporcionan una funcionalidad complementaria sobre el rendimiento. Los índices se pueden clasificar de la siguiente manera:

* **Índice B-tree (árbol balanceado):** Corresponde al tipo de índice estandar y más común. Es excelente para llaves principales y los índices altamente selectivos. Su uso en índices compuestos permite recuperar los datos ordenados por las columnas indexadas. tiene los siguientes subtipos:

|  |
| --- |
| **Index-Organized Tables:** Los datos son el mismo índice. [(Clic aquí para detalle en Oracle Documentation)](http://docs.oracle.com/cd/E11882_01/server.112/e10713/indexiot.htm#CBBJEBIH) |
| **Reverse key Index:** Se trata de un índice B\*-tree en el que los bytes en el valor llave son invertidos. Puede ser utilizado para obtener una distribución más uniforme de las entradas de índice a lo largo de un índice que es llenado con valores crecientes. Por ejemplo, si se utiliza una secuencia para generar una primary key, la secuencia genera valores como 987500, 987501, 987502, etc. Con un índice Reverse key, la base de datos indexa 005789, 105789, 205789, etc. Debido a que los reverse keys posiblemente apuntan a diferentes posiciones no contiguas, puede reducir la contención de bloques particulares. Sin embargo, sólo los predicados de igualdad en un SELECT son beneficiados con estos índices. |
| **Índice ascendente/descendiente:**Permite ordenar la información de menor a mayor o de mayor a menor. |
| **B-tree cluster index:** Como su nombre lo indica, es utilizado para indexar una llave de un clúster de tablas. En lugar de apuntar a un fila, la llave apunta al bloque que contiene filas relacionadas con la cluster key buscada. |
| **Índice key compression:**Cada nodo del árbol está dividido en dos: Un prefijo y un sufijo. El prefijo está construido con las columnas principales del índice concatenado y tiene muchos valores repetidos. El sufijo está construido con las columnas finales del índex key y es el único componente de la entrada por índice con el prefijo. Este índice no es una compresión similar a la archivos como el .ZIP, sino que es una compresión opcional que elimina redundancias en los índices concatenados (multicolumna). |

# Funcionamiento de Árbol B

Si recuerdas qué es un árbol B o árbol balanceado, y su administración (inserción, eliminación), omite la información que viene a continuación. De lo contrario te sugerimos leerla, ya que te ayudará para la comprensión y resolución de  actividades posteriores.

A partir del artículo que encuentras a continuación, te sugerimos realizar su lectura y análisis buscando responder las siguientes preguntas.

1. ¿Qué caracteriza un árbol B?
2. ¿Cómo se busca un valor en el árbol?
3. ¿Cómo se realiza la inserción de un nuevo nodo?
4. ¿Cómo se elimina un nodo?

Árbol-B

De Wikipedia, la enciclopedia libre

En las [ciencias de la computación](http://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n), los **árboles-B** o **B-árboles** son [estructuras de datos de árbol](http://es.wikipedia.org/wiki/%C3%81rbol_(programaci%C3%B3n)) que se encuentran comúnmente en las implementaciones de [bases de datos](http://es.wikipedia.org/wiki/Bases_de_datos) y[sistemas de archivos](http://es.wikipedia.org/wiki/Sistemas_de_archivos). Son [árboles balanceados de búsqueda](http://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda)en los cuales cada nodo puede poseer más de dos hijos.[[1]](http://opensky.open.com.co/moodle/pluginfile.php/11855/mod_scorm/content/3/funcionamiento_de_rbol_b.html#cite_note-1)Los árboles B mantienen los datos ordenados y las inserciones y eliminaciones se realizan en tiempo logarítmico amortizado.

|  |
| --- |
| Índice  * 1 Definición * 2 Definición técnica * 3 Altura: El mejor y el peor caso * 4 Estructura de los nodos * 5 Algoritmos   + 5.1 Búsqueda   + 5.2 Inserción   + 5.3 Eliminación     - 5.3.1 Eliminación en un nodo hoja     - 5.3.2 Eliminación en un nodo interno   + 5.4 Rebalanceo después de la eliminación   + 5.5 Construcción Inicial * 6 Notas   + 6.1 Multi-modo:combinar y dividir   + 6.2 Relación entre U y L   + 6.3 Acceso concurrente |

## Definición

La idea tras los árboles-B es que los nodos internos deben tener un número variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se juntan o se parten. Dado que se permite un rango variable de nodos hijo, los árboles-B no necesitan rebalancearse tan frecuentemente como los árboles binarios de búsqueda auto-balanceables, pero por otro lado pueden desperdiciar memoria, porque los nodos no permanecen totalmente ocupados. Los límites una superior e inferior en el número de nodos hijo son definidos para cada implementación en particular. Por ejemplo, en un **árbol-B 2-3** (A menudo simplemente llamado árbol 2-3 ), cada nodo sólo puede tener 2 ó 3 nodos hijo.

Un árbol-B se mantiene balanceado porque requiere que todos los nodos hoja se encuentren a la misma altura.

Los árboles B tienen ventajas sustanciales sobre otras implementaciones cuando el tiempo de acceso a los nodos excede al tiempo de acceso entre nodos. Este caso se da usualmente cuando los nodos se encuentran en dispositivos de almacenamiento secundario como los discos rígidos. Al maximizar el número de nodos hijo de cada nodo interno, la altura del árbol decrece, las operaciones para balancearlo se reducen, y aumenta la eficiencia. Usualmente este valor se coloca de forma tal que cada nodo ocupe un bloque de disco, o un tamaño análogo en el dispositivo. Mientras que los árboles B 2-3 pueden ser útiles en la memoria principal, y además más fáciles de explicar, si el tamaño de los nodos se ajustan para caber en un bloque de disco, el resultado puede ser un árbol B 129-513.

Los creadores del árbol B, Rudolf Bayer y Ed McCreight, no han explicado el significado de la letra B de su nombre. Se cree que la B es de balanceado, dado que todos los nodos hoja se mantienen al mismo nivel en el árbol. La B también puede referirse a Bayer, o a Boeing, porque sus creadores trabajaban en los Boeing Scientific Research Labs por ese entonces.

## Definición técnica

B-árbol es un árbol de búsqueda que puede estar vacío o aquel cuyos nodos pueden tener varios hijos, existiendo una relación de orden entre ellos, tal como muestra el dibujo.

Un árbol-B de orden M (el máximo número de hijos que puede tener cada nodo) es un árbol que satisface las siguientes propiedades:

1. Cada nodo tiene como máximo M hijos.
2. Cada nodo (excepto raíz y hojas) tiene como mínimo (M+1)/2 claves.
3. La raíz tiene al menos 2 hijos si no es un nodo hoja.
4. Todos los nodos hoja aparecen al mismo nivel.
5. Un nodo no hoja con k hijos contiene k-1 elementos almacenados.
6. Los hijos que cuelgan de la raíz (r1, ···, rm) tienen que cumplir ciertas condiciones:
   1. El primero tiene valor menor que r1.
   2. El segundo tiene valor mayor que r1 y menor que r2, etc.
   3. El último hijo tiene valor mayor que rm.

## Altura: El mejor y el peor caso

En el mejor de los casos,la altura de un árbol-B es:

En el peor de los casos,la altura de un árbol-B es:

Donde M es el número máximo de hijos que puede tener un nodo.

## Estructura de los nodos

Cada elemento de un nodo interno actúa como un valor separador, que lo divide en subárboles. Por ejemplo, si un nodo interno tiene tres nodos hijo, debe tener dos valores separadores o elementos a1 y a2. Todos los valores del subárbol izquierdo deben ser menores a a1, todos los valores del subárbol del centro deben estar entre a1 y a2, y todos los valores del subárbol derecho deben ser mayores a a2.

Los nodos internos de un árbol B, es decir los nodos que no son hoja, usualmente se representan como un conjunto ordenado de elementos y punteros a los hijos. Cada nodo interno contiene un máximo de U hijos y, con excepción del nodo raíz, un mínimo de L hijos. Para todos los nodos internos exceptuando la raíz, el número de elementos es uno menos que el número de punteros a nodos. El número de elementos se encuentra entre L-1 y U-1. El número U debe ser 2L o 2L-1, es decir, cada nodo interno está por lo menos a medio llenar. Esta relación entre U y L implica que dos nodos que están a medio llenar pueden juntarse para formar un nodo legal, y un nodo lleno puede dividirse en dos nodos legales (si es que hay lugar para subir un elemento al nodo padre). Estas propiedades hacen posible que el árbol B se ajuste para preservar sus propiedades ante la inserción y eliminación de elementos.

Los nodos hoja tienen la misma restricción sobre el número de elementos, pero no tienen hijos, y por tanto carecen de punteros.

El nodo raíz tiene límite superior de número de hijos, pero no tiene límite inferior. Por ejemplo, si hubiera menos de L-1elementos en todo el árbol, la raíz sería el único nodo del árbol, y no tendría hijos.

Un árbol B de altura n+1 puede contener U veces por elementos más que un árbol B de profundidad n, pero el costo en la búsqueda, inserción y eliminación crece con la altura del árbol. Como todo árbol balanceado, el crecimiento del costo es más lento que el del número de elementos.

Algunos árboles balanceados guardan valores sólo en los nodos hoja, y por lo tanto sus nodos internos y nodos hoja son de diferente tipo. Los árboles B guardan valores en cada nodo, y pueden utilizar la misma estructura para todos los nodos. Sin embargo, como los nodos hoja no tienen hijos, una estructura especial para éstos mejora el funcionamiento.

## Algoritmos

**Búsqueda**

La búsqueda es similar a la de los árboles binarios. Se empieza en la raíz, y se recorre el árbol hacia abajo, escogiendo el sub-nodo de acuerdo a la posición relativa del valor buscado respecto a los valores de cada nodo. Típicamente se utiliza la búsqueda binaria para determinar esta posición relativa.

Procedimiento

1. Situarse en el nodo raíz.
2. (\*) Comprobar si contiene la clave a buscar.
   1. Encontrada fin de procedimiento.
   2. No encontrada:
      1. Si es hoja no existe la clave.
      2. En otro caso el nodo actual es el hijo que corresponde:
         1. La clave a buscar k < k1: hijo izquierdo.
         2. La clave a buscar k > ki y k < ki+1 hijo iésimo.
         3. Volver a paso 2(\*).

## Inserción

Todas las inserciones se hacen en los nodos hoja.

1. Realizando una búsqueda en el árbol, se halla el nodo hoja en el cual debería ubicarse el nuevo elemento.
2. Si el nodo hoja tiene menos elementos que el máximo número de elementos legales, entonces hay lugar para uno más. Inserte el nuevo elemento en el nodo, respetando el orden de los elementos.
3. De otra forma, el nodo debe ser dividido en dos nodos. La división se realiza de la siguiente manera:
   1. Se escoge el valor medio entre los elementos del nodo y el nuevo elemento.
   2. Los valores menores que el valor medio se colocan en el nuevo nodo izquierdo, y los valores mayores que el valor medio se colocan en el nuevo nodo derecho; el valor medio actúa como valor separador.
   3. El valor separador se debe colocar en el nodo padre, lo que puede provocar que el padre sea dividido en dos, y así sucesivamente.

Si las divisiones de nodos suben hasta la raíz, se crea una nueva raíz con un único elemento como valor separador, y dos hijos. Es por esto por lo que la cota inferior del tamaño de los nodos no se aplica a la raíz. El máximo número de elementos por nodo es U-1. Así que debe ser posible dividir el número máximo de elementos U-1 en dos nodos legales. Si este número fuera impar, entonces U=2L, y cada uno de los nuevos nodos tendrían (U-2)/2 = L-1 elementos, y por lo tanto serían nodos legales. Si U-1 fuera par, U=2L-1, así que habría 2L-2 elementos en el nodo. La mitad de este número es L-1, que es el número mínimo de elementos permitidos por nodo.

Un algoritmo mejorado admite una sola pasada por el árbol desde la raíz,hasta el nodo donde la inserción tenga lugar,dividiendo todos los nodos que estén llenos encontrados a su paso.Esto evita el costo de volver a cargar en memoria los nodos padres (lo que puede llegar a ser caro si los nodos se encuentran en una memoria secundaria). Sin embargo,para usar este algoritmo mejorado, debemos ser capaces de enviar un elemento al nodo padre y dividir el resto U-2 elementos en 2 nodos legales,sin añadir un nuevo elemento.Esto requiere que U=2L en lugar de U=L-1,lo que explica por qué algunos libros de texto imponen este requisito en la definición de árboles-B.

## Eliminación

Hay dos casos especiales a tener en cuenta cuando se elimina un elemento:  
  
1.El elemento en un nodo interno es un separador para sus nodos secundarios  
2.Deleting un elemento puede poner su nodo bajo el número mínimo de elementos y niños. Los procedimientos paraestos casos están en orden a continuación.

## Eliminación en un nodo hoja

Busque el valor a eliminar, si el valor se encuentra en un nodo hoja, se elimina directamente la clave, posiblemente dejándolo con muy pocos elementos; por lo que se requerirán cambios adicionales en el árbol.

## Eliminación en un nodo interno

(recursivamente) del nuevo nodillo.

En el segundo caso, uno de los dos nodos hijos tienen un número de elementos mayor que el mínimo. Entoncesizquierdo o el menor elemento dr el nuevo separador.Como se ha eliminado un elemento de un nodo hoja, se tar este caso de manera equivalente.

## Rebalanceo después de la eliminación

Si al eliminar un elemento de un nodo hoja el nodo se ha quedado con menos elementos que el mínimo permitido, algunos elementos se deben redistribuir. En algunos casos el cambio lleva la deficiencia al nodo padre, y la redistribución se debe aplicar iterativamente hacia arriba del árbol, quizá incluso hasta a la raíz. Dado que la cota mínima en el número de elementos no se aplica a la raíz, el problema desaparece cuando llega a ésta.

## Construcción Inicial

En aplicaciones, es frecuentemente útil construir un árbol-B para representar un gran número de datos existentes y después actualizarlo de forma creciente usando operaciones estándar de los árboles-B. En este caso, el modo más eficiente para construir el árbol-B inicial no sería insertar todos los elementos en el conjunto inicial sucesivamente, si no construir el conjunto inicial de nodos hoja directamente desde la entrada, y después construir los nodos internos a partir de este conjunto. Inicialmente, todas las hojas excepto la última tienen un elemento más, el cual será utilizado para construir los nodos internos. Por ejemplo, si los nodos hoja tienen un tamaño máximo de 4 y el conjunto inicial es de enteros desde el 1 al 24, tenemos que construir inicialmente 5 nodos hoja conteniendo 5 valores cada uno (excepto el último que contiene 4):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | 5 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 6 | 7 | 8 | 9 | 10 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 11 | 12 | 13 | 14 | 15 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 16 | 17 | 18 | 19 | 20 | | |  |  |  |  | | --- | --- | --- | --- | | 21 | 22 | 23 | 24 | |

Construiremos el siguiente nivel hacia arriba desde las hojas tomando el último elemento de cada hoja

**Tipos de Scan sobre los índices**

Un Index Scan (Escaneo en el índice) refiere a la búsqueda y obtención de una o más filas a través de la estructura de un índice, utilizando el valor de la columna indexada, especificada en la sentencia SQL. "El principio básico de los índices en Oracle indica que encontrar un valor requiere de *n* lecturas, donde *n* es el peso del árbol".

Al realizar un Index Scan pueden suceder dos cosas:

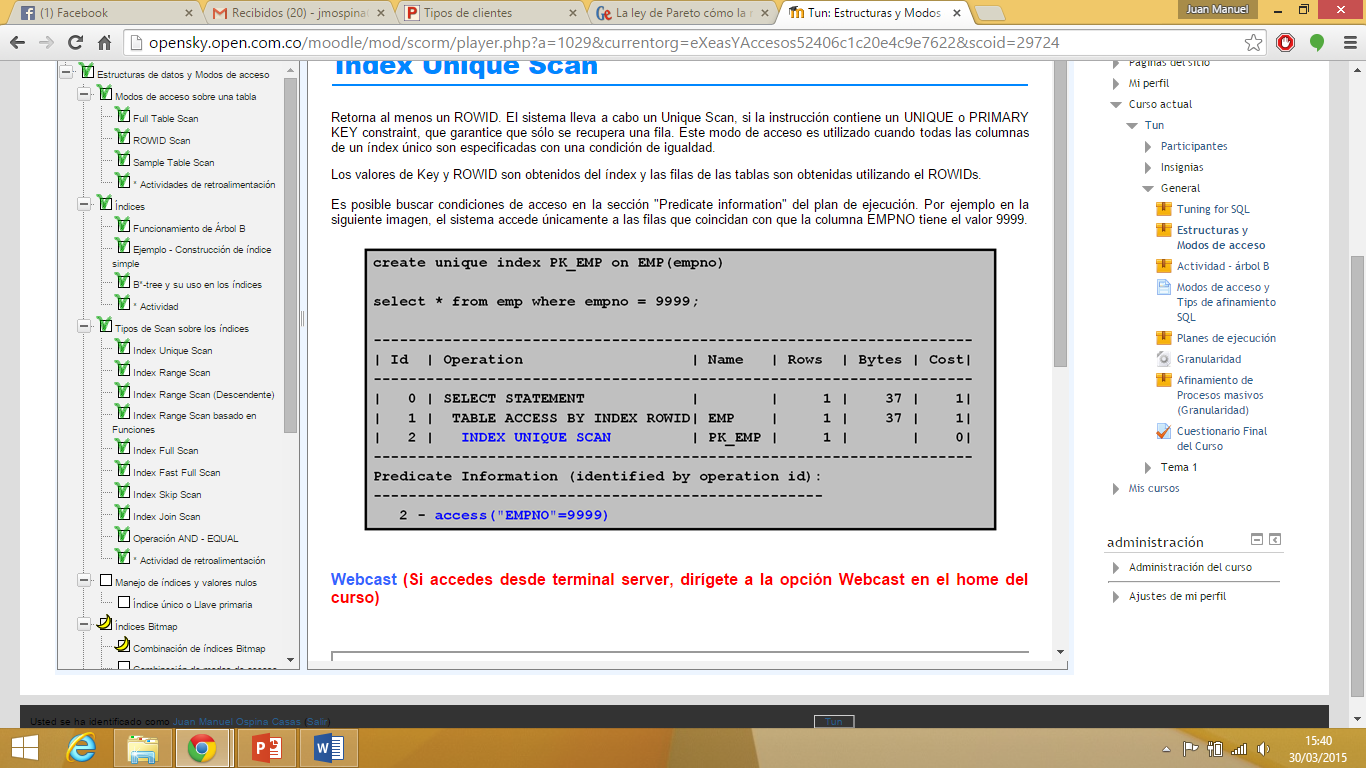
1. Si la sentencia SQL sólo requiere obtener la columna o las columnas del índice, las toma directamente de dicho índice sin ir a la tabla.
2. El índice contiene no sólo el valor indexado, sino también el ROWID de los registros de la tabla que tienen un valor. Por tanto, si la instrucción requiere de columnas adicionales a las indexadas, el sistema puede encontrar las filas en la tabla utilizando el acceso a la tabla con el ROWID o con un clúster Scan.

# Index Unique Scan

Retorna al menos un ROWID. El sistema lleva a cabo un Unique Scan, si la instrucción contiene un UNIQUE o PRIMARY KEY constraint, que garantice que sólo se recupera una fila. Este modo de acceso es utilizado cuando todas las columnas de un índex único son especificadas con una condición de igualdad.

Los valores de Key y ROWID son obtenidos del índex y las filas de las tablas son obtenidas utilizando el ROWIDs.

Es posible buscar condiciones de acceso en la sección "Predicate information" del plan de ejecución. Por ejemplo en la siguiente imagen, el sistema accede únicamente a las filas que coincidan con que la columna EMPNO tiene el valor 9999



# Index Range Scan

Operación más común para el acceso selectivo de información. Puede ser limitado (en ambas vías) o ilimitado (en una o ambas vías). Los datos son retornados en orden ascendente según las columnas de los índices utilizados. Las filas con valores idénticos son ordenadas ascendentemente según su ROWID.

El optimizador utiliza un Range Scan cuando encuentra una o más columnas principales de un índice especificado, en las condiciones (Cláusula WHERE), como por ejemplo:

WHERE col1 = :b1, col1 < :b1, col1 > :b1.

Y cualquier combinación de condiciones precedentes.

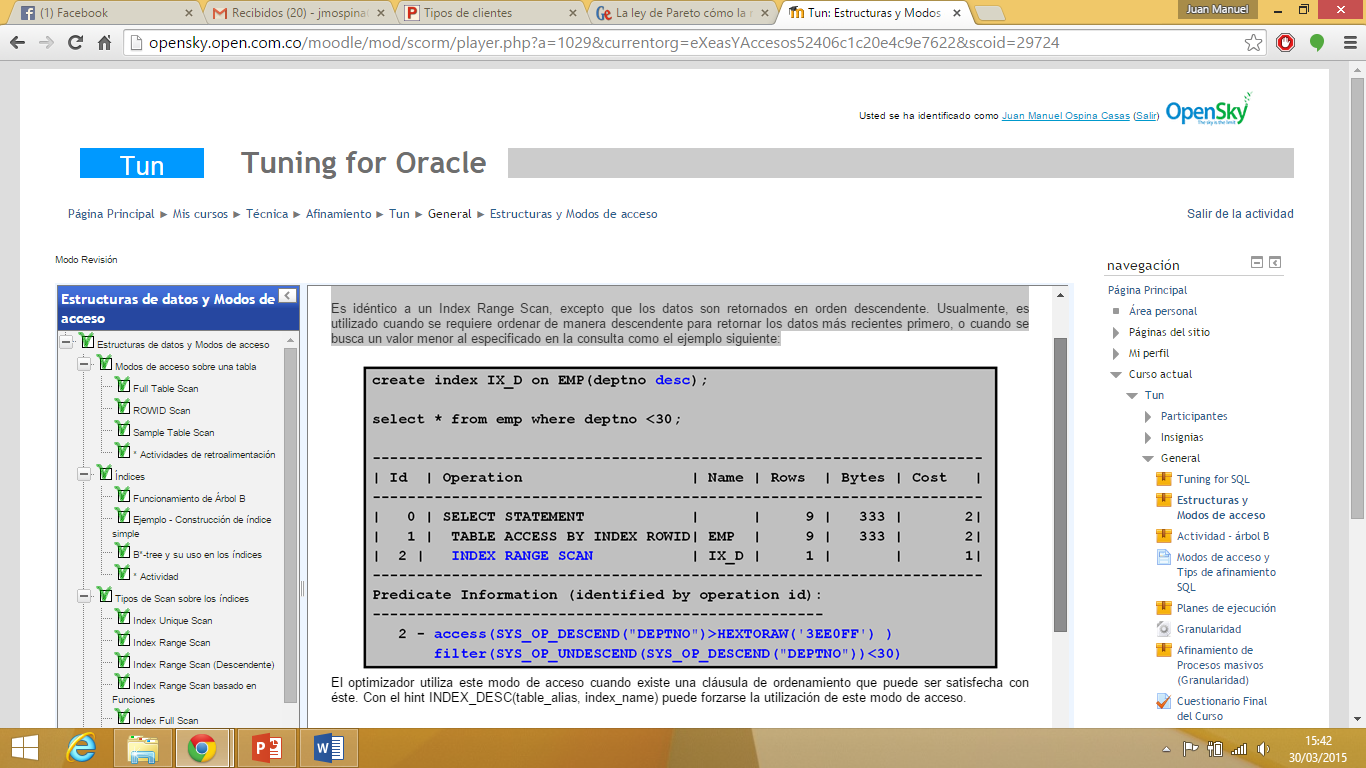
Las búsquedas con caracteres comodines (col1 like ‘%ASD') no deben estar en una posición principal ya que elimina la utilización de un Range Scan.

Este modo de acceso puede utilizar índices únicos o NO únicos, y puede entregar el resultado desordenado cuando los índices constituyen una cláusula ORDER BY/GROUP BY y dichas columnas son NOT NULL.

 En el siguiente ejemplo, utilizando el índice de la columna I\_DEPTNO, el sistema accede a las filas para las cuales EMP.DEPTNO=10. Se obtienen sus ROWIDs, se recuperan las columnas de la tabla EMP y finalmente, se aplica el filtro de la condición EMP.SAL > 1000 sobre las filas recuperadas para obtener el resultado.

# Index Range Scan (Descendente)

Es idéntico a un Index Range Scan, excepto que los datos son retornados en orden descendente. Usualmente, es utilizado cuando se requiere ordenar de manera descendente para retornar los datos más recientes primero, o cuando se busca un valor menor al especificado en la consulta como el ejemplo siguiente:

  
El optimizador utiliza este modo de acceso cuando existe una cláusula de ordenamiento que puede ser satisfecha con éste. Con el hint INDEX\_DESC(table\_alias, index\_name) puede forzarse la utilización de este modo de acceso.

Ten en cuenta que...

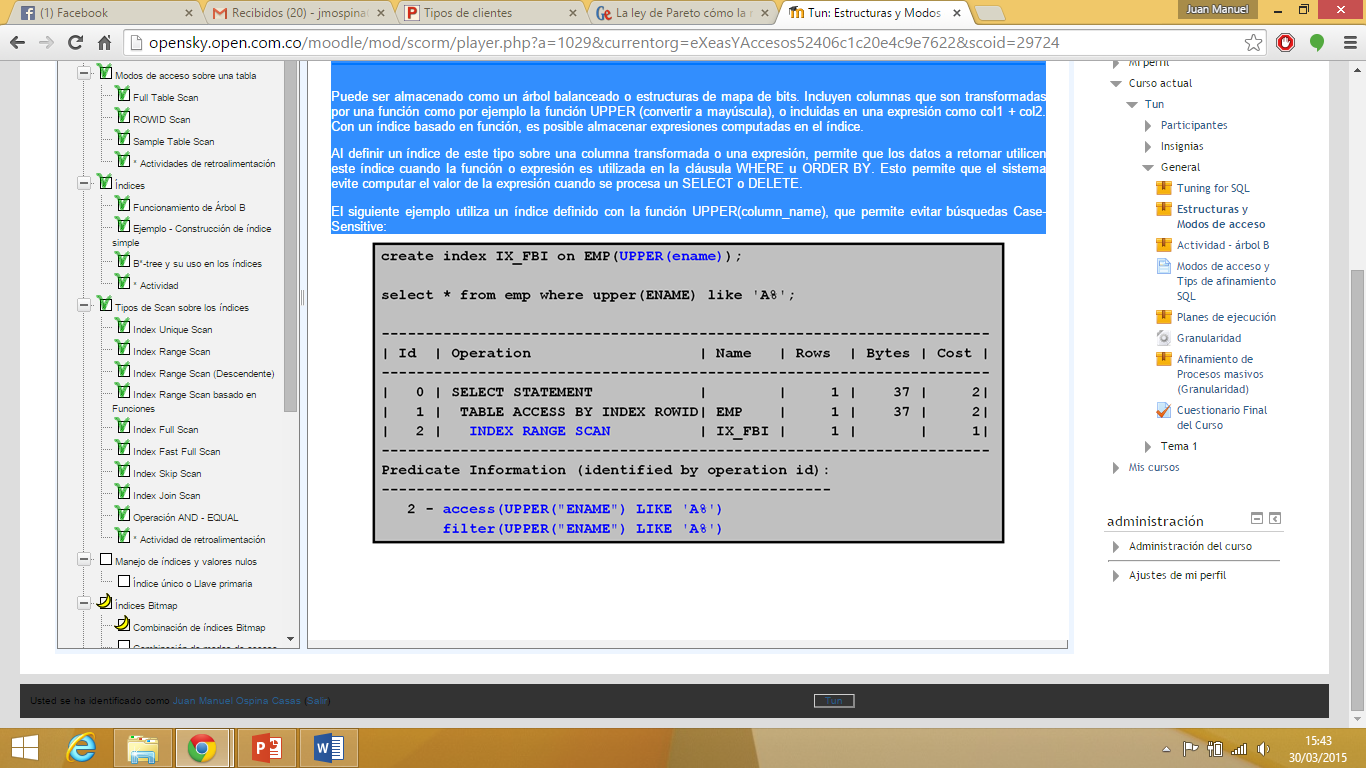
El sistema trata este modo de acceso como un índice basado en función. Las columnas marcadas con DESC en la instrucción, son almacenadas en un orden descendente especial dentro de la estructura del índice que es reversado de nuevo utilizando la función SYS\_op\_UNDESCEND.

# Index Range Scan basado en Funciones

Puede ser almacenado como un árbol balanceado o estructuras de mapa de bits. Incluyen columnas que son transformadas por una función como por ejemplo la función UPPER (convertir a mayúscula), o incluidas en una expresión como col1 + col2. Con un índice basado en función, es posible almacenar expresiones computadas en el índice.

Al definir un índice de este tipo sobre una columna transformada o una expresión, permite que los datos a retornar utilicen este índice cuando la función o expresión es utilizada en la cláusula WHERE u ORDER BY. Esto permite que el sistema evite computar el valor de la expresión cuando se procesa un SELECT o DELETE.

El siguiente ejemplo utiliza un índice definido con la función UPPER(column\_name), que permite evitar búsquedas Case-Sensitive:

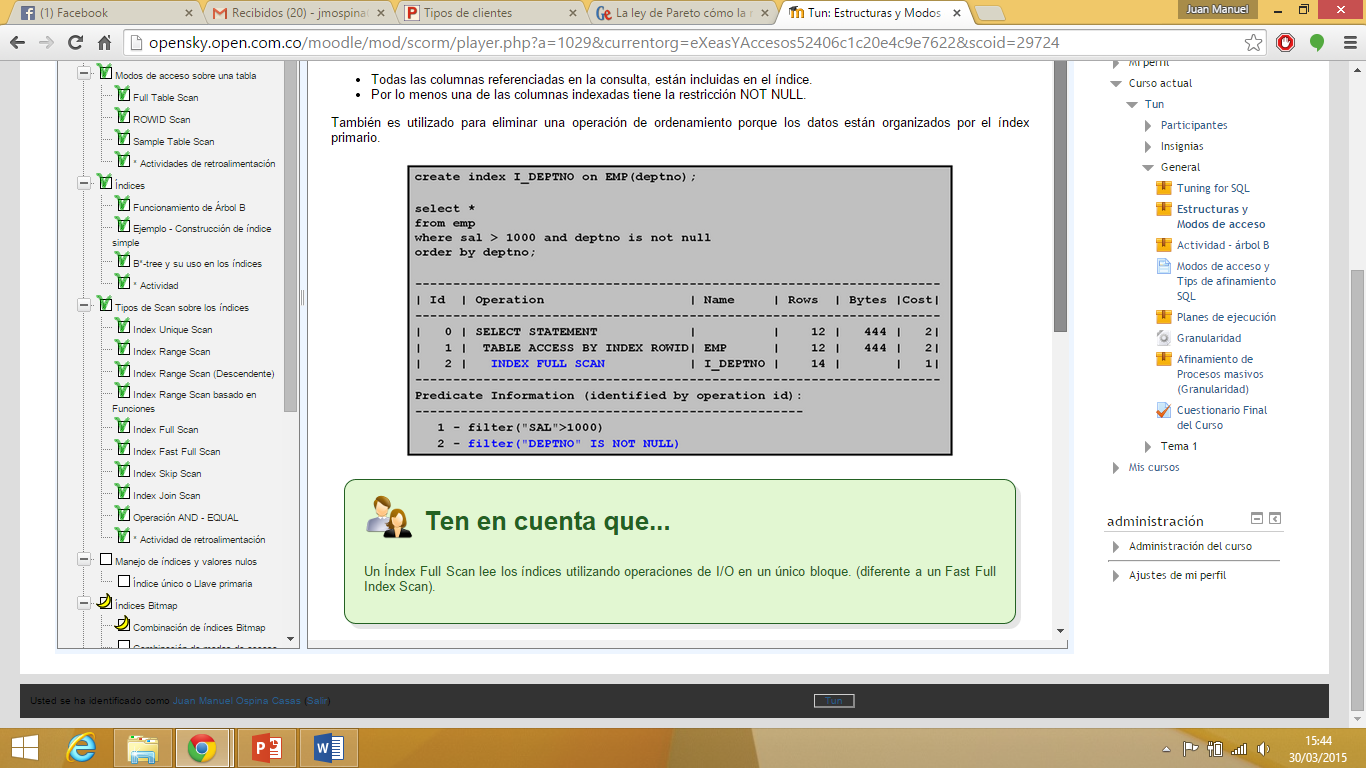


**Index Full Scan**

Se utiliza cuando el predicado referencia a una de las columnas con índice. El predicado no necesita ser un índice conductor (columna principal). El Full Scan también está disponible cuando no existe predicado y se cumplen las siguientes condiciones:

* Todas las columnas referenciadas en la consulta, están incluidas en el índice.
* Por lo menos una de las columnas indexadas tiene la restricción NOT NULL.

También es utilizado para eliminar una operación de ordenamiento porque los datos están organizados por el índex primario.



## Ten en cuenta que...

Un Índex Full Scan lee los índices utilizando operaciones de I/O en un único bloque. (diferente a un Fast Full Index Scan).

# Index Fast Full Scan

Es una alternativa para realizar Full Scan sobre una tabla cuando el índice contiene todas las columnas que requiere la consulta y por lo menos una columna en el índex key tiene la restricción NOT NULL. Un Índex Fast Full Scan accede a los datos por el índice sin acceder a la tabla.

No puede ser utilizado para eliminar una operación de ordenado ya que los datos no están ordenados por el índex key. Puede ser utilizado para funciones agregadas como min/avg/sum. En este caso, el optimizador debe conocer que todas las filas de la tabla están representadas en el índice y por lo menos una columna con la restricción NOT NULL.

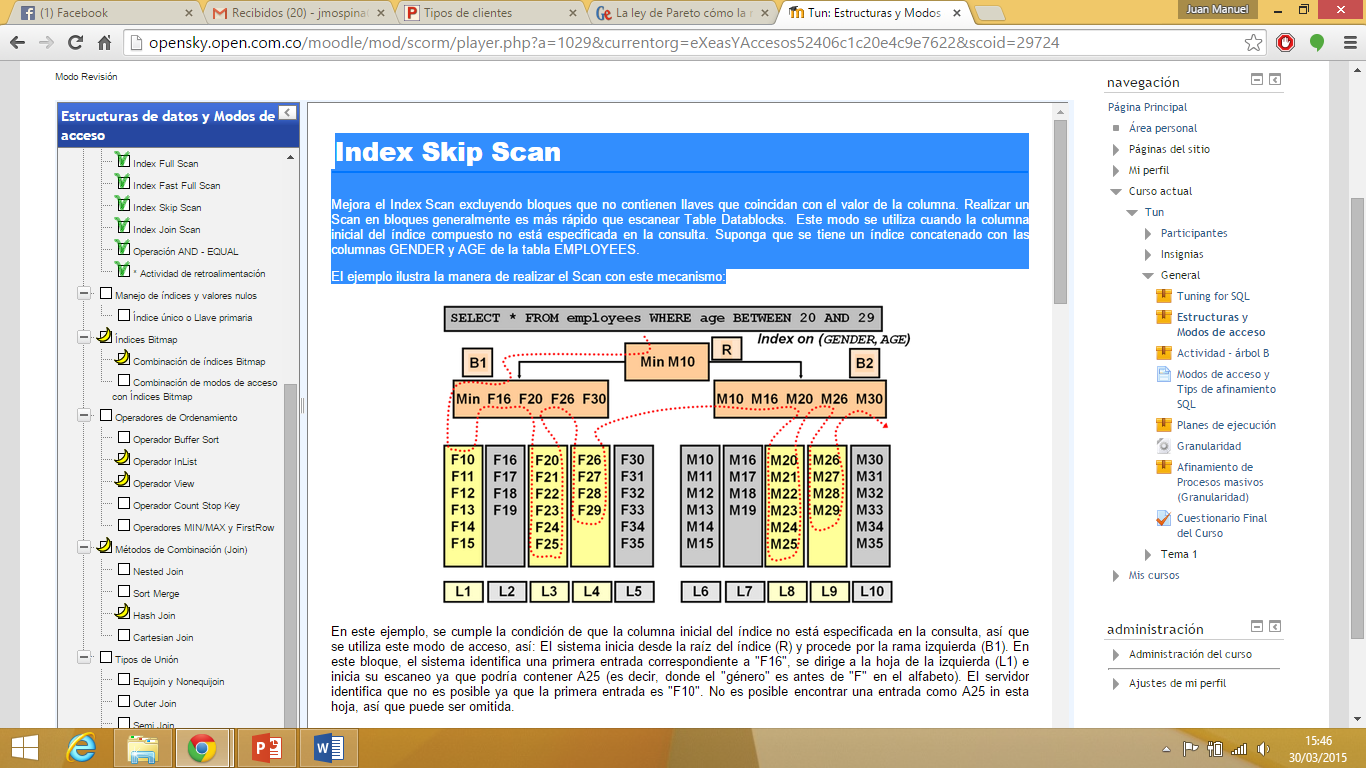
Esta operación lee el índice completo utilizando lectoras multi bloque (a diferencia de un Full Index Scan). Este modo no puede ser realizado contra índices Bitmap. Este modo es más rápido que el anterior porque puede utilizar I/O multibloque como un Table Scan.

Es posible especificar el uso de este modo con el parámetro OPTIMIZER\_FEATURES\_ENABLE o el hint INDEX\_FFS, así:

# Index Skip Scan

Mejora el Index Scan excluyendo bloques que no contienen llaves que coincidan con el valor de la columna. Realizar un Scan en bloques generalmente es más rápido que escanear Table Datablocks.  Este modo se utiliza cuando la columna inicial del índice compuesto no está especificada en la consulta. Suponga que se tiene un índice concatenado con las columnas GENDER y AGE de la tabla EMPLOYEES.

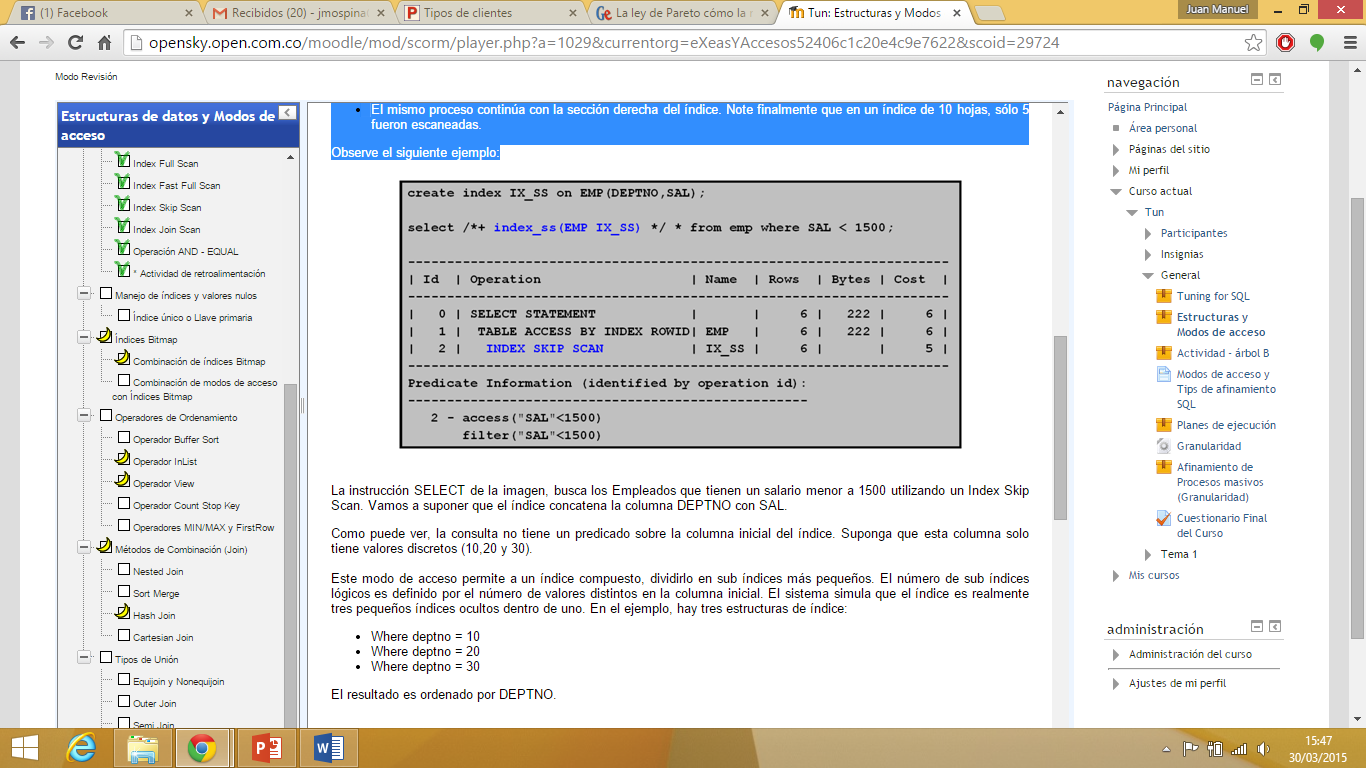
El ejemplo ilustra la manera de realizar el Scan con este mecanismo:



En este ejemplo, se cumple la condición de que la columna inicial del índice no está especificada en la consulta, así que se utiliza este modo de acceso, así: El sistema inicia desde la raíz del índice (R) y procede por la rama izquierda (B1). En este bloque, el sistema identifica una primera entrada correspondiente a "F16", se dirige a la hoja de la izquierda (L1) e inicia su escaneo ya que podría contener A25 (es decir, donde el "género" es antes de "F" en el alfabeto). El servidor identifica que no es posible ya que la primera entrada es "F10". No es posible encontrar una entrada como A25 in esta hoja, así que puede ser omitida.

* Retorna a la primer rama (B1), donde el servidor identifica que el siguiente sub árbol (F16) no necesita ser escaneado ya que la siguiente entrada en (B1) es F20, es decir, no contiene valores que complain con la restricción. Así que el servidor omite el Scan para (L2).
* Retorna nuevamente a (B1) y el servidor encuentra que las siguientes dos entradas tienen en común el prefijo de F2, lo cual indica que son sub árboles donde pueden existir valores que cumplen con la condición. El sistema sabe que los sub árboles están ordenados por edad.
* Entonces se escanean las hojas L3 y L4, recuperando los valores encontrados. Al examinar la cuarta entrada en la primer rama (B1), el sistema determina que no existen más posibilidades de encontrar una entrada F2x y por ende, de encontrar valores que satisfagan la condición.
* El mismo proceso continúa con la sección derecha del índice. Note finalmente que en un índice de 10 hojas, sólo 5 fueron escaneadas.

Observe el siguiente ejemplo:

  
La instrucción SELECT de la imagen, busca los Empleados que tienen un salario menor a 1500 utilizando un Index Skip Scan. Vamos a suponer que el índice concatena la columna DEPTNO con SAL.

Como puede ver, la consulta no tiene un predicado sobre la columna inicial del índice. Suponga que esta columna solo tiene valores discretos (10,20 y 30).

Este modo de acceso permite a un índice compuesto, dividirlo en sub índices más pequeños. El número de sub índices lógicos es definido por el número de valores distintos en la columna inicial. El sistema simula que el índice es realmente tres pequeños índices ocultos dentro de uno. En el ejemplo, hay tres estructuras de índice:

* Where deptno = 10
* Where deptno = 20
* Where deptno = 30

El resultado es ordenado por DEPTNO.

## Ten en cuenta que...

El Skip Scan es ideal si hay pocos valores distintos en la columna principal del índice compuesto, y muchos valores distintos en la otra columna del índice .

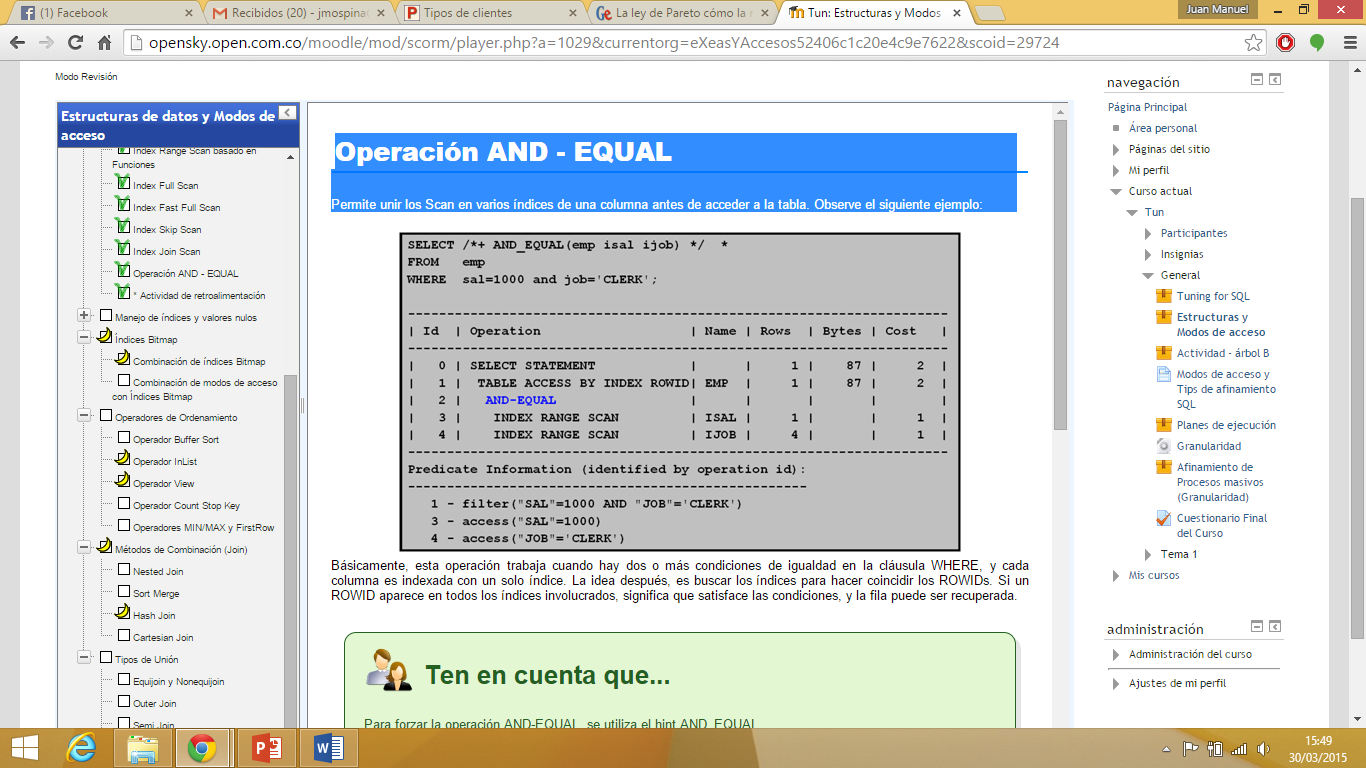
# Index Join Scan

Un Index Join Scan es un Hash Join de muchos índices, que juntos contienen todas las columnas de la tabla referenciadas en la consulta. Si un Index Join es utilizado, no es necesario acceder a la tabla ya que todos los valores necesarios pueden ser recuperados de los índices. No puede ser utilizado para eliminar una operación de ordenamiento.

El Index Join Scan no es realmente una operación de unión, pero es construido utilizando un acceso al índice seguido por una operación join de ROWID. Observe el siguiente ejemplo:

# Operación AND - EQUAL

Permite unir los Scan en varios índices de una columna antes de acceder a la tabla. Observe el siguiente ejemplo:



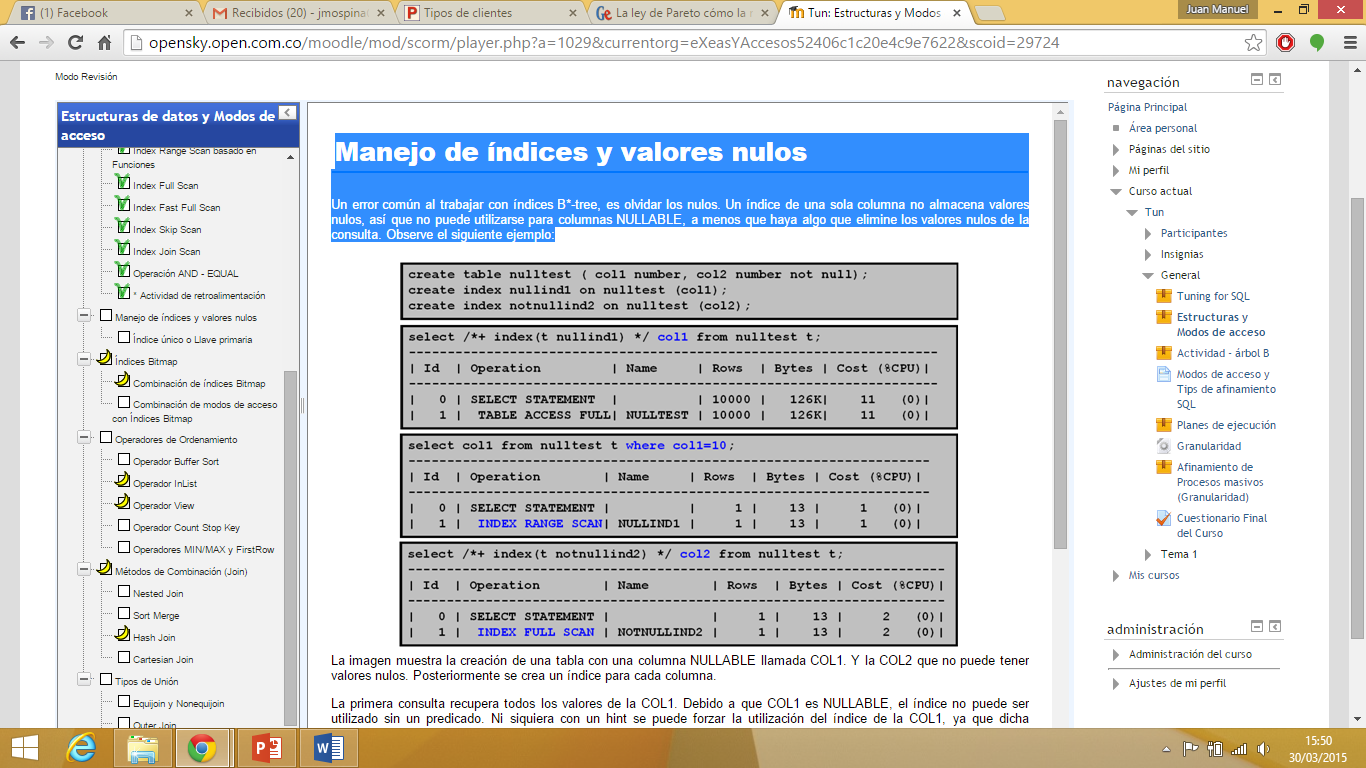
Básicamente, esta operación trabaja cuando hay dos o más condiciones de igualdad en la cláusula WHERE, y cada columna es indexada con un solo índice. La idea después, es buscar los índices para hacer coincidir los ROWIDs. Si un ROWID aparece en todos los índices involucrados, significa que satisface las condiciones, y la fila puede ser recuperada.

Ten en cuenta que...

Para forzar la operación AND-EQUAL, se utiliza el hint AND\_EQUAL.

# Manejo de índices y valores nulos

Un error común al trabajar con índices B\*-tree, es olvidar los nulos. Un índice de una sola columna no almacena valores nulos, así que no puede utilizarse para columnas NULLABLE, a menos que haya algo que elimine los valores nulos de la consulta. Observe el siguiente ejemplo:



La imagen muestra la creación de una tabla con una columna NULLABLE llamada COL1. Y la COL2 que no puede tener valores nulos. Posteriormente se crea un índice para cada columna.

La primera consulta recupera todos los valores de la COL1. Debido a que COL1 es NULLABLE, el índice no puede ser utilizado sin un predicado. Ni siquiera con un hint se puede forzar la utilización del índice de la COL1, ya que dicha columna es NULLABLE. Como sólo se buscan los valores de la COL1, no es necesario leer la tabla.

Sin embargo, en la segunda consulta, el efecto del predicado contra la COL1 resulta en la eliminación de los nulos de los datos que retorna la columna. Esto hace que el índice pueda ser utilizado.

La tercera consulta puede utilizar el índice directamente ya que corresponde a la columna declarada como NOT NULL.

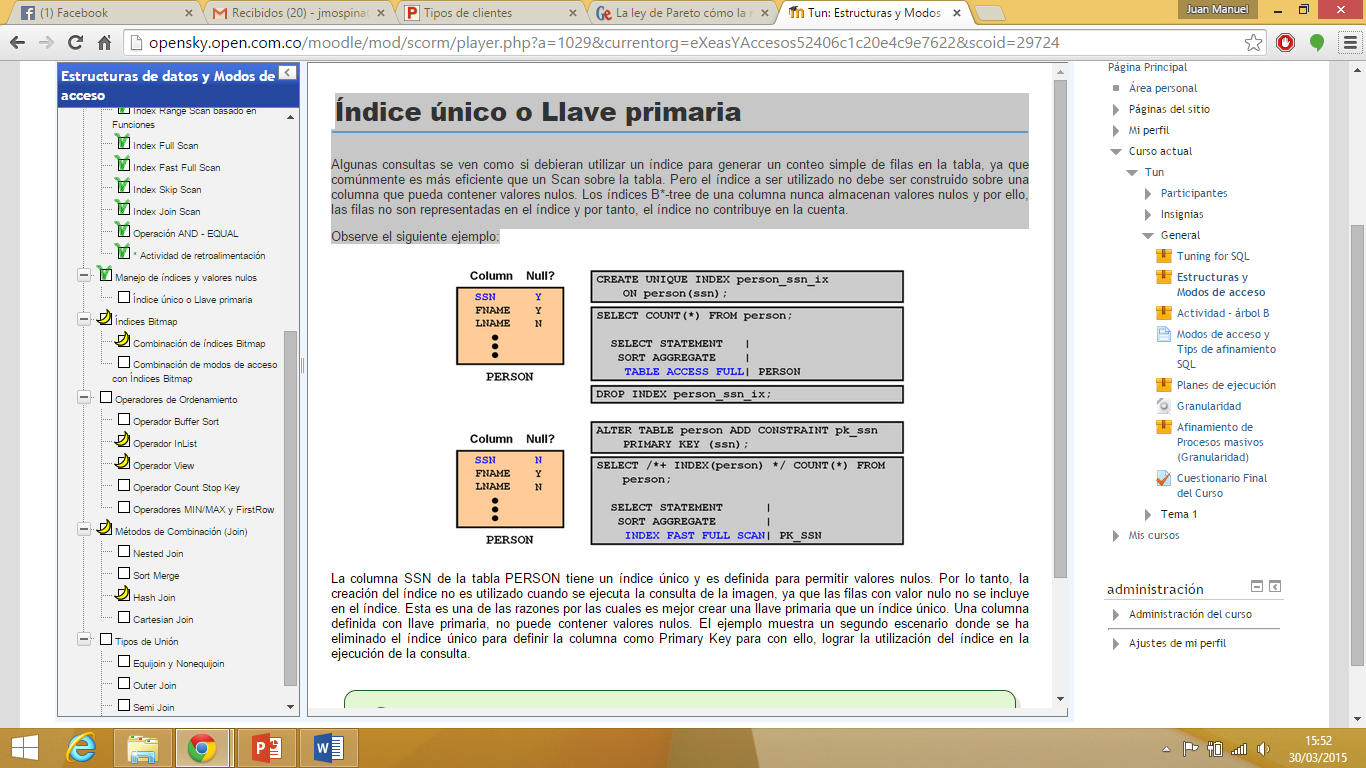
Ten en cuenta que...

El índice también podría ser utilizado para forzar el retorno de únicamente valores no nulos, utilizando el predicado IS NOT NULL, para la columna COL1.

# Índice único o Llave primaria

Algunas consultas se ven como si debieran utilizar un índice para generar un conteo simple de filas en la tabla, ya que comúnmente es más eficiente que un Scan sobre la tabla. Pero el índice a ser utilizado no debe ser construido sobre una columna que pueda contener valores nulos. Los índices B\*-tree de una columna nunca almacenan valores nulos y por ello, las filas no son representadas en el índice y por tanto, el índice no contribuye en la cuenta.

Observe el siguiente ejemplo:



La columna SSN de la tabla PERSON tiene un índice único y es definida para permitir valores nulos. Por lo tanto, la creación del índice no es utilizado cuando se ejecuta la consulta de la imagen, ya que las filas con valor nulo no se incluye en el índice. Esta es una de las razones por las cuales es mejor crear una llave primaria que un índice único. Una columna definida con llave primaria, no puede contener valores nulos. El ejemplo muestra un segundo escenario donde se ha eliminado el índice único para definir la columna como Primary Key para con ello, lograr la utilización del índice en la ejecución de la consulta.

Ten en cuenta que...

La restricción PRIMARY KEY combina la restricción de NOT NULL y la restricción de valores únicos.

# Índices Bitmap

En un B\*-tree, existe una relación uno a uno entre una entrada del índice y una fila. Un índice Bitmap está organizado como un índice B\*-tree pero con la particularidad que una entrada utiliza un mapa de bits para apuntar a muchas filas simultáneamente. Si el índice Bitmap envuelve más de una columna, se crea un Bitmap por cada posible combinación. Cada Bitmap almacena el primero y último ROWID. A partir de estos valores, el sistema utiliza un algoritmo interno para asignar mapas de bit en ROWIDs. Esto es posible gracias a que el sistema conoce el número máximo posible de filas que pueden ser almacenadas en un sistema de bloques. Cada posición en un Bitmap, mapea a una fila potencial en la tabla inclusive si la fila no existe. El contenido de la posición en el Bitmap para un valor particular indica si la fila tiene el valor en las columnas del Bitmap. El valor almacenado es 1 si los valores de la fila cumplen con la condición, de lo contrario es 0. Los índices Bitmap son utilizados principalmente en ambientes de Data warehousing, que típicamente tienen volúmenes de datos amplios y consultas Ad Hoc sin concurrencia en las transacciones ya que un Bitmap bloqueado, bloquea un conjunto amplio de filas en la tabla al mismo tiempo.

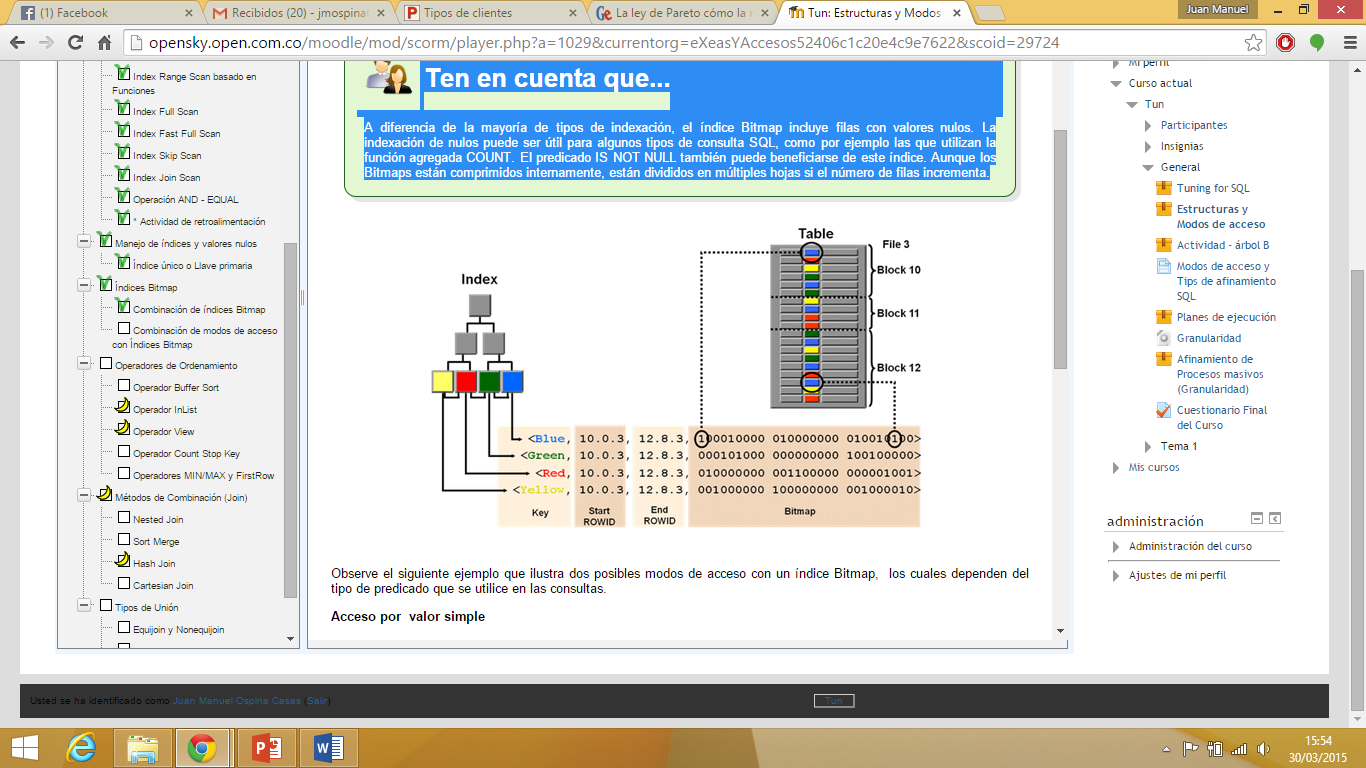
Para estas aplicaciones, la indexación con Bitmap provee tiempos de respuesta reducidos para una amplia cantidad de consultas Ad Hoc, reduce los requerimientos de almacenamiento en comparación con otras técnicas de indexación, incrementa el performance incluso en hardware gracias a un número relativamente pequeño de llamadas a CPU o una pequeña cantidad de memoria, y un mantenimiento eficiente durante operaciones DML paralelas y cargas.

## Ten en cuenta que...

A diferencia de la mayoría de tipos de indexación, el índice Bitmap incluye filas con valores nulos. La indexación de nulos puede ser útil para algunos tipos de consulta SQL, como por ejemplo las que utilizan la función agregada COUNT. El predicado IS NOT NULL también puede beneficiarse de este índice. Aunque los Bitmaps están comprimidos internamente, están divididos en múltiples hojas si el número de filas incrementa.

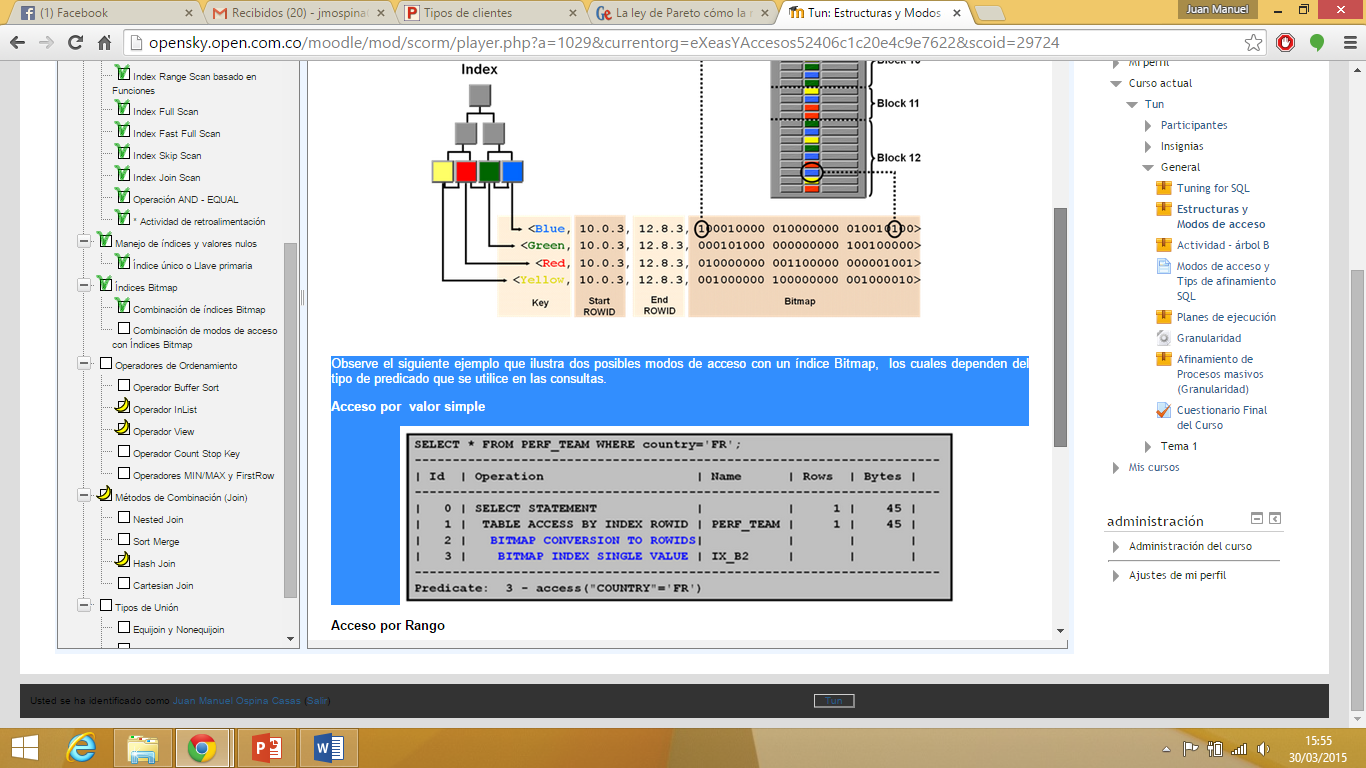
Observe el siguiente ejemplo que ilustra dos posibles modos de acceso con un índice Bitmap,  los cuales dependen del tipo de predicado que se utilice en las consultas.

**Acceso por  valor simple**

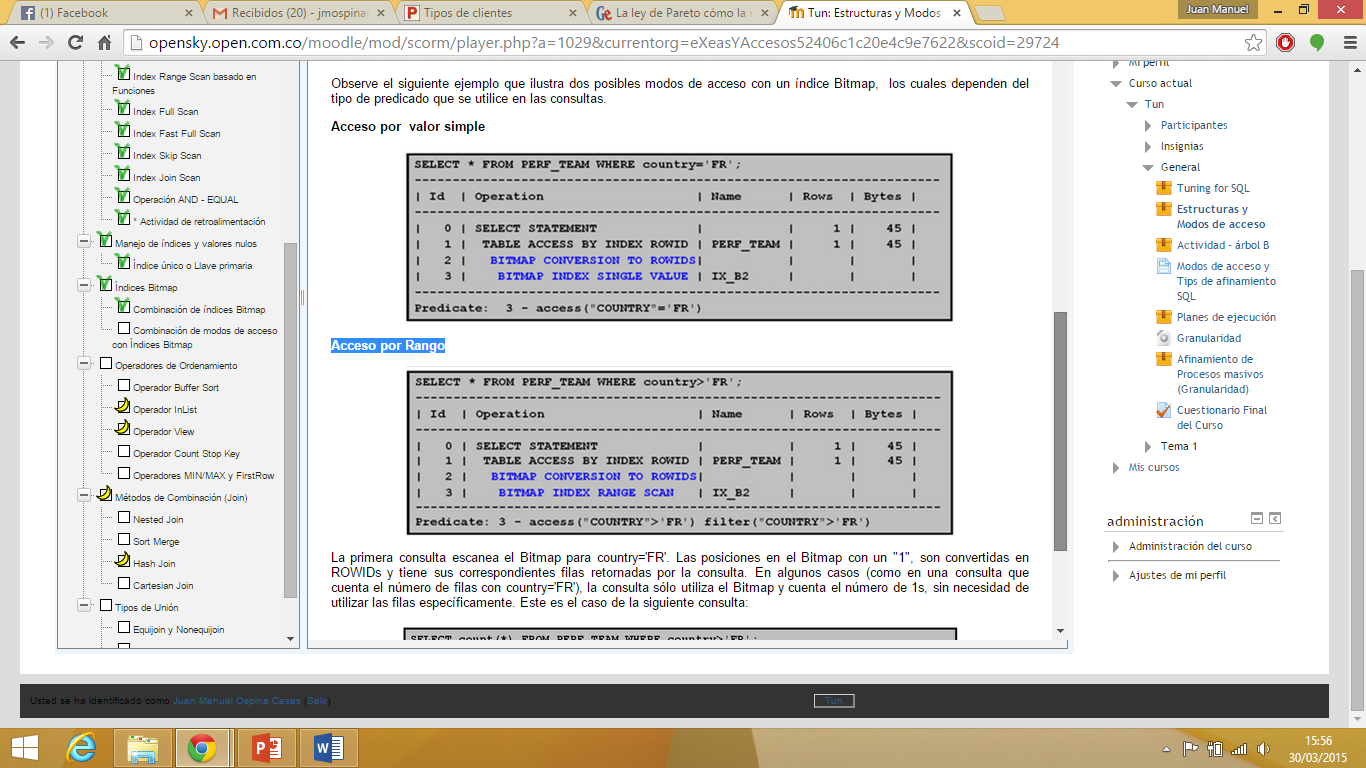


Observe el siguiente ejemplo que ilustra dos posibles modos de acceso con un índice Bitmap,  los cuales dependen del tipo de predicado que se utilice en las consultas.

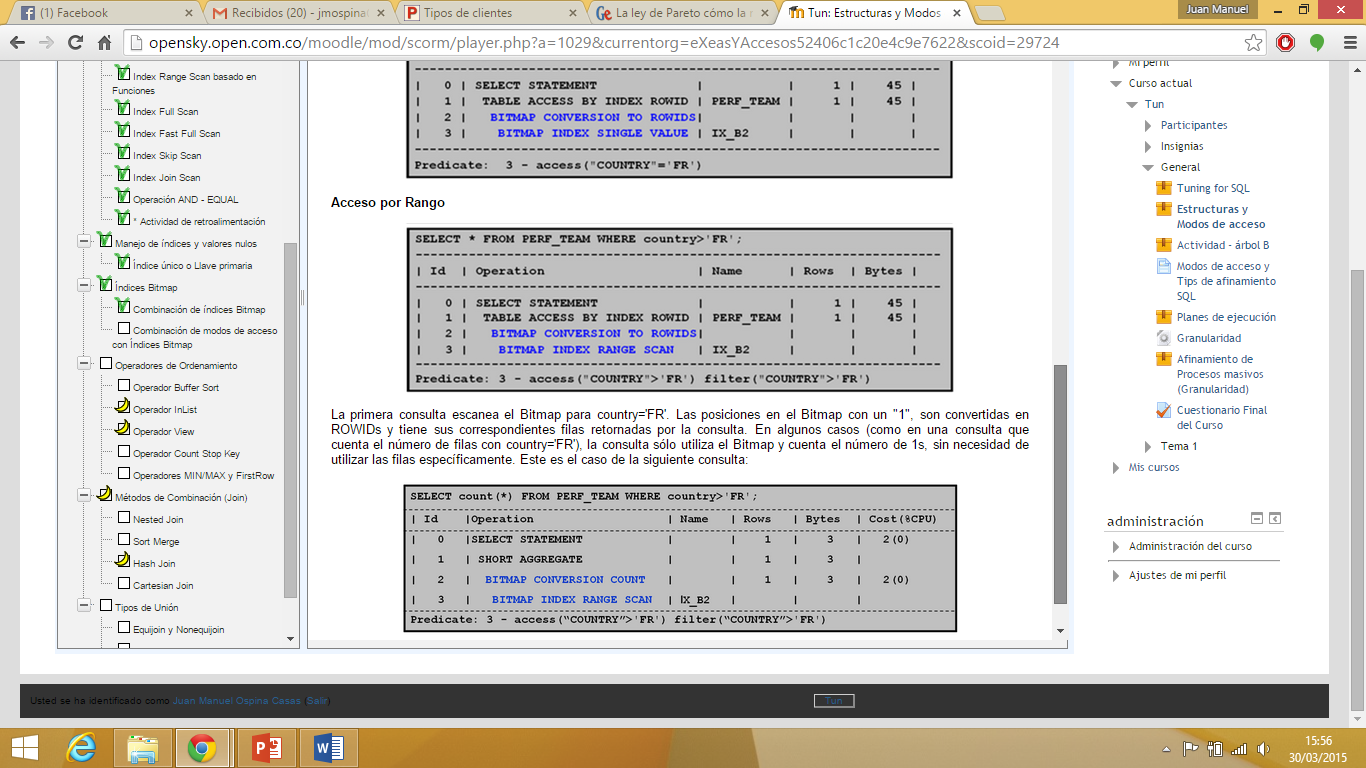
**Acceso por  valor simple**



**Acceso por Rango**

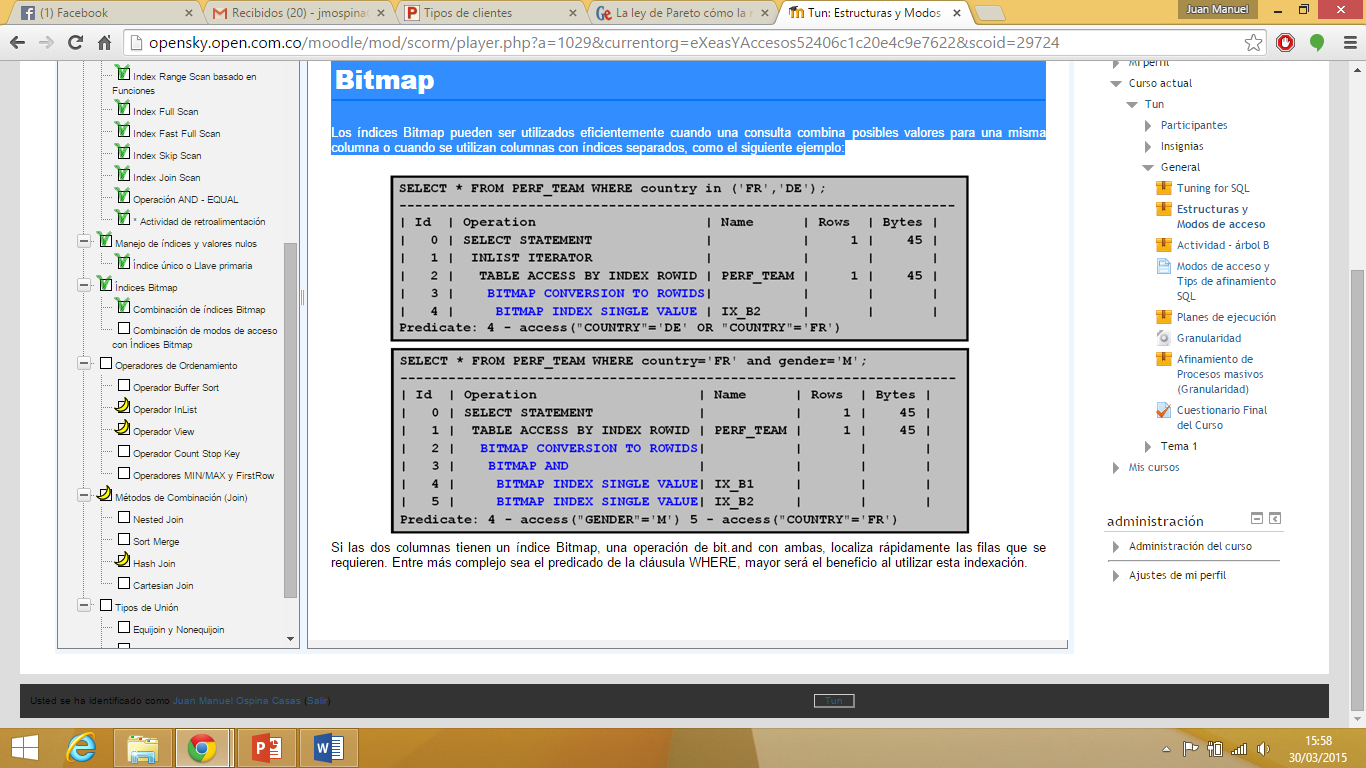


La primera consulta escanea el Bitmap para country='FR'. Las posiciones en el Bitmap con un "1", son convertidas en ROWIDs y tiene sus correspondientes filas retornadas por la consulta. En algunos casos (como en una consulta que cuenta el número de filas con country='FR'), la consulta sólo utiliza el Bitmap y cuenta el número de 1s, sin necesidad de utilizar las filas específicamente. Este es el caso de la siguiente consulta:



# Combinación de modos de acceso con Índices Bitmap

Los índices Bitmap pueden ser utilizados eficientemente cuando una consulta combina posibles valores para una misma columna o cuando se utilizan columnas con índices separados, como el siguiente ejemplo:



Si las dos columnas tienen un índice Bitmap, una operación de bit.and con ambas, localiza rápidamente las filas que se requieren. Entre más complejo sea el predicado de la cláusula WHERE, mayor será el beneficio al utilizar esta indexació