# SQL

Transacciones

# Transacciones

- Un problema puede surgir en la actualización de información, supongamos que en el mismo instante se realizan dos transacciones críticas como transacciones financieras:

  Se debe garantizar que un "deposito a cuenta" de un cliente genere una "deducción de cuenta" a otro. Que ocurre si se depositan $10,000 al "usuario X", pero a la mitad de la transacción el equipo de computo falla, y al "usuario Y" no se le deducen los $10,000 ?

# ACID

- En bases de datos se denomina **ACID** a un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción.
- **ACID** es un acrónimo de

**A**tomicity (Atomicidad)
**C**onsistency (Consistencia)
**I**solation (Aislamiento )
**D**urability (Durabilidad)

- **Atomicidad:** es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
- **Consistencia:** es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.
- **Aislamiento:** es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.
- **Durabilidad:** es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.
- Cumpliendo estas 4 condiciones se considera ACID Compliant

- Existen varios vendedores de Bases de Datos,pero quizás la marca que tiene mayor mercado es *Oracle* y en orden de uso posiblemente le sigan: Sybase, Postgres ,MySql y DB2; todas ofrecen las funcionalidades antes mencionadas y utilizan el lenguaje SQL.
- Sin embargo, debido a la misma complejidad de las operaciones que se llevan acabo, cada vendedor utiliza diferentes algoritmos y diseños por lo que el migrar y aprender a utilizar un producto de cierta compañía requiere de un esfuerzo e inversión substancial.

# COMMIT

- Se refiere a la idea de hacer que un conjunto de cambios "tentativos, o no permanentes" se conviertan en permanentes.
- Un uso popular es al final de una transacción de base de datos.
- En SQL finaliza una transacción de base de datos dentro de un sistema gestor de base de datos relacional (RDBMS) y pone visibles todos los cambios a otros usuarios.
- El formato general es emitir una sentencia BEGIN WORK, una o más sentencias SQL, y entonces la sentencia COMMIT.

# MYSQL COMMIT

- Libera los recursos bloqueados por cualquier actualización hecha con la transacción actual (LOCK TABLE).
- Por ejemplo:

  DELETE FROM T_PEDIDOS WHERE COD_PEDIDO=15;

  COMMIT;

Borra un registro y guarda los cambios.

# ROLLBACK

- Deshace los cambios de la transacción en curso.
- Libera los recursos bloqueados por cualquier actualización hecha con la transacción actual (LOCK TABLE).
- Por ejemplo:

```
DELETE FROM T_PEDIDOS WHERE COD_PEDIDO=15;
ROLLBACK;
```

Borra un registro pero cancela los cambios. Queda como si no hubiésemos hecho nada.

```
START TRANSACTION;
SELECT @A:=PRESUPUESTO FROM
departamentos_externos WHERE codigo=11;
UPDATE departamentos SET PRESUPUESTO
= PRESUPUESTO + @A WHERE codigo=33;
COMMIT;
```

```
START TRANSACTION;
SELECT @A:=PRESUPUESTO FROM
departamentos_externos WHERE codigo=11;
INSERT INTO departamentos( codigodep,
nombredep, presupuesto ) VALUES (@B , @C ,
@A );
ROLLBACK;
```

# Bloqueos por defecto

- Las sentencias del DML pueden producir bloqueos sobre las filas de la tabla:
- Una sentencia SELECT normal no bloquea filas.
- Las sentencias INSERT, UPDATE o DELETE realiza un bloqueo ROW EXCLUSIVE de las filas afectadas por el WHERE.
- Las sentencias COMMIT y ROLLBACK desbloquean las filas bloqueadas anteriormente dentro de la transacción actual.
- Aunque una fila este bloqueada (por otra transacción), siempre podemos hacer un SELECT sobre esa fila. Los valores retornados son los anteriores al bloqueo.
- Las sentencias UPDATE Y DELETE pueden provocar o sufrir esperas si hay conflictos con otra transacción.

```
drop procedure if exists CrearActor;
delimiter %%
create procedure CrearActor (in nombre varchar(50), in apellido varchar(50), out exito int)
begin

start transaction;

insert into actor(first_name, last_name, last_update)
values(nombre, apellido, now());

select left(nombre, 1) as primera_letra;

if left(nombre, 1) <> "F" then
    set exito = 0;
    rollback;
else
    set exito = 1;
    commit;
end if;

end
%% delimiter ;
```

```
call CrearActor("Mario", "Bros", @resultado);
select @resultado;
select * from actor;
```

# Manejando excepciones

```
use sakila;
drop  procedure if exists AgregarActor;

delimiter %%
create procedure AgregarActor(in nombre varchar(50), in apellido varchar(50), out exito integer)
begin

   DECLARE exit handler for sqlexception
   BEGIN
   ROLLBACK;
   set exito = 0;
   end;

   START TRANSACTION;
      if LEFT (nombre, 1) <> "F" then
         insert into actor(first_name, last_name) values (nombre, apellido);
      else
         rollback;
      end if;
      set exito = 1;
   COMMIT;
END
%% delimiter ;
```

```
call AgregarActor(1/0, "Malo Pinza", @e);
select @e;
```

# SQL

INDICES

# Índices

- A veces queremos obtener registros especificando los valores *en uno o más registros*, ej.,
  - Encontrar todos los estudiantes en el departamento de “computación”
  - Encontrar todos los estudiantes con un prom > 3

# Índice

- Es una estructura de datos que mejora la velocidad de las operaciones, permitiendo un rápido acceso a los registros de una tabla.
- Al aumentar drásticamente la velocidad de acceso, se suelen usar sobre aquellos campos sobre los cuales se hagan frecuentes búsquedas.

# Índice

- Tiene un funcionamiento similar al índice de un libro, guardando parejas de elementos: el elemento que se desea indexar y su posición en la base de datos.
- Para buscar un elemento que esté indexado, sólo hay que buscar en el índice dicho elemento para, una vez encontrado, devolver el registro que se encuentre en la posición marcada por el índice.

# Índice

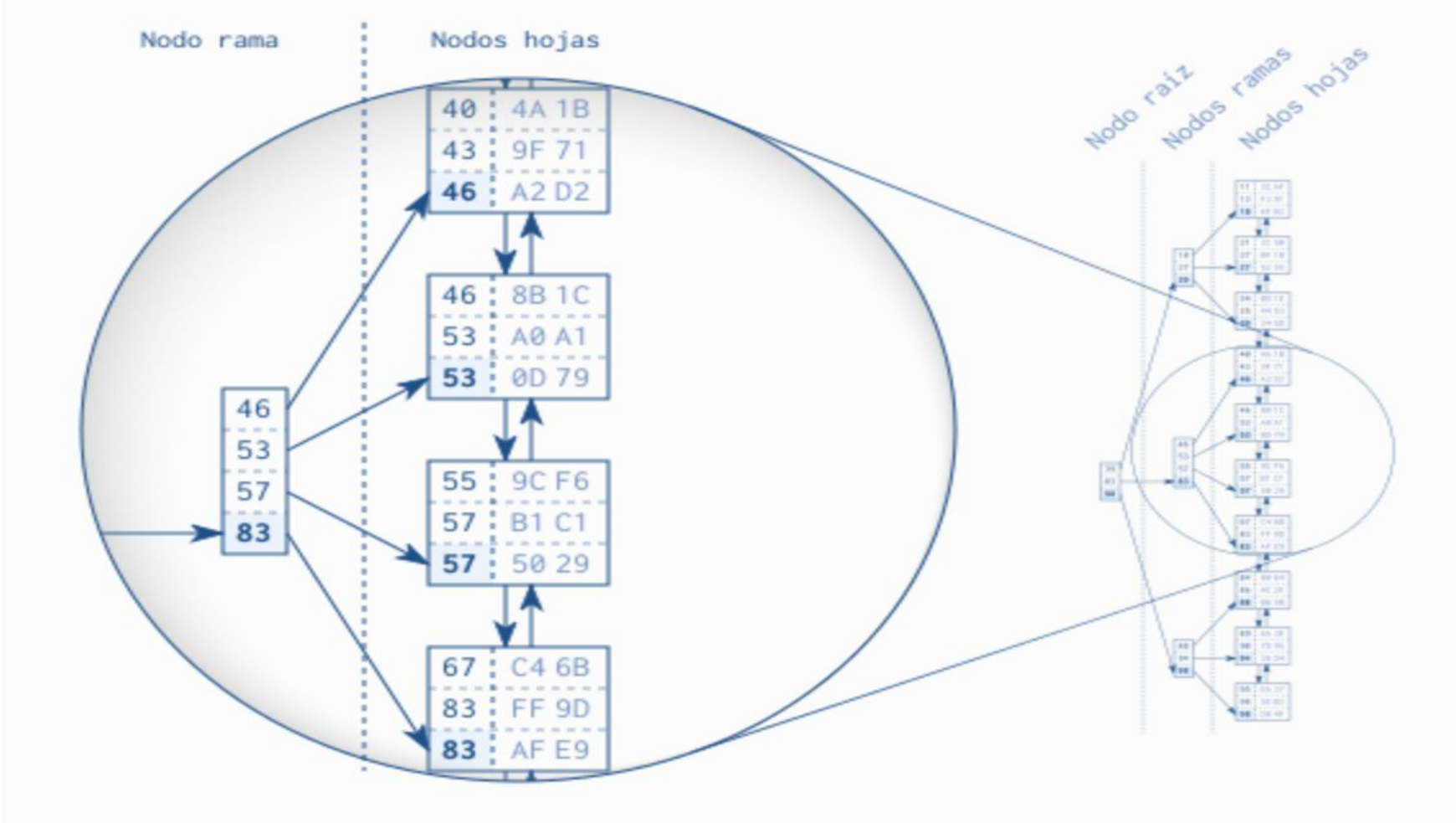
- Los índices pueden ser creados usando una o más columnas, proporcionando la base tanto para búsquedas rápidas al azar como de un ordenado acceso a registros eficiente.
- Campos que se hacen queries seguidos.
- Campos con alta cardinalidad.
- Registros peque.os y tama.os fijos son preferidos.

- El espacio en disco requerido para almacenar el índice es típicamente menor que el espacio de almacenamiento de la tabla
- Los índices generalmente contienen solamente los campos clave de acuerdo con los que la tabla será ordenada, y excluyen el resto de los detalles de la tabla.

# Clasificación de índices

- Representación de entradas de datos en los índices
  - ej., qué tipo de información está guardando el índice?
- Índices Primarios vs. Secundarios
- Índices Clustered vs. Unclustered
- Índices Llave Simple vs. Compuestos
- Basados en arbol
- Asociativa (Hash).

# Basados en árboles B, B+ TREE



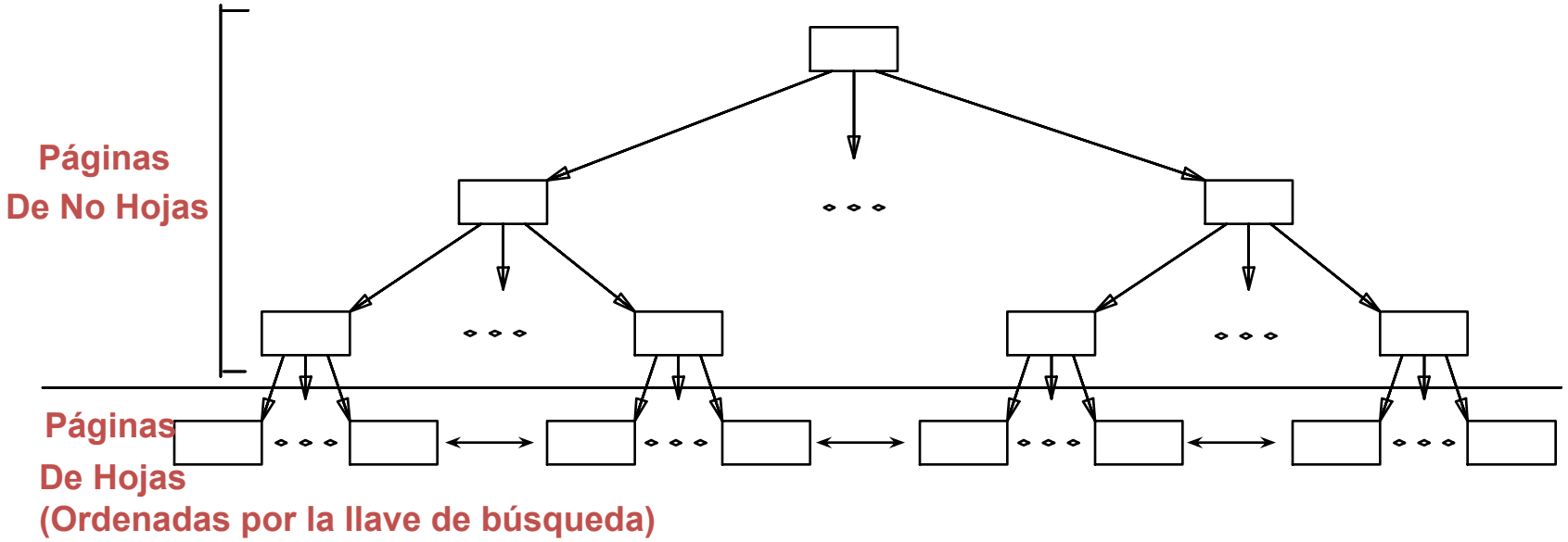
# Simulador de B+tree

- https://www.cs.usfca.edu/~galles/visualization/BTree.html

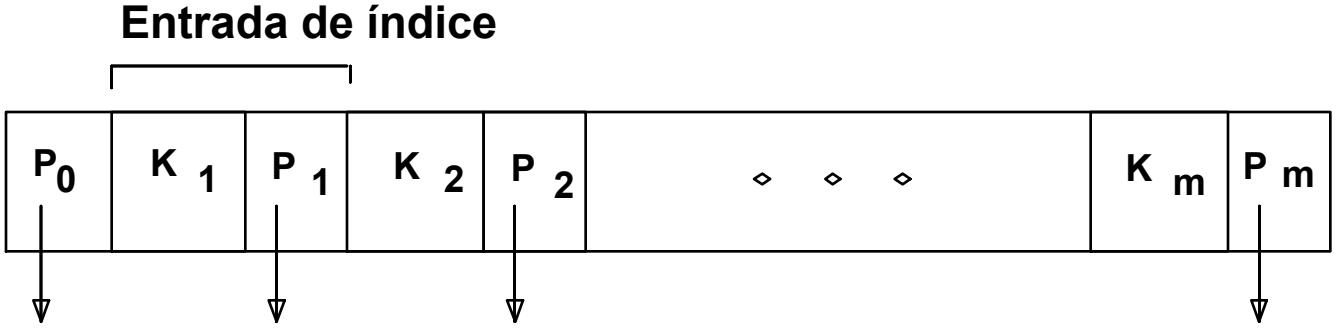
- https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

- Los árboles B + no almacenan el indicador de datos en los nodos interiores, solo se almacenan en los nodos de hoja. Esto no es opcional como en B-Tree. Esto significa que los nodos interiores pueden encajar más teclas en el bloque de memoria.
- Los nodos de hoja de los árboles B + están vinculados, por lo que hacer un escaneo lineal de todas las claves requerirá un solo paso a través de todos los nodos de hoja. Un árbol B, por otro lado, requeriría un recorrido de cada nivel en el árbol. Esta propiedad también puede utilizarse para una búsqueda eficiente, ya que los datos se almacenan solo en hojas.

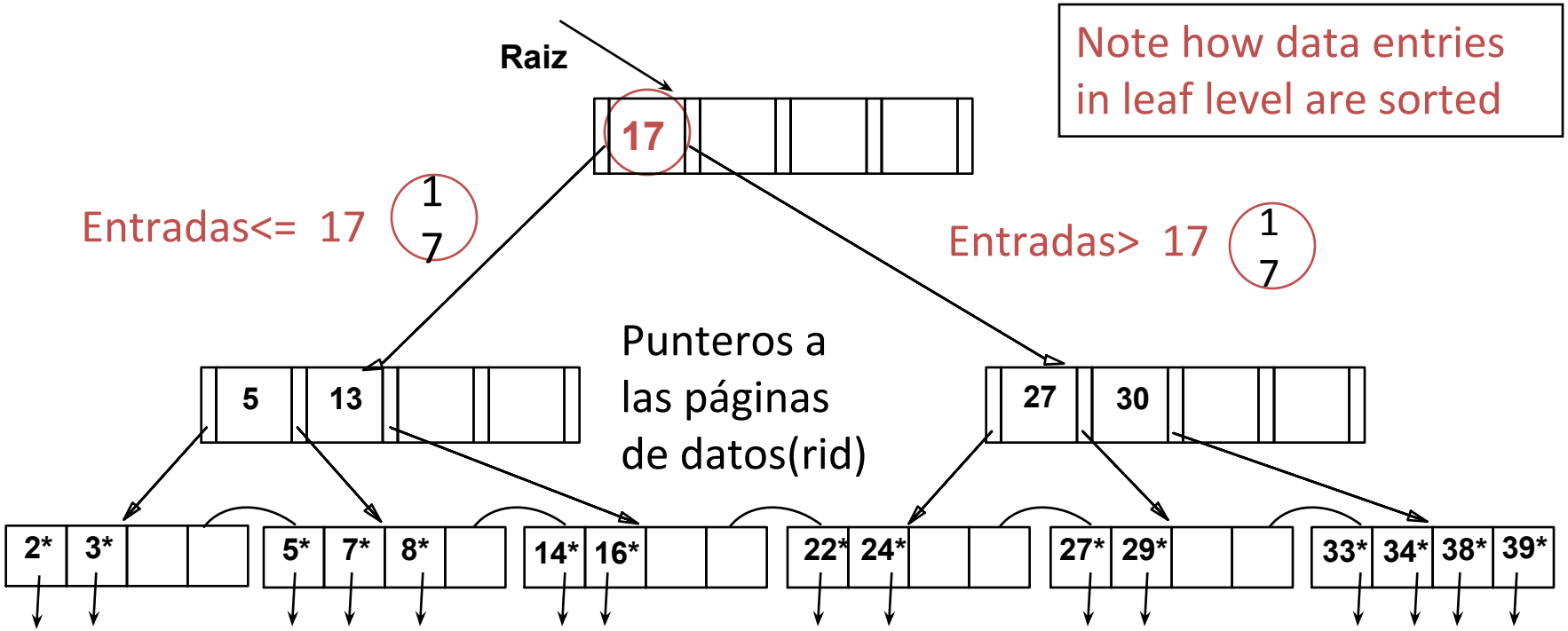
# Índices B+ Tree




- Las páginas de hojas contienen *entradas de datos*, encadenadas(prev & next)
- Las páginas de no hojas tienen *entradas de índices;* solo usadas para búsquedas directas:



# Ejemplo B+ Tree



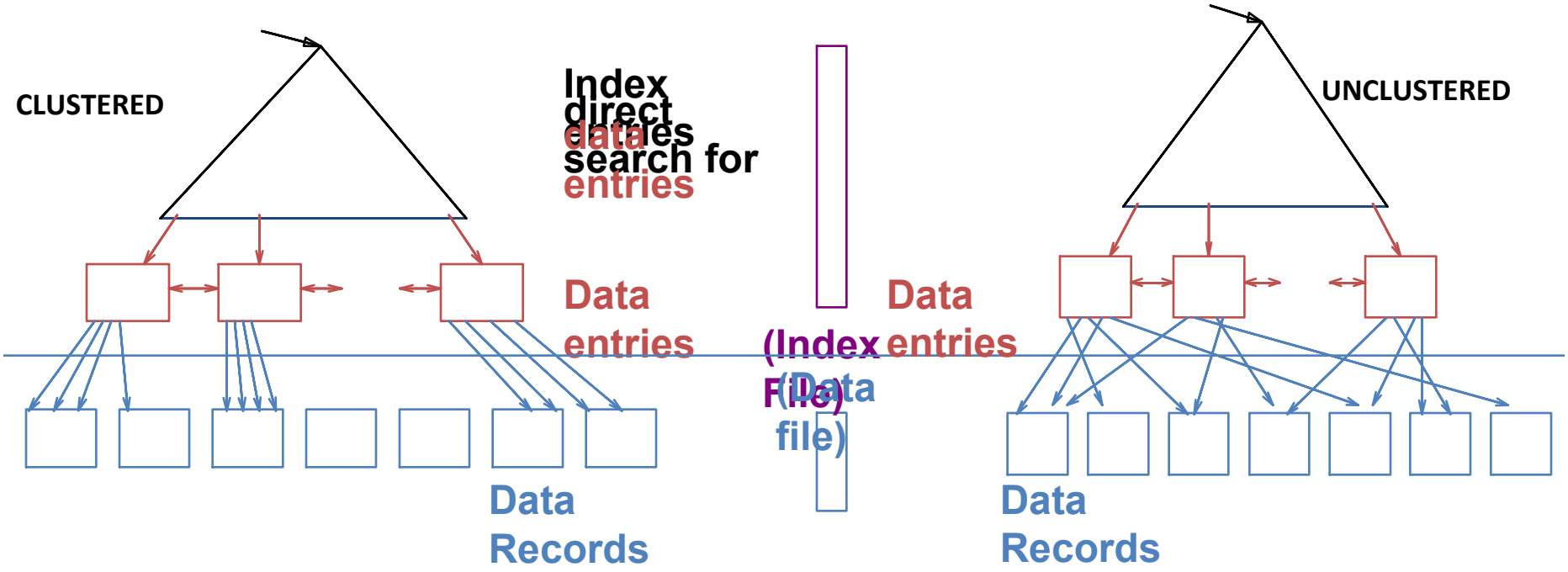

- Ingresar/borrar: Encontrar los datos de entradas en la hoja, luego cambiarlos. A veces se necesita ajustar el padre.

- *Primarios* vs. *Secundarios*: Si la clave de búsqueda contiene una clave primaria entonces se llama índice primario.
  - *Único* índice: La clave de búsqueda contiene una clave candidata.

# Clasificación de índices

- *Clustered* vs. *unclustered*: Si el orden de los registros de datos es lo mismo, o parecido al orden de los registros de datos de índices, entonces se llama *índice clustered*.
  - Un archivo puede ser clustered en al menos una clave de búsqueda.
  - El costo de seleccionar registros de datos a través de índices varia mucho si un índice es clustered o no

# Clustered vs. Unclustered



# Unclustered vs. Clustered

- Clustered Pros
  - Eficiente para búsquedas de rango
  - Se puede hacer algún tipo de compresión
- Clustered Cons
  - Caro de mantener

# Cuándo utilizar índices?

- Campos que se hacen queries seguidos
- Campo con alta cardinalidad
- Registros pequeños y tamaños fijos son preferidos.

(Obs: La mayoría de los DBMSs indexan automáticamente el PK)

# Create Index command

- Create index <iNombre> on
  <nombre_tabla> (<col_nombre>);

- Ejemplo:
- Create index ieid on estudiantes(eid);

# Indexes (Defaults)

- Cada vez que un PK es creado, un índice es automáticamente creado.
- Cada vez que el tipo de índice no se especifica, el tipo de índice creado es un B-Trees.