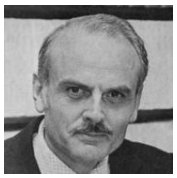
TEMA 3
MODELO RELACIONAL



3.1. CARACTERÍSTICAS DEL MODELO RELACIONAL.

El modelo relacional fue introducido por E. F. Codd en 1970 basado en 2 principales conceptos matemáticos:

- *Lógica de predicados* (lógica de primer orden): permite verificar si algo es verdadero o falso a través del uso de aserciones.
- *Teoría de conjuntos*: Se emplea para manejar conjuntos de datos y representa la base para realizar su manipulación.

El modelo relacional define 3 principales componentes:

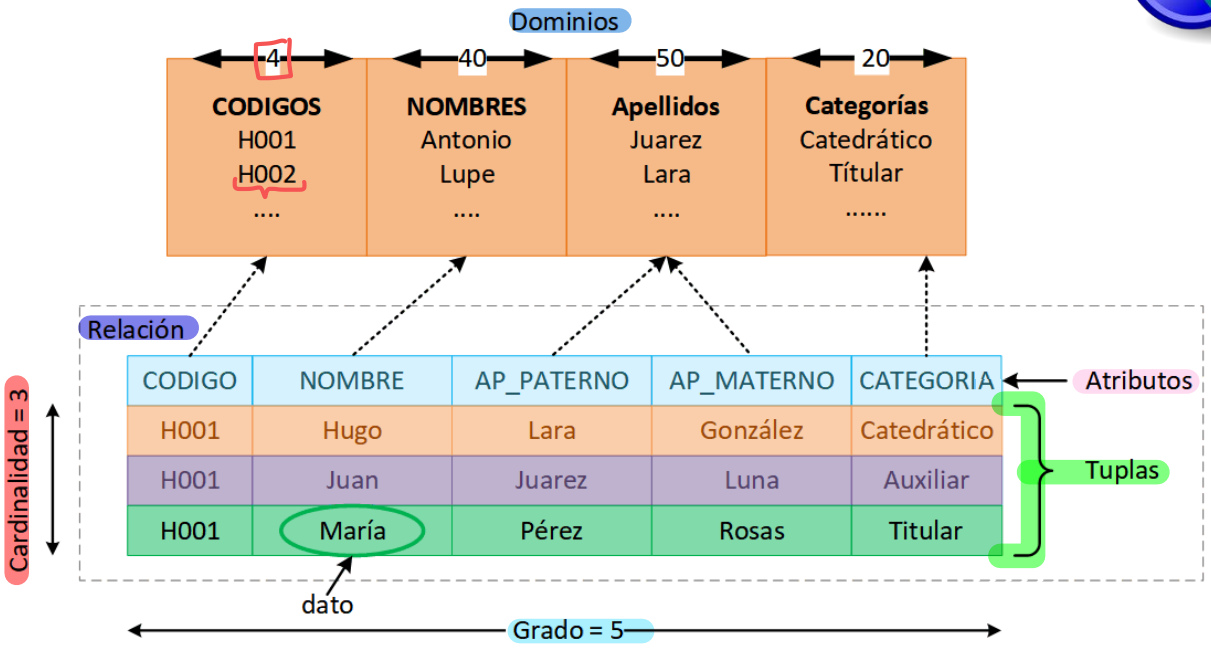
1. Proporciona una *estructura lógica* de los datos representada por *relaciones*.
   - Esta estructura lógica permite al diseñador enfocarse en la representación lógica de los datos y sus relaciones sin preocuparse de detalles en cuanto a su almacenamiento físico.
2. Un conjunto de *reglas de integridad* empleadas para garantizar que los datos permanecerán consistentes a lo largo del tiempo.
3. Un conjunto de *operaciones* que define la forma en la que se manipulan los datos.

Codd también propuso dos lenguajes para manipular los datos en las tablas: El *álgebra relacional* y el *cálculo relacional*.

- Ambos lenguajes soportan la manipulación de datos empleando *operadores lógicos* en lugar de punteros físicos empleados en el modelo jerárquico y de red.
- Para adquirir una mayor comprensión en cuanto al álgebra relacional y al cálculo relacional se retomarán al revisar la instrucción SQL `select` en temas posteriores.

3.2. ESTRUCTURA LÓGICA DE LOS DATOS.

La estructura lógica de los datos se basa en los siguientes conceptos:



- *Relación*: Corresponde con la idea general de una *tabla de datos*. El nombre relación es el término matemático. Entidad
- *Tupla*: Corresponde a cada fila de la tabla. Registro // Instancia de una entidad
- *Atributo*: Corresponde a cada una de las columnas de la tabla. Campo
- *Cardinalidad*: Número de tuplas de la tabla.
- *Grado*: Número de atributos de la tabla.
- *Dominio*: Rango de valores que puede tomar un atributo. Se puede definir también como una colección de valores a partir de la cual uno o más atributos toman sus valores reales.

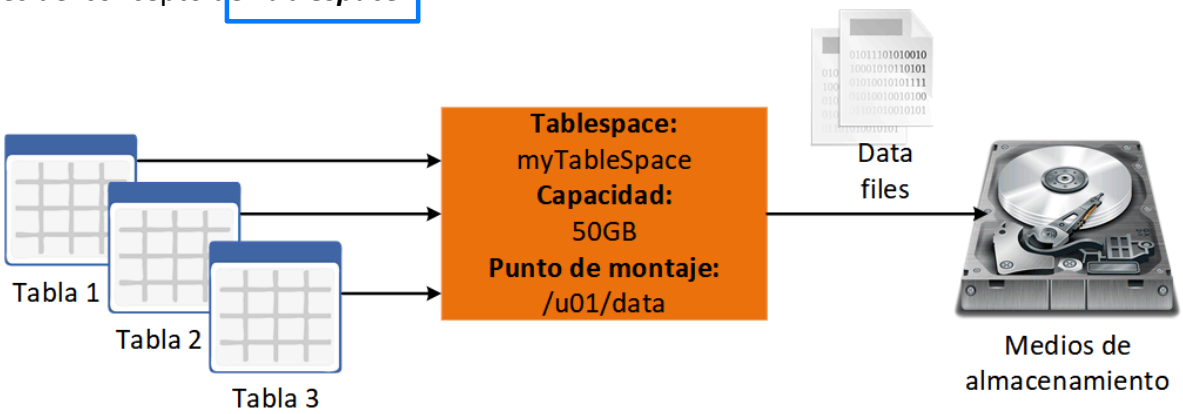
3.2.1. Tablas

- Representan la unidad primaria de almacenamiento y la implementación del concepto de ***relación***.
- Una tabla generalmente representa a una *entidad* identificada en el diseño conceptual.
- Su nombre debe ser único dentro de un mismo ***esquema*** (se le asigna el nombre de la entidad que representa).
- Un esquema está formado por un conjunto de objetos. En Oracle, todos los objetos de un esquema pertenecen a un usuario, al esquema se le asigna el nombre del usuario.
- Cada registro (tupla) representa la ocurrencia de una entidad (instancia de una entidad).
- Cada columna representa a un atributo de la entidad, y esta debe tener nombres únicos.
- Los datos de una columna deben ser del mismo tipo de dato.
- El orden en el que se definen las columnas es irrelevante.
- Cada columna tiene un rango de valores al que se le llama ***dominio.***
- Una tabla puede contar con una o varias columnas que identifiquen a un registro de manera única llamada ***llave primaria.***
- Una tabla puede contar con una o más columnas empleadas para relacionarse con otras llamadas ***llaves foráneas.***

3.2.1.1. Estructuras lógicas y físicas de almacenamiento en Oracle.

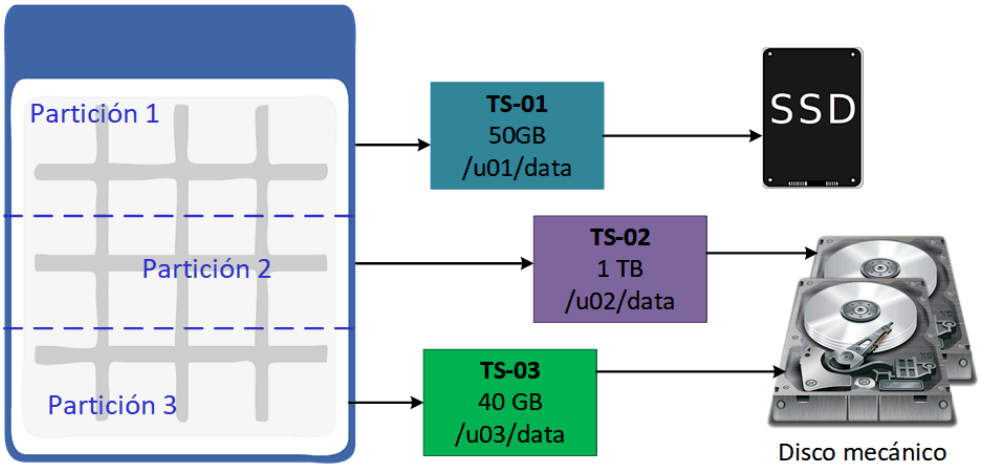
En sentido estricto, esta sección no corresponde a las características que ofrece el modelo relacional. Sin embargo, se considera esta sección para ilustrar las diferencias entre la estructura lógica de una tabla que ofrece el modelo relacional, y la estructura física de una tabla la cual es totalmente dependiente al RDBMS seleccionado, en este caso Oracle.

Físicamente, la tabla como tal no existe. En la mayoría de los manejadores, no se genera o existe un archivo por tabla. La organización del almacenamiento de la tabla por ejemplo en Oracle, se realiza a través del concepto de ***Tablespace.***



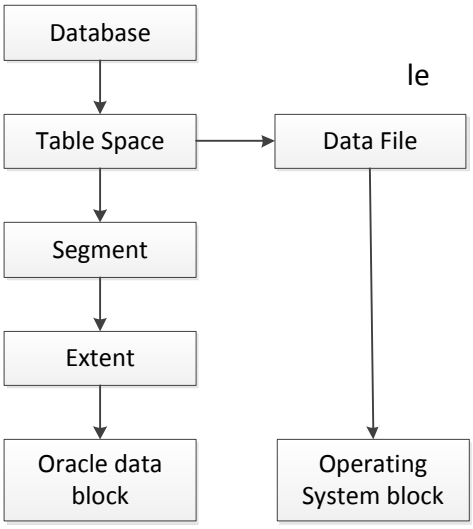
- Varias tablas pueden estar asociadas a un mismo tablespace.
- Un tablespace es un término abstracto, y se define en términos de los siguientes elementos:
  - Nombre
  - Capacidad
  - Ubicación o punto de montaje.
- Cuando se crea una tabla, en la mayoría de los manejadores, no es necesario especificar su tablespace. El manejador le asigna uno por default.

Por el contrario, una tabla puede tener un solo tablespace, o ser ***particionada*** en múltiples tablespaces para grandes cantidades de datos.



Estructuras de almacenamiento lógico en Oracle.

- *Segmento*: Objeto en la base de datos que requiere almacenamiento: tabla, índice, etc. Cuando se crea una tabla, se le asigna un segmento.
- *Extensión*: Segmentos formados por un conjunto de bloques de datos contiguos empleados para almacenar datos específicos.
- *Bloque de datos*: Unidad mínima de lectura y escritura. Generalmente del mismo tamaño que un bloque de datos a nivel de sistema operativo.



Estructura lógica de almacenamiento.

Estructura fíisica de almacenamiento.

Tarea: Investigar el nombre del tablespace por default que se le asigna a un usuario para almacenar sus objetos en una BD Oracle. Considerar que la base fue creada con dbca.

3.3. Restricciones (constraints).

Las restricciones, como se comentó anteriormente, representan uno de los elementos de un modelo de datos. Para el caso del modelo relacional, se considera la siguiente clasificación:

Nombre de la restricción	Implementación en un RDBMS
Restricciones de llave primaria	PRIMARY KEY
Restricciones de referencia	FOREIGN KEY
Restricciones de integridad	• CHECK • NULL, NOT NULL • UNIQUE
Validación por trigger	• TRIGGER

3.3.1. Restricciones de llave primaria.

- Una llave primaria (PK) es una columna o conjunto de columnas cuyo(s) valor(es) identifican de manera única a un registro.
- El RDBMS implementa esta restricción haciendo uso de los 2 siguientes elementos:
  - Un índice tipo Unique
  - Una restricción de integridad NOT NULL.

Ejemplo:

Observar la siguiente tabla de datos que corresponde a un catálogo de automóviles. ¿Qué atributo(s) podrán actuar como llave primaria?

AUTOMOVIL

C1 C2 C3 C4 C5 C6

numero_matricula	numero_motor	marca	modelo	Año	País_ origen
CCA-341	91234908123	Toyota	Yaris	2005	Japón
OFG-851	53489787679	Fiat	Fiorino	2008	Tailandia
XTV-657	30752312386	Ford	Mustang	2011	México
WGB-959	50934187123	Toyota	Avensis	2010	Tailandia

Los siguientes puntos permiten confirmar o ayudar a identificar a la PK de una tabla.

- Dependencia funcional.
- Consideraciones requeridas.
- Consideraciones opcionales.
- Tipos de llaves primarias.

3.3.1.1. Dependencia funcional.

Consiste en determinar y verificar el o los campos que actuarán o tendrán el papel de PK.

Definición: El atributo "B" es funcionalmente dependiente de un atributo "A" si cada valor de la columna "A" determina uno y solo un valor de la columna "B". Se expresa:

$$A \rightarrow B$$

De lo anterior, se puede concluir que "A" puede tomar el rol de llave primaria, siempre y cuando A, también pueda determinar el valor de las demás columnas. Es decir:

$$A \rightarrow B, C, D, E \ldots$$

Ejemplo:

Considerando los campos de la tabla anterior, iniciando en C_1, C_2, hasta C_6 de izquierda a derecha, determinar si las siguientes expresiones son verdaderas

- ¿C_4 será funcionalmente dependiente de C_3? (C_3 -> C_4) , Respuesta: FALSE.
- ¿C_4 ->C_1? (¿C_1 será funcionalmente dependiente de C_4?), Respuesta: FALSE
- ¿**C_1**->C_4? (¿C_1 será funcionalmente dependiente de C_4?), Respuesta: TRUE
- ¿**C_2**->C_4? (¿C_2 será funcionalmente dependiente de C_4?), Respuesta: TRUE

C_1 y **C_2** Pueden determinar al menos un atributo. ¿Podrán determinar al resto de los atributos?, es decir:

- ¿**C_1**->C_2,C_3,C_4,C_5,C_6 ? Respuesta TRUE
- ¿**C_2**->C_1,C_3,C_4,C_5,C_6 ? Respuesta TRUE

De lo anterior, se concluye que `numero_matricula` o `numero_motor`, pueden actuar como PK. Notar que no se requiere de ambos atributos.

3.3.1.2. Consideraciones requeridas.

- La PK debe ser un campo cuyo valor ***nunca*** debe ser modificado después a que se ha asignado o almacenado en la BD. Modificar PKs implica modificar los valores de todas las referencias en tablas relacionadas lo que puede originar inconsistencias.
- El (los) campo(s) que actúen como PK no deben contener valores nulos.

Para el ejemplo, podríamos descartar al campo ~~`numero_matricula`~~ como PK, ya que, en algunos casos, el auto no cuenta con matrícula (placa) asignada, y esta puede cambiar, por ejemplo, si el auto cambia de dueño.

3.3.1.3. Consideraciones opcionales

- Derivado a que el RDBMS asigna un índice único a los campos que actúen como PK, se requiere procesamiento adicional para calcular y almacenar dicho índice.
- Como se verá más adelante, la creación y uso de índices implican almacenamiento y procesamiento adicional.
- El mejor desempeño se obtiene cuando el índice es construido sobre un atributo cuyo tipo de dato es numérico entero, de preferencia, valores consecutivos no duplicados iniciando en 1.
- Por lo anterior, si se desea obtener este beneficio, la PK de una tabla puede ser formada por un nuevo atributo que cumpla con estas características. A este tipo de llave primaria se le conoce como ***llave primaria artificial***. subrogada

subrogada

Para el ejemplo, si se desea obtener el mejor desempeño, el atributo `numero_motor` también se descarta y en su lugar se creará una llave *primaria artificial* cuyas características se explican a continuación.

nombreTabla_id

3.3.1.4. Tipos de llaves primarias.

Llave primaria natural.

Son representadas por atributos nativos de una entidad. Un atributo nativo es aquel que tiene significado para las reglas de negocio del caso de estudio. Por ejemplo, en la entidad estudiante los atributos *número de cuenta, CURP,* tienen significado para las reglas de negocio y cumplen con la definición de PK, por lo que se les denomina PKs naturales.

Llave primaria artificial o subrogada.

A diferencia de una PK natural, una llave primaria artificial es representada por un campo que se agrega a una tabla que no tiene significado alguno para las reglas de negocio, su única función es actuar como llave primaria que cumple con todos los requisitos anteriores:

- Es un campo numérico (mejora del desempeño al aplicarle un índice tipo unique).
- Sus valores son enteros consecutivos.
- Su valor nunca es modificado una vez asignado.
- Se define como un campo obligatorio.
- Para cada registro que se agrega a la tabla se incrementa su valor, similar a un secuenciador, por lo que cumple con el concepto de dependencia funcional, es decir:

$$llave\_artificial\_id \rightarrow A, B, C, D, E \ldots$$

Por convención una PK artificial se le asigna como nombre: `<nombre_tabla>_id`.

Ejemplo:

Para el caso de la tabla anterior, la PK seleccionada, puede ser una llave primaria artificial llamada `automovil_id`

AUTOMOVIL

automovil_id (PK)	numero_matricula	numero_motor	marca	modelo	año	pais_origen
1	CCA-341	91234908123	Toyota	Yaris	2005	Japón
2	OFG-851	53489787679	Fiat	Fiorino	2008	Tailandia
3	XTV-657	30752312386	Ford	Mustang	2011	México
4	WGB-959	50934187123	Toyota	Avensis	2010	Tailandia

Llave primaria candidata.

Son llaves primarias naturales que no fueron seleccionadas para representar a la PK de la tabla por alguna razón. Por ejemplo, en la tabla anterior, se generó una llave primaria artificial. El campo `numero_motor` también pudo haberse seleccionado, representa a un PK candidata.

Llave primaria compuesta.

Una PK compuesta está formada por más de un atributo. Es importante destacar que el valor de la PK se forma por la combinación de los valores de los N campos que la integren. Para determinar un registro duplicado, se deberá buscar la existencia de la combinación registrada en la tabla.

Ejemplo:

Suponer la siguiente tabla de datos, con PK compuesta la cual contiene las faltas y las calificaciones de los alumnos.

HISTORIAL_ALUMNO

asignatura_id (PK)	alumno_id (PK)	faltas	calificación
1	1000	1	9
2	1000	2	8
3	1001	0	10
4	1001	5	6

- Observar que se marcan a las 2 primeras columnas de la tabla como parte de la PK. Es importante mencionar que solo existe una PK. Ninguna tabla puede tener más de una restricción de llave primaria. En este caso se trata de una sola PK formada por 2 atributos.
- ¿Por qué es necesario una PK compuesta en este ejemplo?
  - Para poder identificar a los valores de las calificaciones y las faltas se requiere no solo del alumno, también se requiere saber la asignatura. Los valores de las columnas `faltas` y `calificación` dependen de las combinaciones únicas de las columnas que forman a la PK.
- ¿Qué pasa al insertar los siguientes valores?
  - 5,1000 OK
  - 1000,5 OK
  - **1,1001 OK**
  - 2,1001 OK
  - 1001,2 OK
  - **1,1001 ¡Error!** La combinación ya existe, PK duplicada. Para que un registro se considere como duplicado, la combinación de los valores de los atributos que forman a la PK tuvieron que haber sido registrados anteriormente.
  - Notar que las combinaciones 2,1001 y 1001,2 ***no*** se consideran como duplicados. Esto se debe a que las columnas que forman a la PK tienen valores diferentes para cada uno de estos registros. Es decir:
    - `asignatura_id = 2, alumno_id =1001` es diferente a
    - `asignatura_id = 1001, alumno_id = 2`

Es importante revisar cuidadosamente las reglas de negocio del caso de estudio a modelar para verificar si realmente una tabla requiere de más de un campo como PK. A nivel general, una tabla requiere de una PK compuesta en los siguientes 2 escenarios:

- Existe al menos un atributo de la tabla cuyos valores requiere de más de un atributo para ser identificado de forma única. En el ejemplo anterior, los campos `faltas` y `calificación` requieren tanto del alumno como de la asignatura para identificarlos de forma única.
- Existe una relación M:N entre 2 entidades A y B. En el siguiente tema se revisa a detalle este concepto. Típicamente, una relación M:N implica la incorporación de una tercera tabla para romper este tipo de relación en 2 relaciones 1:M. La nueva tabla estará formada por una PK compuesta cuyos atributos serán las PKs de las entidades A y B.

Una inadecuada asignación de varios campos como PK puede dar origen a redundancias e inconsistencias en los datos.

Ejemplo:

Suponer que en el ejemplo de los autos se asigna de forma incorrecta a la PK de la siguiente manera:

AUTOMOVIL

numero_matricula (pk)	numero_motor (pk)	marca	modelo	año	pais_origen
CCA-341	91234908123	Toyota	Yaris	2005	Japón
OFG-851	53489787679	Fiat	Fiorino	2008	Tailandia

- Lo anterior implica que sería factible insertar el siguiente registro:

numero_matricula (pk)	numero_motor (pk)	marca	modelo	año	pais_origen
CCA-341	9999555993	Ford	Mustang	2011	México

¡Los registros con matrícula CCA-341 son inconsistentes! La PK compuesta permite registrar autos con características diferentes, inclusive con números de motor diferentes con el mismo número de matrícula o placa.

3.3.2. Restricciones de referencia

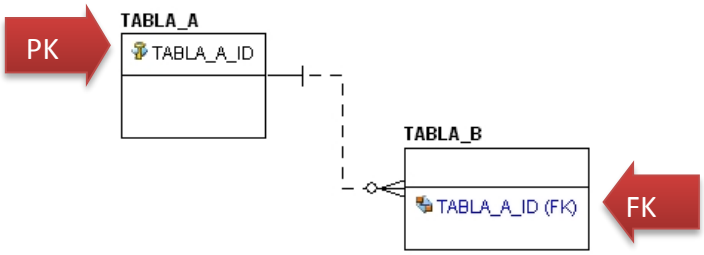
Para poder realizar la implementación de los tipos de relaciones existentes entre 2 entidades, el modelo relacional hace uso del concepto de ***llave foránea (foreign key: FK)***. relación 1:1/1:M/M:N

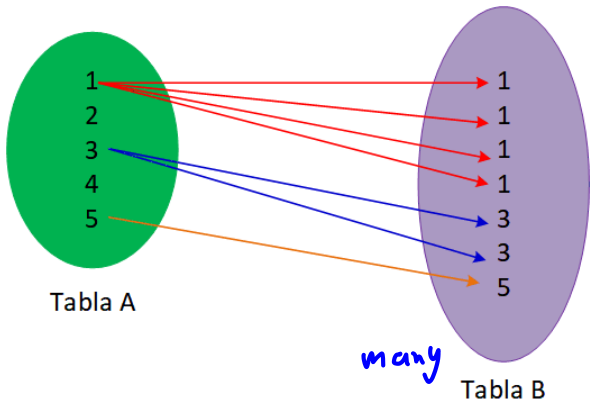
- Una FK es un campo dentro de una tabla B que hace referencia a una tabla A.
- El campo de la tabla A representa a la ***PK de la tabla*** o en su defecto a un atributo que contenga una restricción de tipo UNIQUE.
- Típicamente, a la tabla A se le conoce como tabla "***padre***" (aquella que dona o donde se origina la FK), y a la tabla B como ***tabla hija*** (aquella que recibe el campo).
- Este tipo de restricciones se emplean para relacionar entidades.

Ejemplo:

- una instancia de A se asocia con varias instancias de B, o también:
- En la tabla B, pueden aparecer varios registros que se asocien al mismo registro de A.

Gráficamente:



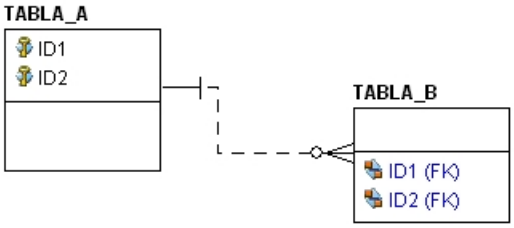


- Generalmente (no requerido) el nombre del atributo correspondiente a la FK coincide con el nombre de la PK de la tabla asociada.

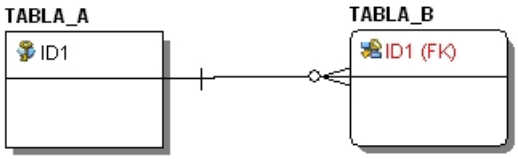
el manejador valida ambos

Otras variantes:

- La PK de la tabla A, es compuesta: Ambos campos se agregan a la tabla B.

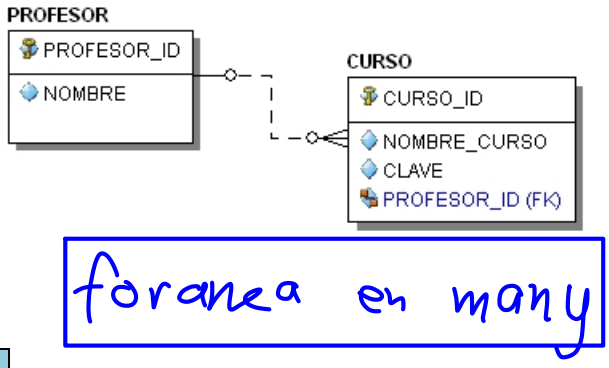


- La llave foránea puede formar parte o ser la PK y FK a la vez de la tabla B.



Existen diversas notaciones y variantes adicionales a las antes mencionadas. Su significado de cada una se revisará a detalle en el tema 4.

Ejemplo:
Un profesor imparte uno o varios cursos.
Un curso es impartido por un profesor.



PROFESOR

profesor_id (PK)	nombre
1	Juan
2	Mario
3	Luisa

CURSO

curso_id (PK)	nombre_curso	profesor_id (FK)
1	Java básico	3
2	C# básico	2
4	Bases de datos	2
5	Economía	3
6	Diseño gráfico	1

- El RDBMS verificará de manera automática que el valor asignado a una FK en la tabla hija exista como PK en la tabla padre. En caso de que no se cumpla esta restricción de referencia, el manejador generará un error (*referential integrity violation*).
- ¿Qué pasa si se intenta eliminar al profesor con id = 3?
- ¿Qué pasa si se agrega al a tabla curso el siguiente registro?

curso_id (PK)	nombre_curso	profesor_id (FK)
7	Diseño gráfico	5

3.3.3. Restricciones de integridad.

- Ayudan a mantener el estado correcto o verídico de los datos (consistencia e integridad).
- Son verificadas por el RDBMS de forma automática.

Tipos de restricciones de integridad:

- NOT NULL
- CHECK
- UNIQUE
- Verificación empleando Triggers.

3.3.3.1. NOT NULL

Esta restricción le indica al RDBMS que el valor de un campo es requerido, por lo que, si se intenta insertar un nuevo registro con valores "nulos" a un campo definido como NOT NULL, el manejador generará un error de "violación de restricción".

¿Qué es un valor nulo?

- NULL se define como la ausencia de valor (dato) para un determinado registro y columna.
- A nivel de negocio, si el valor de un atributo de una entidad es opcional, el campo se debe definir como NULL.
- Importante, NULL no es equivalente a escribir 0, ""(cadena vacia), " " (espacio), "null".
- Por lo anterior, es inválido hacer comparaciones empleando los valores anteriores. Para ello se deben emplear los operadores `is null`, `is not null`.

Ejemplo:

La siguiente sentencia obtiene los datos de los empleados que no registraron su RFC, es decir, el campo no tiene asignado algún valor:

```
select *
from empleado
where rfc is null;
```

3.3.3.2. CHECK

Check es una restricción que hace uso de una expresión booleana para validar que el valor a asignar a un campo sea correcto.

Ejemplo: propia columna

```
check (sueldo_mensual>=5000 and sueldo_mensual<=999999)
```

- La instrucción anterior, hará que el RDBMS, verifique que el sueldo mensual de un trabajador, por ejemplo, este en el rango [5000,999999]
- Esta restricción se especifica al momento de definir el campo al momento de crear una tabla. (Esto se revisará en la última parte del curso: SQL).

3.3.3.3. UNIQUE

Esta restricción le indica al RDBMS que los valores que puede tener un campo deben ser únicos. Es decir, no deben existir 2 registros con el mismo valor para las columnas que tengan esta restricción.

- Para implementar esta restricción, el RDBMS creará un ***índice de tipo unique*** y se lo aplicará al campo en cuestión.
- ¿Qué elemento del modelo relacional tiene esta característica? R: Las llaves primarias.
- Para el caso de las PKs, por default, el manejador siempre agregará un índice de tipo ***unique***.
- Para campos que no sean parte de la PK, si se desea que sus valores sean únicos, se debe crear el índice de manera explícita, y asociarlo al campo. Los índices se revisarán en la siguiente sección.

3.3.3.4. Verificación empleando Triggers

- Como su nombre lo indica, este tipo de restricciones hacen uso de un Trigger para validar el complimiento de alguna condición que permita la inserción o modificación de un registro.
- Un trigger es un programa (script SQL) que se ejecuta cuando ocurre algún evento, por ejemplo: antes o después de insertar un registro, al modificar, o al eliminar, etc.
- La diferencia con CHECK es que un trigger puede realizar cualquier tipo de validación y/o procedimiento más complejo. Algunos ejemplos:
  - La validación implica leer datos de otros registros diferentes al registro que se está validando
  - La validación requiere consultar datos de otras tablas.
  - La validación requiere de cierta lógica que implica programación, por ejemplo, uso de sentencias de control `if, else`, o ciclos.
- Los triggers se estudiarán en la parte final del curso (SQL).

3.4. Reglas de CODD.

En 1985 Edgar Frank Codd publicó 12 reglas que en su conjunto definen las características que debe cumplir un sistema de bases de datos para que este sea considerado como relacional.

Regla 1: Información

Toda información contenida en una base de datos relacional debe ser representada de manera lógica como los valores de una columna contenidos en los registros de una tabla.

Regla 2: Acceso garantizado

Cada valor en una tabla está garantizado a ser accesible a través de la combinación nombre *de la tabla, nombre de la llave primaria y nombre de la columna.*

Regla 3: Tratamiento sistemático de valores nulos

Valores nulos deben ser tratados de manera sistemática independiente del tipo de dato.

Regla 4: Existencia de un catálogo dinámico basado en el modelo relacional.

La información del modelo relacional (metadatos) debe ser almacenada en tablas y administrada como cualquier dato ordinario contenido en la base de datos. Esta información debe estar disponible a usuarios y serán accedidos de la misma forma que los datos ordinarios, empleando un lenguaje relacional estándar.

Regla 5: Soporte de varios lenguajes de datos.

La base de datos relacional puede tener soporte para varios lenguajes, sin embargo, esta debe contar con el soporte para un lenguaje bien definido, lenguaje declarativo, que tenga soporte para definición

de datos, definición de vistas, manipulación de datos, constraints de integridad, autorización, y administración de transacciones (commit, rollback, begin).

Regla 6: Actualización de vistas.
Cualquier vista que teóricamente es actualizable debe hacerse a través del sistema.

Regla 7: Insert, update y delete de alto nivel.
La base de datos debe contar con un soporte para realizar operaciones de insert, update, y delete con un lenguaje de alto nivel (al igual que select), fácilmente.

Regla 8: Independencia física de los datos.
Aplicaciones y herramientas que interactúan con la base de datos no deben sufrir cambios si se altera la estructura y los métodos de acceso a datos.

Regla 9: Independencia lógica de los datos.
Aplicaciones y herramientas no deben ser afectadas al realizar cambios a tablas y su estructura mientras sigan conservando los datos antes de dicha modificación. Ejemplo:

- Cambiar el orden de columnas
- Insertar una columna.

Regla 10: Independencia de la integridad.
Las restricciones de integridad deben ser definidas a través del lenguaje relacional y almacenadas en el sistema (Diccionario de datos), y no deben estar definidas del lado de la aplicación.

Regla 11: Independencia de la distribución.
Tanto usuarios como programas no deben tomar en cuenta o resultar afectadas si se cambia la ubicación o si se realiza una distribución física de los datos (Bases de datos distribuidas vs locales)

Regla 12: No subversión.
Si el sistema tiene lenguajes de bajo nivel, estos lenguajes de ninguna manera pueden ser usados para violar la integridad de las reglas y restricciones expresadas en un lenguaje de alto nivel (SQL).

Regla 0: Regla cero.
Para que un sistema de bases de datos relacional se le denomine como tal, deberá usar (exclusivamente) sus capacidades relacionales para gestionar la base de datos.

3.5. ÍNDICES

Como se observó en la sección anterior, el uso de índices se emplea para implementar algunas restricciones que ofrece el modelo relacional. Por lo anterior, resulta importante revisar su funcionamiento, utilidad y beneficios que estos ofrecen.

- Un índice es una estructura de datos. Requiere almacenamiento al igual que una tabla.
- El principal objetivo de un índice es el incrementar el desempeño de la base de datos principalmente para realizar la extracción o explotación de los datos en el menor tiempo posible.
- Esto se logra ***minimizando*** el número de lecturas que el RDBMS tendría que realizar a cada uno de los archivos de datos asociados a una tabla.

Un índice se emplea principalmente para:

- Localizar datos sin tener que recorrer todo el contenido de una tabla.
- Para realizar el ordenamiento de datos de uno o más campos
- A través del ordenamiento, para garantizar la no duplicidad de los valores de un campo empleados en especial en la PK.

3.5.1. Elementos de un índice.

El concepto de un índice es muy similar al de un libro, está formado por 2 elementos:

- Una llave (key)
- Un valor. raw_id

Analogía con el índice de un libro:

- Llave → Representa el título del tema o palabra que estamos buscando
- Valor → Número de página donde se encuentra la información solicitada.

Índice en una Base de datos:

- Llave → El valor del dato que se quiere buscar. Por ejemplo, obtener todos los registros con `email = juan@m.com`.
- Valor → Un identificador único que indica la ubicación en disco donde se encuentra el registro solicitado. En Oracle se le conoce como ***row_id***.

3.5.1.1. Row_ids en Oracle.

- Oracle utiliza una codificación en base 64 para representar las direcciones físicas de cada registro haciendo uso de una cadena de 18 caracteres.
- El DBMS agrega una columna a cada tabla llamada ROW_ID en la que se almacena esta cadena.
- Los ROW_ID están compuestos de cuatro partes:
  - **OOOOOO**: Data Object Number o ID que identifica el segmento de la base de datos.
  - **FFF**: El número de data file que contiene los datos del registro relativo a un tablespace.
  - **BBBBBB**: El bloque de datos que contiene al registro. Los números de bloque son relativos al data file al que pertenecen, no al tablespace. Dos registros con el mismo número de bloque pueden residir en dos data files diferentes del mismo tablespace.
  - **RRR**: El número de registro en el bloque.

- Un ROW_ID representa la estrategia más eficiente para localizar un registro en la base de datos.

Ejemplo:

- La siguiente tabla muestra los datos de los clientes de una empresa.

CLIENTE

CLIENTE_ID	NOMBRE	AP_PATERNO	AP_MATERNO	ENTIDAD	SEXO	CODIGO_POSTAL	EMAIL
1000	JUAN	LUNA	JUAREZ	ZAC	M	32948	juan@m.com
1002	MARIO	MARTINEZ	AGUILAR	AGS	M	00293	mario@h.com
1003	ALEJANDRO	PEREZ	MORALES	GRO	M	02934	ale@hd.com
1004	HUGO	LINARES	HURTADO	DF	M	20398	hg@aol.com
1005	TANIA	SANCHEZ	GUTIERREZ	GRO	F	98909	ts@em.com
1006	ALONSO	LUGO	OLVERA	AGS	M	02934	alo@yo.com
1007	MIRIAM	MONDRAGON	RUBIO	YUC	F	02934	lug@tr.com
1008	JULIETA	ZAVALA	YAÑEZ	CHIH	F	02342	miri@msn.com
1009	JUAN	BENITEZ	LOPEZ	NL	M	02349	beno@su.com
1010	MARIA	JIMENEZ	BALDERRAMA	GRO	F	03453	mja@aol.com

La siguiente tabla muestra todos los datos de la tabla incluyendo el ROW_ID

ROWIDs de la tabla CLIENTE

ROW_ID	CLIENTE_ID	NOMBRE	AP_PATERNO	AP_MATERNO	ENTIDAD	SEXO	CODIGO_POSTAL	EMAIL
AAASSdAAEAAAASdAAA	1000	JUAN	LUNA	JUAREZ	ZAC	M	32948	juan@m.com
AAASSdAAEAAAASdAAB	1002	MARIO	MARTINEZ	AGUILAR	AGS	M	00293	mario@h.com
AAASSdAAEAAAASdAAC	1003	ALEJANDRO	PEREZ	MORALES	GRO	M	02934	ale@hd.com
AAASSdAAEAAAASdAAD	1004	HUGO	LINARES	HURTADO	DF	M	20398	hg@aol.com
AAASSdAAEAAAASdAAE	1005	TANIA	SANCHEZ	GUTIERREZ	GRO	F	98909	ts@em.com
AAASSdAAEAAAASdAAF	1006	ALONSO	LUGO	OLVERA	AGS	M	02934	alo@yo.com
AAASSdAAEAAAASdAAG	1007	MIRIAM	MONDRAGON	RUBIO	YUC	F	02934	lug@tr.com
AAASSdAAEAAAASdAAH	1008	JULIETA	ZAVALA	YAÑEZ	CHIH	F	02342	miri@msn.com
AAASSdAAEAAAASdAAI	1009	JUAN	BENITEZ	LOPEZ	NL	M	02349	beno@su.com
AAASSdAAEAAAASdAAJ	1010	MARIA	JIMENEZ	BALDERRAMA	GRO	F	03453	mja@aol.com

3.5.2. Construcción de un índice.

Suponer que el DBA agrega un índice al campo `email`. Responder las siguientes preguntas:

¿Qué información contiene el índice generado?

Para construir el índice sobre dicho campo, el elemento ***key*** del índice corresponde con los valores del campo `email` y el elemento ***valor*** corresponde al `row_id` del registro:

key (campo email)	valor(row_id)
juan@m.com	AAASSdAAEAAAASdAAA
mario@h.com	AAASSdAAEAAAASdAAB
. . .	
mja@aol.com	AAASSdAAEAAAASdAAJ

¿Qué sucede cuando se inserta, modifica o elimina un nuevo registro que contiene columnas indexadas?

- El manejador actualiza el índice asociado con el campo `EMAIL`.
- Si la tabla contiene un número considerable de índices, estas actualizaciones pueden representar un problema de **desempeño**. Se debe tener cuidado al seleccionar las columnas a indexar.

¿Qué acciones realiza el manejador al ejecutar la siguiente sentencia?

```
select *
from cliente
where email ='juan@m.com'
```

- El manejador realiza un escaneo del índice el cual puede ser **:**
  - Escaneo completo (full index scan)
  - Escaneo parcial (partial index scan)
  - Escaneo único (unique index scan)
- Durante este escaneo, localiza el valor del elemento KEY y obtiene los ROW_ID de donde KEY = 'juan@m.com'

key (campo email)	valor(row_id)
juan@m.com	AAASSdAAEAAAASdAAA

- La estrategia para encontrar estos registros en el índice depende de su tipo.
- Con la lista de ROW_IDs obtenidos, realiza un acceso directo a los datos de la tabla `cliente` para obtener los datos del registro completo.

- *¿Cuándo se debe emplear un índice? , ¿Indexar todos los campos?*

Los siguientes puntos representan recomendaciones generales para decidir si a un campo se le agrega o no un índice.

- Existe la necesidad de consultar datos con condiciones que ocurren con mucha frecuencia. Por ejemplo, se realizan consultas muy frecuentes empleando el campo `email`.
- La diversidad de valores de un campo es alta. En este caso:
  - el campo `sexo`, no tendría sentido indexarlo, solo tiene 2 valores M y F.
  - El campo `email` es correcto ya que es poco probable que su valor se repita.
  - ¿Qué otros campos de la tabla son candidatos para indexar?
- El número de registros esperado como respuesta debe ser pequeño: entre el 2% y 4% del número total de registros de la tabla.
- El número de registros de la tabla es grande, tablas con pocos registros cuyo contenido se puede leer con 1 o 2 accesos a disco, se prefiere hacer un escaneo completo de la tabla (FULL TABLE SCAN)
- Tener en cuenta que, a mayor número de índices, aumenta el tiempo requerido para hacer una inserción o modificación de registros.

3.5.3. Tipos de Índices:

- HASH index
- B+ tree index (Balanced Trees)
- Bitmap index.

3.5.3.1. Índices tipo HASH (HASH index)

Hash: Es una clave que representa prácticamente de manera única a un registro, a un documento, a otra cadena, etc.

***Función hash*:** Encargada de generar valores hash. Sus características son:

- Recibe un dato de entrada y genera un valor hash.
- Este valor hash debe ser prácticamente único para cada entrada.
- Pueden existir valores hash idénticos para 2 entradas diferentes (Colisión). Entre mejor sea su algoritmo, menor probabilidad de colisión.

Ejemplo:

Cálculo de firma digital. Considere un texto cualquiera, empleando los códigos ASCII de cada carácter, es posible generar una función hash que represente de forma única a un texto:

Hash table

E	n		u	n		r	i	n	c	ó	n		d	e	
69	110	32	117	110	32	114	105	110	99	243	110	32	100	101	
		-1312			224			990			-15840			-6868	-22806

	l	a		M	a	n	c	h	a		d	e		c	
32	108	97	32	77	97	110	99	104	97	32	100	101	32	99	
		-7372			-4365			1144			6500			6831	2738

u	y	o		n	o	m	b	r	e		n	o		q	
117	121	111	32	110	111	109	98	114	101	32	110	111	32	113	
		-444			-8658			1254			7590			8927	8669
															-11399

Para cada 3 caracteres, aplicamos la siguiente operación: $(C_1 - C_2) * C_3$

Y finalmente el resultado de la función hash es la suma de cada renglón

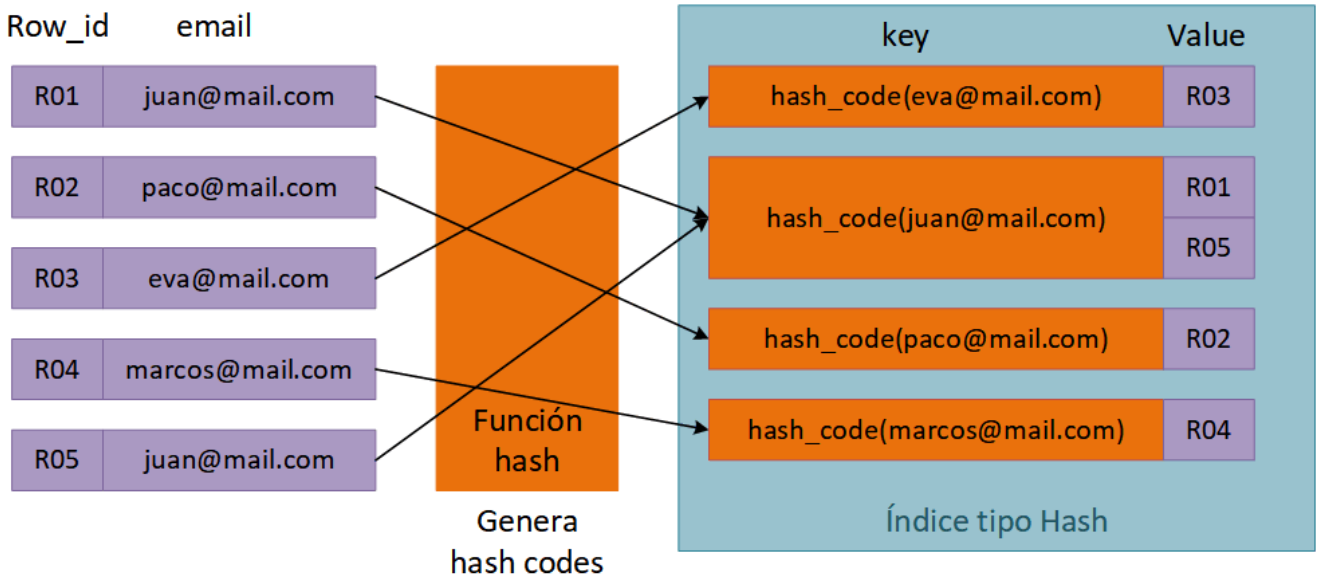
$$hash = \sum_{i=0}^{n} (C_{i1} - C_{i2}) * C_{i3}$$

Este valor numérico nos representa al texto anterior. Cualquier cambio en un carácter, cambiará el resultado.

Usos principales:

- Firmas digitales
- Criptografía
- Búsquedas.

En base de datos, los códigos hash se emplean para determinar la ubicación de los datos de una tabla a partir del valor de la llave (key) del índice. Este proceso se realiza a través del uso de un ***hash table***.



- Para cada llave (key), se calcula su código hash empleando una función hash (fx).
- Una tabla hash contiene un conjunto de parejas formadas por `{hash_code(key), value}`.
  - Key: valores de la columna indexada
  - Value: Row Id del registro

- El hash code generado se emplea para distribuir y ubicar a las parejas dentro del hash table.

Algoritmo de búsqueda.

Suponer que el manejador decide ejecutar la siguiente sentencia:

```
select * from cliente where email = 'juan@mail.com'
```

- Debido a que la columna email ha sido indexada con un índice tipo hash, el manejador hará uso de esta columna aplicando el siguiente procedimiento:

1. Calcula el código hash para `juan@mail.com`: `hash_code = fx('juan@mail.con')`
2. Con este valor se realiza una búsqueda en el hash table para encontrar a todos los valores cuyo hash code corresponda con el email `juan@mail.com`. Es decir:

   ```
   buscaEnHT(hash_code('juan@mail.com'));
   ```

3. El resultado de la búsqueda obtendrá una lista de valores que en este caso corresponde con una lista de row_ids. Para el ejemplo, notar que existen 2 row_ids asociados al mismo email. Por lo tanto, la búsqueda regresaría los row_ids `{R01,R05}`.
4. Una vez que el manejador obtiene los row_ids como resultado de hacer uso del índice, este hará acceso a la base de datos para recuperar solo esos 2 registros de forma eficiente, sin tener que leer o acceder a todo el contenido de la tabla.

- Recordar: Cuando se usa un índice independiente de su tipo, el manejador ***siempre*** obtendrá una lista de row_ids los cuales le servirán para acceder a los datos de forma eficiente.

A nivel general, este tipo de índices son muy eficientes para realizar búsquedas, sin embargo, cuentan con las siguientes desventajas , lo que hace que este tipo de índice no es muy utilizado en la práctica.

- No soportan ordenamiento de datos
- No soportan consultas con operadores lógicos como >=, >, <=, <. Las búsquedas en un hash table siempre hacen uso del operador de igualdad.
- Para tablas grandes se requiere mayor cantidad de memoria para construir el hash table.
- No soporta valores nulos. Columnas indexadas con valores nulos no son incluidas en este tipo de índice. Esto implica que si se intenta ejecutar la siguiente sentencia que justamente realice la búsqueda de valores nulos, el manejador no hará uso del índice debido a que el hash table no puede contener keys nulas. El manejador tendrá que recorrer toda la tabla y encontrar los valores que son nulos.

  ```
  select * from cliente where email is null
  ```

3.5.3.2. Índices Bit Map

- Este tipo de índice, emplea un arreglo de bits (0s y 1s) para representar la existencia de un valor o condición.
- Se emplea más en data warehouses en tablas grandes donde los valores de las columnas se repiten mucho (esparcimiento bajo).
- Las tablas tienen una gran cantidad de datos. Emplean muy poco espacio de almacenamiento .

A	B	C	D	E	F	G	H	**NULL**	I	J
Bit1	Bit2	…	…							Bit N
0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0

- Cada columna de la matriz, representa a cada uno de los posibles valores que puede tener la columna indexada. En este caso A, B, C.. son los posibles valores de una columna que contiene un índice bit map.
- Si el valor de una columna es NULL, se considera como un valor diferente. Esto significa que un índice bitMap soporta valores nulos, a diferencia de un índice hash.
- Por cada registro de la tabla, se crea un nuevo registro en la matriz.
- Para determinar el valor del bit, se realiza el siguiente procesamiento (para cada registro):
    - Si el valor de la columna indexada coincide con alguna de las columnas de la matriz, su valor será 1. De lo contrario, su valor será 0.

Ejemplo:

Retomando la tabla CLIENTE:

ROW_ID	CLIENTE_ID	NOMBRE	AP_PATERNO	AP_MATERNO	ENTIDAD	SEXO	CODIGO_POSTAL	EMAIL
AAASSdAAEAAAASdAAA	1000	JUAN	LUNA	JUAREZ	ZAC	M	32948	juan@m.com
AAASSdAAEAAAASdAAB	1002	MARIO	MARTINEZ	AGUILAR	AGS	M	00293	mario@h.com
AAASSdAAEAAAASdAAC	1003	ALEJANDRO	PEREZ	MORALES	GRO	M	02934	ale@hd.com
AAASSdAAEAAAASdAAD	1004	HUGO	LINARES	HURTADO	DF	M	20398	hg@aol.com
AAASSdAAEAAAASdAAE	1005	TANIA	SANCHEZ	GUTIERREZ	GRO	F	98909	ts@em.com
AAASSdAAEAAAASdAAF	1006	ALONSO	LUGO	OLVERA	AGS	M	02934	alo@yo.com
AAASSdAAEAAAASdAAG	1007	MIRIAM	MONDRAGON	RUBIO	YUC	F	02934	lug@tr.com
AAASSdAAEAAAASdAAH	1008	JULIETA	ZAVALA	YAÑEZ	CHIH	F	02342	miri@msn.com
AAASSdAAEAAAASdAAI	1009	JUAN	BENITEZ	LOPEZ	NL	M	02349	beno@su.com
AAASSdAAEAAAASdAAJ	1010	MARIA	JIMENEZ	BALDERRAMA	GRO	F	03453	mja@aol.com
AAASSdAAEAAAASdAAK	1011	PEDRO	LUNA	JIMENEZ	NULL	M	02343	pe@mail.com

El índice bit Map del campo ENTIDAD será el siguiente:

ROW_ID	AGS	BCN	CHIH	DF	GRO	NL	YUC	ZAC	NULL
AAASSdAAEAAAASdAAA	0	0	0	0	0	0	0	**1**	0
AAASSdAAEAAAASdAAB	**1**	0	0	0	0	0	0	0	0
AAASSdAAEAAAASdAAC	0	0	0	0	**1**	0	0	0	0
AAASSdAAEAAAASdAAD	0	0	0	**1**	0	0	0	0	0
AAASSdAAEAAAASdAAE	0	0	0	0	**1**	0	0	0	0
AAASSdAAEAAAASdAAF	**1**	0	0	0	0	0	0	0	0
AAASSdAAEAAAASdAAG	0	0	0	0	0	0	**1**	0	0
AAASSdAAEAAAASdAAH	0	0	**1**	0	0	0	0	0	0
AAASSdAAEAAAASdAAI	0	0	0	0	0	**1**	0	0	0
AAASSdAAEAAAASdAAJ	0	0	0	0	**1**	0	0	0	0
AAASSdAAEAAAASdAAK	0	0	0	0	0	0	0	0	**1**

Observar que por renglón solo se puede prender un bit. El bit encendido corresponde a la columna ZAC ya que el valor de la columna indexada para este primer registro tiene justamente el valor ZAC.

Algoritmo de búsqueda.

Suponer que el manejador decide realizar la siguiente consulta:

```
select * from cliente where entidad = 'GRO';
```

1. El manejador determina la existencia de un índice BitMap que coincide con la columna empleada en el predicado de la búsqueda por lo que decide emplearlo.
2. El manejador ubica a la columna con nombre 'GRO'.
3. Realiza un escaneo vertical sobre dicha columna. Si el bit es 1, recuperará el valor del row_id del registro en turno.
4. Al final del recorrido se obtendrá de forma similar a otros tipos de índice una lista de row_ids.
5. El manejador accede de forma eficiente a los registros solicitados empelando la lista de row_ids obtenida en el punto anterior.

ROW_ID	GRO
AAASSdAAEAAAASdAAA	0
AAASSdAAEAAAASdAAB	0
AAASSdAAEAAAASdAAC	**1**
AAASSdAAEAAAASdAAD	0
AAASSdAAEAAAASdAAE	**1**
AAASSdAAEAAAASdAAF	0
AAASSdAAEAAAASdAAG	0
AAASSdAAEAAAASdAAH	0
AAASSdAAEAAAASdAAI	0
AAASSdAAEAAAASdAAJ	**1**
AAASSdAAEAAAASdAAK	0

Lista de row_ids: { AAASSdAAEAAAASdAAC , AAASSdAAEAAAASdAAJ }

Ventajas de los índices BitMap.

- Requieren de poco espacio de almacenamiento y uso de memoria
- Soportan valores nulos
- Útil a pesar de contar con columnas con muy poca variedad de valores.

Desventajas.

- No soportan ordenamiento. Este último punto es el que evita que el índice no sea el más popular empleado en la práctica.

3.5.3.1. Índices árboles B+ (B = Balanced)

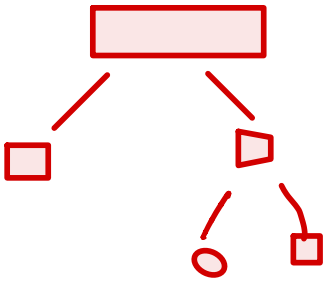
Características de un árbol B+

- Representa a una colección de datos ordenados organizados en forma de árbol.
- Permite inserción y eliminación de elementos de forma eficiente.
- El crecimiento de un árbol B+ es principalmente hacia a los lados evitando el crecimiento de su profundidad. Esta característica permite reducir el número de lecturas I/O que se tienen que realizar para recuperar un dato.
- Cada nodo almacena los valores de la columna a indexar, y tiene una capacidad finita. Típicamente, la capacidad de cada nodo coincide con la capacidad de un bloque de disco (una sola lectura para leer el contenido del nodo).
- Todos los valores de las KEYs (valores de la columna indexada) se encuentran contenidas en las hojas del árbol, los nodos intermedios solo contienen nodos y punteros

- Al tener todos los valores en las hojas, significa que la longitud de recorrido desde la raíz para llegar a un valor es el mismo sin importar el valor de la KEY. De aquí que al árbol se le conozca como balanceado.

Ejemplo:

Construir un árbol B+ para indexar el campo `nombre` de la siguiente tabla considerando que en cada página de disco se pueden almacenar hasta 4 nombres.

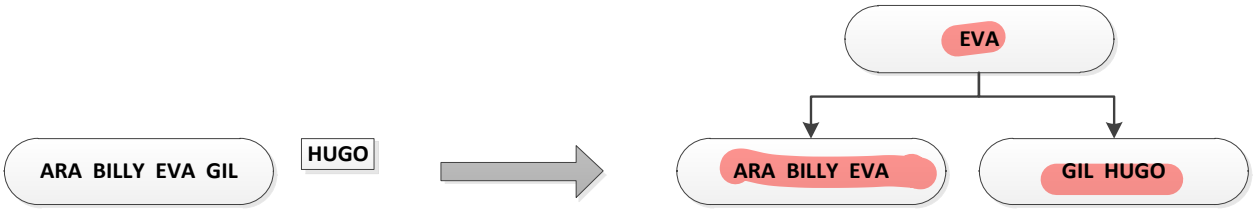


ROW_ID	CLIENTE_ID	NOMBRE
1	1	JUAN
2	2	EVA
3	3	SARA
4	4	LILI
5	5	HUGO
6	6	JULIO
7	7	IVAN
8	8	IRMA
9	9	ARA
10	10	MILA
11	11	YURI
12	12	LALO
13	13	BILLY
14	14	ZULY
15	20	OMAR
16	21	GIL
17	22	LUZ
18	25	MARA

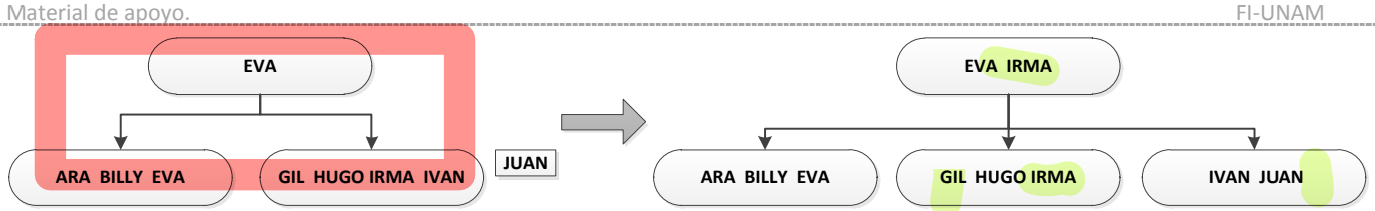
- Paso 1, ordenando los datos:

`D={ARA,BILLY,EVA,GIL,HUGO,IRMA,IVAN,JUAN,JULIO,LALO,LILI,LUZ,MARA,MILA,OMAR,SARA,YURI,ZULY }`

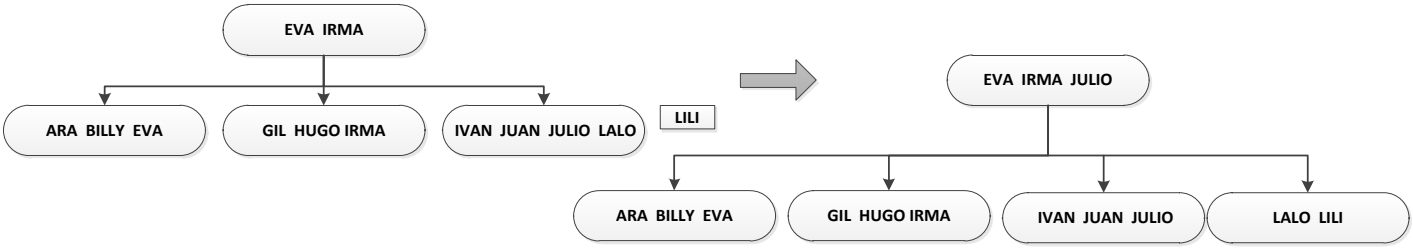
- Paso 2, Construir nodos, considerando longitud máxima = 4. Para agregar al quinto elemento, se parte el nodo existente en 2, y se copia el elemento del centro (considerando al nuevo elemento) a un nuevo nodo padre. En este caso, en nodo con el nuevo elemento quedaría: {ARA,BILLY,**EVA**,GIL,HUGO}. El elemento del centro es **EVA**, por lo tanto, este valor debe incluirse en el nuevo nodo y en el nodo izquierdo.
- Los valores que le siguen al valor **EVA (**en este caso, GIL), se incluyen en el nodo derecho.
- Se agrega el nuevo valor en el nodo derecho.
- La inserción de nuevos valores continua hasta que nuevamente se exceda la capacidad, se repite el procedimiento para las siguientes entradas.



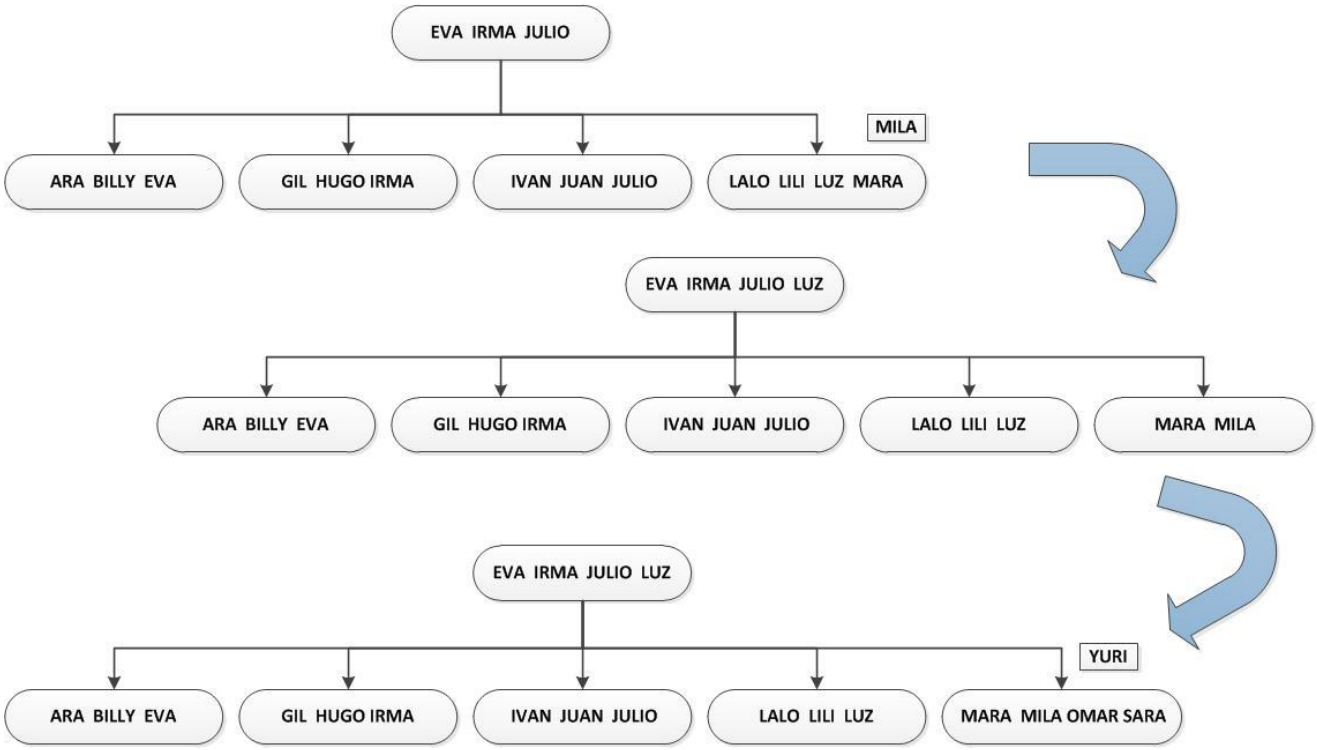
- Observar, el último valor del nodo izquierdo corresponde con el último valor del nodo padre. El proceso se repite para los demás valores:



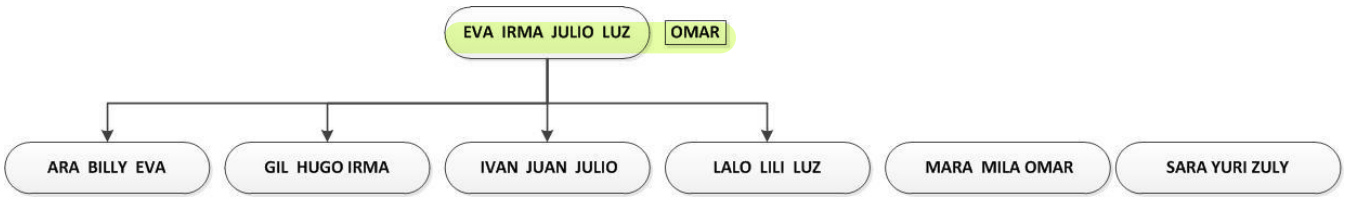
- Observar, el nodo raíz, el valor `EVA` corresponde al valor máximo del nodo hijo izquierdo.
- El valor `IRMA` corresponde al valor máximo del nodo hijo del centro.
- Todos los valores contenidos en el nodo derecho son mayores al valor `IRMA`.
- Estas 3 reglas deben cumplirse para poder realizar una correcta navegación del árbol.



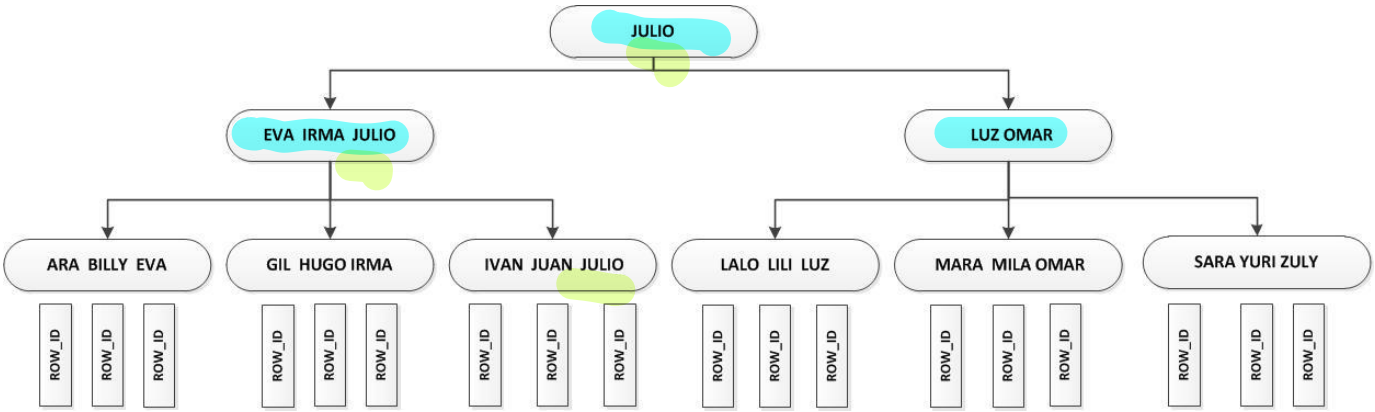
- El proceso se repite para los demás valores.



- ¿Qué pasa al insertar el valor YURI?, el nodo padre ya no tiene capacidad. Lo anterior significa que es momento de que el árbol crezca en forma vertical.

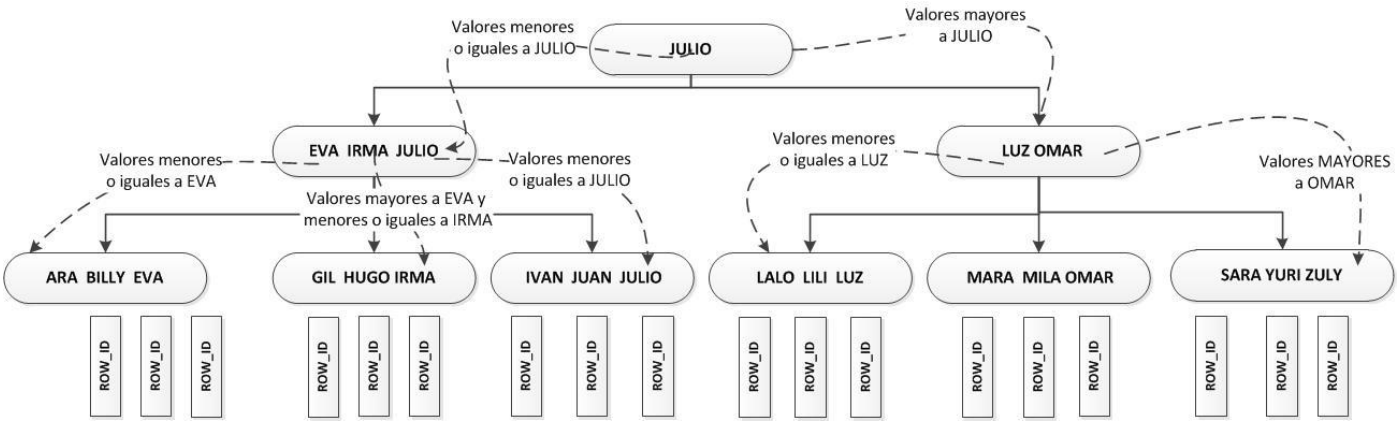


- Como se puede observar, al insertar al valor YURI, provoca la separación del nodo {MARIA,MILA,OMAR,SARA}. El valor OMAR se encuentra al centro, por lo que debe ser incluido en el nodo padre (nodo raíz).
- Sin embargo, el nodo raíz ya no tiene capacidad, por lo que se le aplica el mismo procedimiento: dividir al nodo raíz en 2 y generar un nodo que corresponde con la nueva raíz del árbol.



- Observar que al dividir el nodo raíz, el valor **JULIO** se encuentra al centro, por lo que dicho valor formará parte del nuevo nodo raíz.
- Se realiza un reajuste de los nodos hijos para respetar la regla de ordenamiento y navegación mencionada anteriormente. Es decir:
  - Del lado izquierdo del nodo raíz están todos los elementos menores o iguales al valor JULIO.
  - Del lado derecho están todos los valores mayores al valor JULIO.

- Finalmente, observar que las hojas del árbol corresponden con todos los valores ordenados
- A cada valor de las hojas se le asocia el ROW_ID para realizar el acceso directo a los datos. Es decir, las hojas contienen entradas {KEY,ROW_ID}
- Cada entrada contenida en los nodos intermedios contienen punteros hacia los nodos inferiores, es decir: {KEY,P->NODO HIJO}. Estos punteros son muy importantes ya que indican el camino que el DBMS debe seguir para localizar el ROW_ID de la KEY.



- Observar en el diagrama anterior, cada valor de los nodos intermedios contiene un puntero a uno de los nodos hijos. En todos los casos, el nodo referenciado contiene valores menores o igual al valor de cada elemento del nodo.

3.5.3.2. Búsqueda de datos en un árbol B+

Ejemplo:

Suponer que se desea obtener todos los registros de los clientes que se llamen **HUGO**

1. Se inicia en el nodo raíz.
2. Se compara 'HUGO' con cada KEY de las parejas {KEY, puntero} del nodo.
3. Se busca la pareja {KEY,puntero} que llevará a la hoja del árbol que contenta la pareja {HUGO,ROW_ID}
   ¿Cuál será la pareja {KEY, puntero} que se debe seleccionar?
4. Se debe seleccionar la primer pareja donde se cumpla la expresión: 'HUGO' <= {KEY,puntero}.
   a. Lo anterior se debe a que cada pareja {KEY,puntero} conduce a todas las KEYs que son menores o iguales a ella.
5. Si se termina de comparar todas las KEYs del nodo todas son mayores a 'HUGO', se elige el nodo hijo más a la derecha.
   a. Lo anterior se debe a que el nodo hijo contiene todas las KEYs mayores a la última KEY del nodo padre.
6. Para el ejemplo, HUGO es menor a JULIO, por lo que seguimos el camino indicado por el puntero del valor JULIO, que en este caso es en nodo hijo izquierdo: {EVA,IRMA,JULIO}.
7. Si el valor a buscar fuera mayor al mayor de las KEYs contenidas en el árbol, se emplea el nodo hijo derecho ya que por las características del árbol, los valores están ordenados de izquierda a derecha en forma ascendente.
8. En el nuevo nodo se vuelve a realizar la comparación. En este caso, se compara:
   HUGO<=EVA => FALSE
   HUGO <=IRMA => TRUE, se selecciona este nodo cuyo puntero lleva a {GIL,HUGO,IRMA}.
9. Este nuevo nodo, es hoja del árbol, por lo tanto, el valor se debe encontrar en este nodo. Se lee el bloque de este nodo, y se obtiene el ROW_ID de la KEY HUGO.

Conclusión: Se realizaron 3 lecturas para encontrar el ROW_ID.

Ejemplo:

Suponer que se desea obtener todos los registros de los clientes que se llamen **LILI**

1. Se inicia el nodo raíz.
2. En este caso no existe un nodo mayor a LILI por lo que se emplea el mayor del nodo. Debido a que el valor es mayor, se emplea el nodo derecho {LUZ,OMAR}.
3. En este nuevo nodo, LUZ es el nodo mínimo que es mayor a el valor LILI, por lo tanto seguimos el puntero de LUZ, lo que nos lleva al nodo hoja {LALO,LILI,LUZ}. De este último se obtiene el ROW_ID

Conclusión: Se realizaron 3 lecturas I/O para encontrar el ROW_ID.

Ejemplo:

Suponer que se desea obtener todos los registros de los clientes que se llamen **SARA**

1. Se inicia en el nodo raíz.
2. No existe KEY mayor a SARA, por lo que empleamos la mayor del nodo: JULIO. SARA es mayor a JULIO, por lo tanto, empleamos el nodo derecho {LUZ, OMAR}.
3. En este nodo, nuevamente, no hay llave mínima mayor al valor SARA, el valor SARA es mayor al valor máximo OMAR, por lo tanto empleamos el nodo hijo del lado derecho {SARA, YURI, ZULY}. De este último se obtiene el ROW_ID

Conclusión: Se realizaron 3 lecturas I/O para encontrar el ROW_ID.

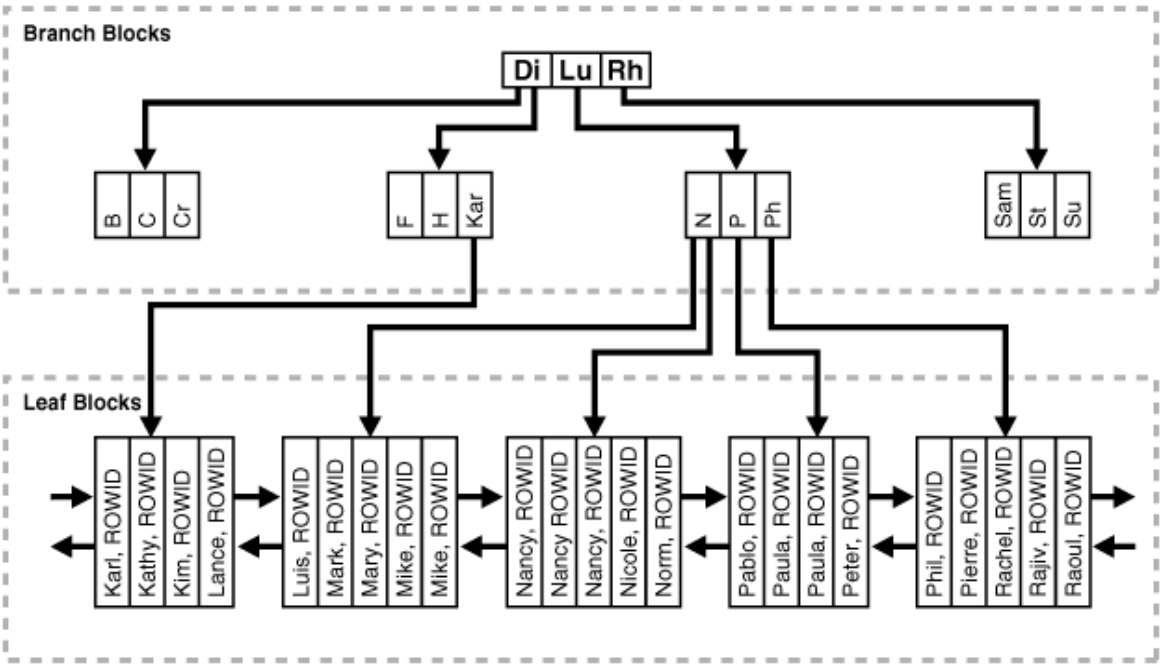
Los árboles B+ permiten conocer el camino a recorrer para obtener un ROW_ID sin posibilidad a equivocarse de ruta. El número máximo de lecturas I/O corresponde con la profundidad del árbol, en este caso 3.

Preguntas tipo examen, discutirlas en clase.

1. ¿Qué sucede con el árbol si se inserta un nuevo registro con el valor PACO?
2. ¿Qué sucede con el árbol si se inserta un nuevo registro con el valor MARA? Considerar que ya existe una KEY con dicho valor.
3. ¿Qué sucede con el árbol si se inserta un nuevo registro y el valor del campo indexado es nulo?
4. ¿Por qué razón todos los nodos hoja tiene 3 elementos?
5. Considerando el árbol resultante de la lámina anterior, ¿cuántas KEYs más se pueden insertar en el árbol sin que este aumente de profundidad, es decir, antes que el árbol tenga profundidad =4.

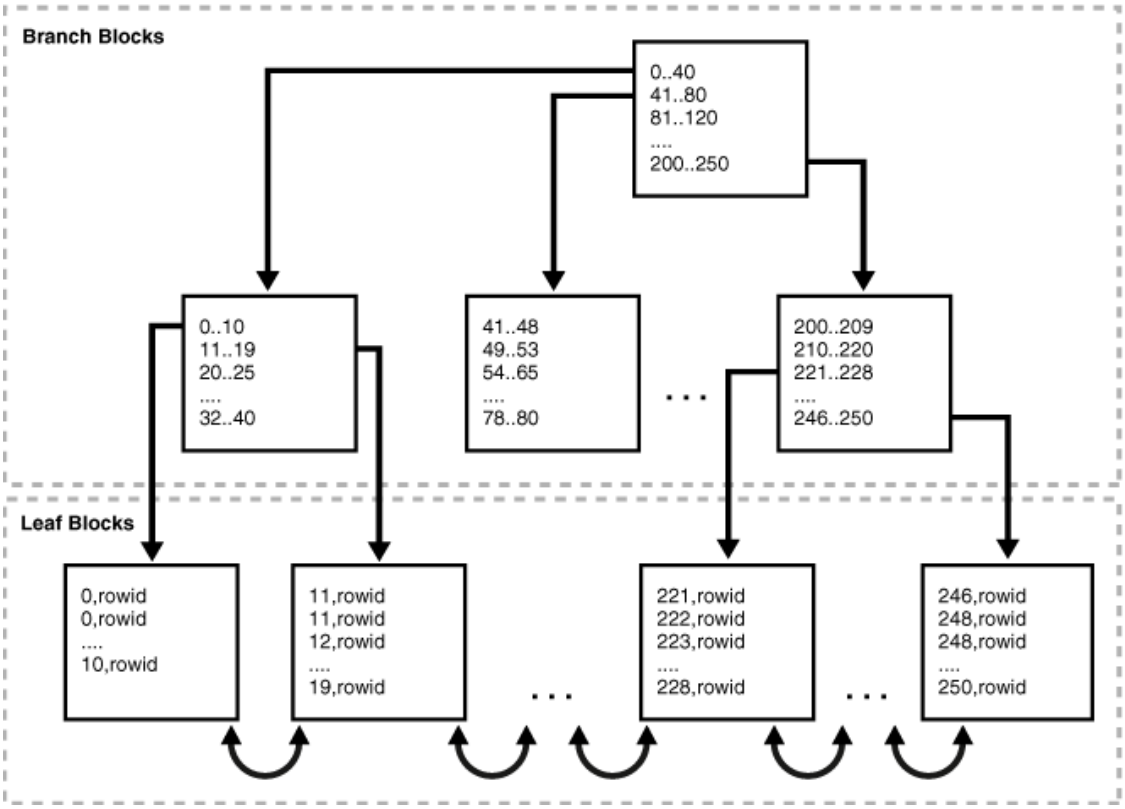
Ejemplo:

Estructura interna de un árbol B* en Oracle:



Ejemplo:

Estructura interna de un árbol B* en Oracle con datos numéricos:


Branch Blocks
0..40
41..80
81..120
....
200..250
0..10
11..19
20..25
....
32..40
41..48
49..53
54..65
....
78..80
200..209
210..220
221..228
....
246..250
Leaf Blocks
0,rowid
0,rowid
....
10,rowid
11,rowid
11,rowid
12,rowid
....
19,rowid
221,rowid
222,rowid
223,rowid
....
228,rowid
246,rowid
248,rowid
248,rowid
....
250,rowid