# SQL DDL

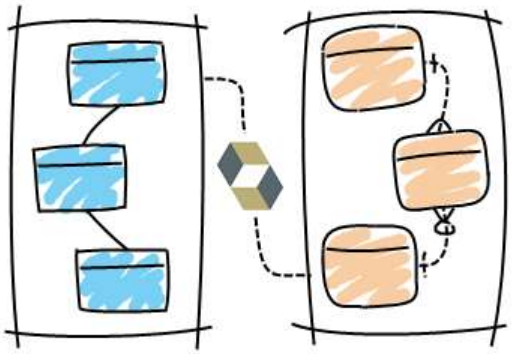
Esc. Normal Superior N° 10
Analista de Sistemas
Seminario 2017
Prof.: Fabio Dos Santos

# Temario

- Tipos de datos
- Creación, eliminación y modificación de esquemas
- Restricciones
- Claves e índices

# DDL

- Comandos del Lenguaje de Definición de Datos o DDL (Data Definition Language): permiten crear bases de datos, definir esquemas de relación, crear índices, seguridad y accesos.



# Creación de una DB

- CREATE DATABASE <nombre_db> crea una nueva base de datos
- USE <nombre_db>: Cambia el contexto de la DB a la DB especificada

# Atributos y Dominio

- Atributo: En bases de datos, un atributo representa una propiedad de interés de una entidad. En SQL un atributo es llamado columna.
- Dominio: describe un conjunto de posibles valores para cierto atributo. Matemáticamente, atribuir un dominio a un atributo significa "todos los valores de este atributo deben de ser elementos del conjunto especificado". Distintos tipos de dominios son: enteros, cadenas de texto, fecha, etc.

# Tipos de datos

| Nombre | Tipo | Bytes | Descripción |
|---|---|---|---|
| Bit | Entero | 1* | 0 ó 1 (equivalente a boolean en otros lenguajes) |
| * Si una tabla contiene 8 columnas o menos de tipo bit, éstas se almacenan como 1 byte. Si hay entre 9 y 16 columnas de tipo bit, se almacenan como 2 bytes, y así sucesivamente. | | | |
| Tinyint | Entero | 1 | De 0 a 255 |
| Smallint | Entero | 2 | De -2^15 (-32,768) a 2^15 – 1 (32,767) |
| Int | Entero | 4 | De -2^31 (-2,147,483,648) a 2^31 - 1 (2,147,483,647) |
| Bigint | Entero | 8 | De -2^63 (-9223372036854775808) a 2^63 – 1 (9223372036854775807) |

# Sobre los tipos de datos enteros

- El tipo de datos int es el principal tipo de datos de valores enteros de SQL Server. El tipo de datos bigint está pensado para utilizarse cuando los valores enteros pueden exceder el intervalo admitido por el tipo de datos int.
- bigint se encuentra entre smallmoney e int en el gráfico de prioridad de tipo de datos.
- Las funciones sólo devuelven bigint si la expresión de parámetro es un tipo de datos bigint. SQL Server no convierte automáticamente otros tipos de datos enteros (tinyint, smallint e int) en bigint.

# Tipos de datos

| Nombre | Tipo | Precisión | Bytes | Descripción |
|---|---|---|---|---|
| Decimal [(p[, s])]<br>Numeric [(p[, s])] | Real | 1 - 9 | 5 | p dígitos totales llamado precision (a la izquierda y a la derecha del punto decimal) y s dígitos decimales llamado escala. La escala debe ser un valor comprendido entre 0 y p. |
| | | 10-19 | 9 | |
| | | 20-28 | 13 | |
| | | 29-38 | 17 | |
| Money | Real | | 8 | 4 dígitos decimales entre -2^63 (-922,337,203,685,477.5808) y 2^63 - 1 (+922,337,203,685,477.5807) |
| Smallmoney | Real | | 4 | 4 dígitos decimales entre -214,748.3648 y 214,748.3647 |
| Float | Real | 1-24 = 7 | 4 | Punto flotante entre -1.79E + 308 y 1.79E + 308 |
| | | 25-53 =15 | 8 | |
| Real | Real | | 4 | punto flotante entre -3.40E + 38 y 3.40E + 38 |

# Tipos de datos

- Char [(n)] Cadena de caracteres de longitud fija con n caracteres, n puede ser un valor entre 1 y 8000, si no se especifica se asume 1.
- Varchar [(n)] Cadena de caracteres de longitud variable con un máximo de n caracteres, n puede ser un valor entre 1 y 8000, si no se especifica se asume 1.
- Text Cadena de caracteres de longitud variable con un máximo de 231-1 (2,147,483,647) caracteres.
- Datetime Fecha y hora entre 1 de Enero de 1753 y 31 de Diciembre de 9999. Los intervalos de horas pueden tener precisión de 3.33 milisegundos.
- Smalldatetime Fecha y hora entre 1 de Enero de 1900 y 6 de Junio del 2079. Los intervalos de horas pueden tener precisión de 1 minuto.
- Binary [(n)] Cadena de caracteres binarios de longitud fija con un máximo de 8000 caracteres.
- Varbinary [(n)] Cadena de caracteres binarios de longitud variable con un máximo de 8000 caracteres.
- Image Cadena de caracteres binarios de longitud variable con un máximo de 231-1 (2,147,483,647) caracteres.
- Timestamp Un numero único de 8 bytes por cada fila en la base de datos que actualiza el sistema cada vez que cambia un valor de la fila (comando INSERT o UPDATE). Una tabla puede tener un único atributo con este dominio.
- Uniqueidentifier Un identificador único global de 16 bytes dentro de la Base de Datos (GUID)

# Creación de esquemas

- Un esquema o tabla se define utilizando la orden CREATE TABLE correspondiente al lenguaje de definición de datos (DDL).

  CREATE TABLE r
  ( A1 D1 , A2 D2 , ... , An Dn )
- Donde:
  - r es el nombre de la tabla
  - A1 , A2 , ... , An: nombres de atributos
  - D1 , D2 , ... , Dn: dominios de atributos
- Ejemplo:

```
CREATE TABLE cliente
( cli_id INT,
cli_razon VARCHAR(30),
cli_domicilio VARCHAR(20),
cli_depto CHAR(3),
cli_piso TINYINT,
cli_localidad VARCHAR(20)
prov_id CHAR(3)
)
```

# DEFAULT

- Valor predeterminado para un atributo. Tal valor será ingresado en el atributo cuando se inserte una fila que no tenga valores para dicho atributo.
- Ejemplo

```
CREATE TABLE cuenta
( cli_id INT,
cli_saldo INT DEFAULT 0 )
```

# IDENTITY

- Se puede utilizar en las columnas con tipos de datos escalares (enteros: bigint, int, smallint, tinyint), numeric o decimal. Proporciona automáticamente los valores correspondientes a dicho atributo a partir de un valor inicial conocido como semilla (identity seed) e incrementado dicho valor por cada inserción de filas que se realiza en la relación con un valor conocido como incremento (identity increment).
- Solo se permite un atributo con esta propiedad por tabla. Por lo general, este atributo es además clave en la relación.
- El atributo que cuente con la propiedad identity no puede ser modificado
- Si se eliminan filas de una relación, los valores asignados a los atributos con propiedad identity se pierden.
- La forma de definir esta propiedad a un atributo es:

```
CREATE TABLE r
( A1 D1,
...
Ax Dx IDENTITY [ (semilla [, incremento) ],
...
An, Dn
).
```

# IDENTITY

- En el tipo de dato numeric ó decimal, solo se permite definir esta propiedad si la escala es 0 (es decir si no tiene decimales).
- No se permite el valor Null para atributos definidos con esta propiedad.
- Si se omite el valor de semilla se asume 1 y si se omite el valor del incremento también se asume 1.
- Ejemplo:

```
CREATE TABLE provincias
( prov_id TINYINT IDENTITY,
prov_desc VARCHAR(20)
)
CREATE TABLE t1
( a INT IDENTITY(0,5),
b int
)
```

Al insertar filas en la tabla t1, la columna a tomaria los valores 0, 5, 10, 15, etc.

# IDENTITY

- Si se necesita especificar el valor de un atributo definido con la propiedad identity, se utiliza el comando SET IDENTITY_INSERT <nombre de la tabla> ON
- Ejemplo:

```
SET IDENTITY_NSERT provincias ON
INSERT INTO provincias(prov_id,prov_desc)
VALUES (4,'SANTA FE')
SET IDENTITY_INSERT provincias OFF
```

- Para saber el ultimo numero asignado por el sistema en un atributo con propiedad se utiliza la variable del sistema @@IDENTITY o la funciones IDENT_SEED, IDENT_INCR, IDENT_CURRENT que toman como argumento una tabla y devuelven el valor de la semilla, el valor del incremento o el ultimo valor asignado.
- Ejemplo:

```
SELECT ident_seed('provincias'), ident_incr('provincias'),
    ident_current('provincias')
```

# Restricciones

- Son reglas definidas en las relaciones que se utilizan para mantener la integridad de los datos.
- Bajo las mismas se pueden definir claves primarias, control de valores únicos, control de valores no nulos, claves foráneas o restricciones en los valores de un dominio.
- Las restricciones se pueden aplicar a un atributo (se declara después de la definición del atributo en el esquema) o a varios atributos (exige colocar un nombre a la restricción).

# Valores no nulos

- En forma prederminada, cada atributo incluye al valor NULL en su dominio, excepto los atributos que se definan como clave primaria.
- Si se desea que un atributo no incluya valores nulos se debe especificar al lado de la definición de dicho atributo la palabra reservada NOT NULL (NULL es el default).
- Por ejemplo en el esquema anterior si definimos:

```
CREATE TABLE cliente
( cli_id INT,
cli_razon VARCHAR(30) NOT NULL,
cli_domicilio VARCHAR(20),
cli_depto CHAR(3),
cli_piso TINYINT,
cli_localidad VARCHAR(20),
prov_id CHAR(3) NOT NULL
)
```

- estamos obligando el ingreso de los atributos cli_razon y prov_id cada vez que se inserte una upla.

# Restricciones de dominio

- Limitan los valores que pueden ser introducidos en un atributo.
- Se definen con la palabra reservada CHECK seguida de un predicado que deberá ser satisfecho para que la upla se inserte o se modifique en la relación.
- Ejemplo:

```
saldo MONEY CHECK (saldo > 0)
Nombre VarChar(30) CHECK nombre <> ‘’
cli_sexo CHAR(1) CHECK ( sexo IN (‘F’,’M’) )
Edad TinyInt CHECK (edad BETWEEN 18 AND 60)
Sigla Char(3) CHECK (sigla Like '[A-Z][0-9]_')
```

# Valores únicos

- La restricción UNIQUE fuerza el ingreso de valores únicos en un atributo (en las claves primarias esto se cumple por definición).
- Tener presente que de existir la restricción UNIQUE y que el atributo sobre el cual se define pueda contener valores nulos, se permitirá solo el ingreso de un valor NULL para todas las uplas de la relación.
- Ejemplo:

```
CREATE TABLE persona
( dni INT,
apellido VARCHAR(30),
nombres VARCHAR(30),
cuil INT UNIQUE )
```

# Clausula Constraint

- Permite establecer el nombre de una restricción, el mismo puede ser usado para captura de errores.
- **Convenciones:**

| Constraint | Prefix |
|---|---|
| check | ck |
| unique | un |
| primary key | pk |
| foreign key | fk |
| index | idx |

# Clausula Constraint

- **CONSTRAINT <nombre> CHECK ( <predicado>)**
- CONSTRAINT <nombre> UNIQUE(DNI)

# Claves primarias

- Se especifican con la palabras reservadas PRIMARY KEY.
- Si esta formada por un solo atributo, se especifica a continuación de la definición del atributo.

# Claves primarias

- Ejemplo:

```
CREATE TABLE cliente
( cli_id INT PRIMARY KEY,
...
cli_sexo CHAR(1),
saldo MONEY,
CONSTRAINT saldo_ok CHECK (saldo > 0),
CONSTRAINT sexo_ok CHECK sexo IN (‘F’,’M’) )
)
```

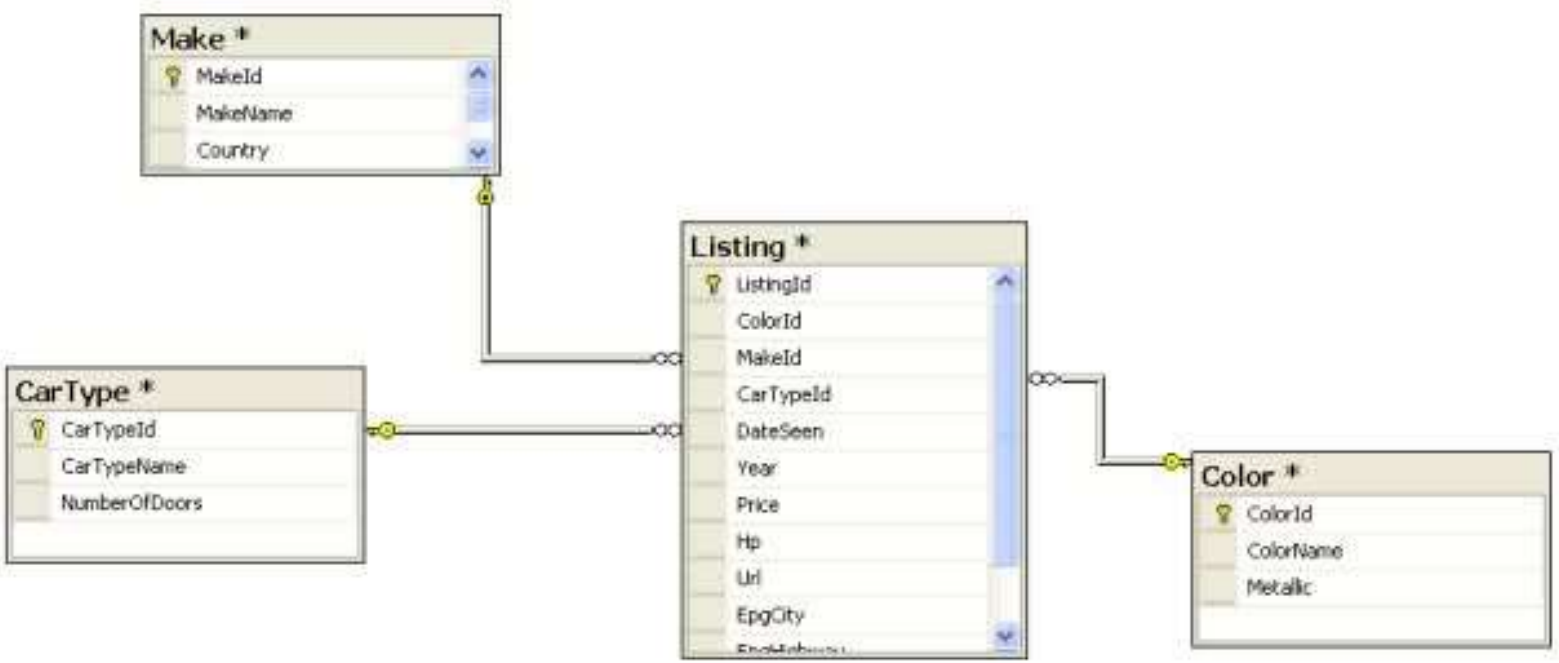
# Claves primarias

- Si es clave compuesta, se especifica al final de la relación con la palabra reservada CONSTRAINTS y asignando un nombre a la restricción.
- Ejemplo de una clave primaria compuesta

```
CREATE TABLE facturas_cab(
fc_nro_suc CHAR(4),
fc_nro INT,
fc_fecha SMALLDATETIME,
CONSTRAINT pk_factura_cab_fc_nro_suc_fc_nro PRIMARY KEY
    (fc_nro_suc, fc_nro)
)
```

# Claves foráneas

- Las claves foráneas apuntan a claves primarias de otras tablas. Con ellas se maneja la integridad referencial.
- La tabla que tiene la clave primaria se denomina tabla fuerte
- La tabla que tiene la clave foránea se denomina tabla débil



# Claves foráneas

- Si la clave foránea es un solo atributo, se especifica a continuación del atributo que se relaciona con la clave de otra tabla la cláusula FOREIGN KEY REFERENCES Ri(Ai) donde Ri es el nombre de la relación a la que se hace referencia y Ai es el atributo clave dentro de dicho esquema (Ri deberá ser forzosamente clave primaria de Ri)
- Ejemplo:

```
CREATE TABLE provincias
( prov_id CHAR(3) PRIMARY KEY,
prov_desc CHAR(20)
)
CREATE TABLE cliente
( cli_id INT,
cli_razon VARCHAR(30),
cli_domicilio VARCHAR(20),
cli_depto CHAR(3),
cli_piso TINYINT,
cli_localidad VARCHAR(20),
prov_codigo CHAR(3) FOREIGN KEY REFERENCES provincias(prov_id)
)
```

# Claves foraneas

- Si la clave foránea son atributos compuestos, se definen al final del esquema con la cláusula CONSTRAINT asignándole un nombre a la restricción.
- Ejemplo:

```
CREATE TABLE articulos
( art_id SMALLINT PRIMARY KEY,
art_desc VARCHAR(30)
)
CREATE TABLE facturas_cab
( fc_nro_suc char(4),
fc_nro INT,
fc_fecha SMALLDATETIME,
CONSTRAINT pk PRIMARY KEY (fc_nro_suc, fc_nro)
)
CREATE TABLE facturas_det
(fc_nro_suc char(4),
fc_nro INT,
art_id SMALLINT FOREIGN KEY REFERENCES articulos(art_id),
CONSTRAINT fd_fc FOREIGN KEY (fc_nro_suc,fc_nro) REFERENCES
facturas_cab(fc_nro_suc,fc_nro)
)
```

# Claves foraneas

- Se puede indicar al gestor de Base de Datos la acción a seguir en cado de modificación de algún atributo que sea parte de una clave foránea o en el caso de eliminar una upla relacionada. Las acciones posibles son:
  - ON DELETE { CASCADE | NO ACTION }
  - ON UPDATE { CASCADE | NO ACTION }
- En el ejemplo anterior si especificamos:

```
CREATE TABLE facturas_det
(
fc_nro_suc char(4),
fc_nro INT,
art_id SMALLINT FOREIGN KEY REFERENCES articulos(art_id),
CONSTRAINT fd_fc FOREIGN KEY (fc_nro_suc,fc_nro) REFERENCES
facturas_cab(fc_nro_suc,fc_nro) ON DELETE cascade ON UPDATE cascade
)
```

- Cada vez que se borre una upla de la tabla facturas_cab, se eliminaran todas las uplas asociadas en facturas_det y cada vez que se modificar el numero de sucursal o el numero de factura en facturas_cab se modificaran dichos atributos en las uplas relacionadas de facturas_det.

# Eliminación de esquemas

- Para eliminar una tabla se utiliza el comando DROP TABLE r
- Donde r representa al nombre de la tabla o relación.
- Para que la sentencia de borrado pueda ser cumplida, la tabla que estamos intentando eliminar no debe ser referenciada por otra tabla en alguna restricción de tipo FOREIGN KEY.

# Modificación de esquemas

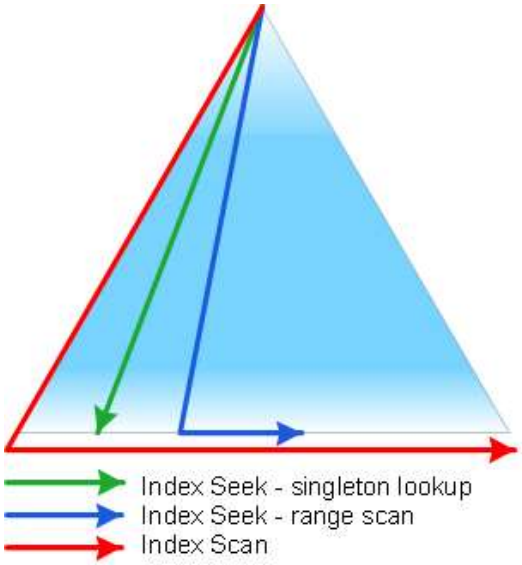
- Un esquema puede ser modificado agregando o quitando atributos, modificando dominios o eliminando restricciones y agregando restricciones.

  ALTER TABLE r
  [ ALTER COLUMN Ai Di NULL | NOT NULL ]
  [ DROP COLUMN Ai ]
  [ ADD Ai Di ]
  [ ADD CONSTRAINT <nombre_restriccion> CHECK *p* ]
  [ ADD CONSTRAINT <nombre_restriccion> FOREIGN KEY (A1,..,An)
  REFERENCES
  Ri(B1,..,Bn)
  [ ON DELETE { CASCADE | NO ACTION } ]
  [ ON UPDATE { CASCADE | NO ACTION } ] ]
  [ ADD CONSTRAINT <nombre_restriccion> PRIMARY KEY (A1,..,An) ]
  [ DROP CONSTRAINT <nombre_restriccion> ]

- Donde:
  - Ai, Bi corresponden a nombres de atributos.
  - Di corresponde a un tipo de datos.
  - *p* corresponde a un predicado.
  - Ri corresponde al nombre de otro esquema o relación.

# Índices

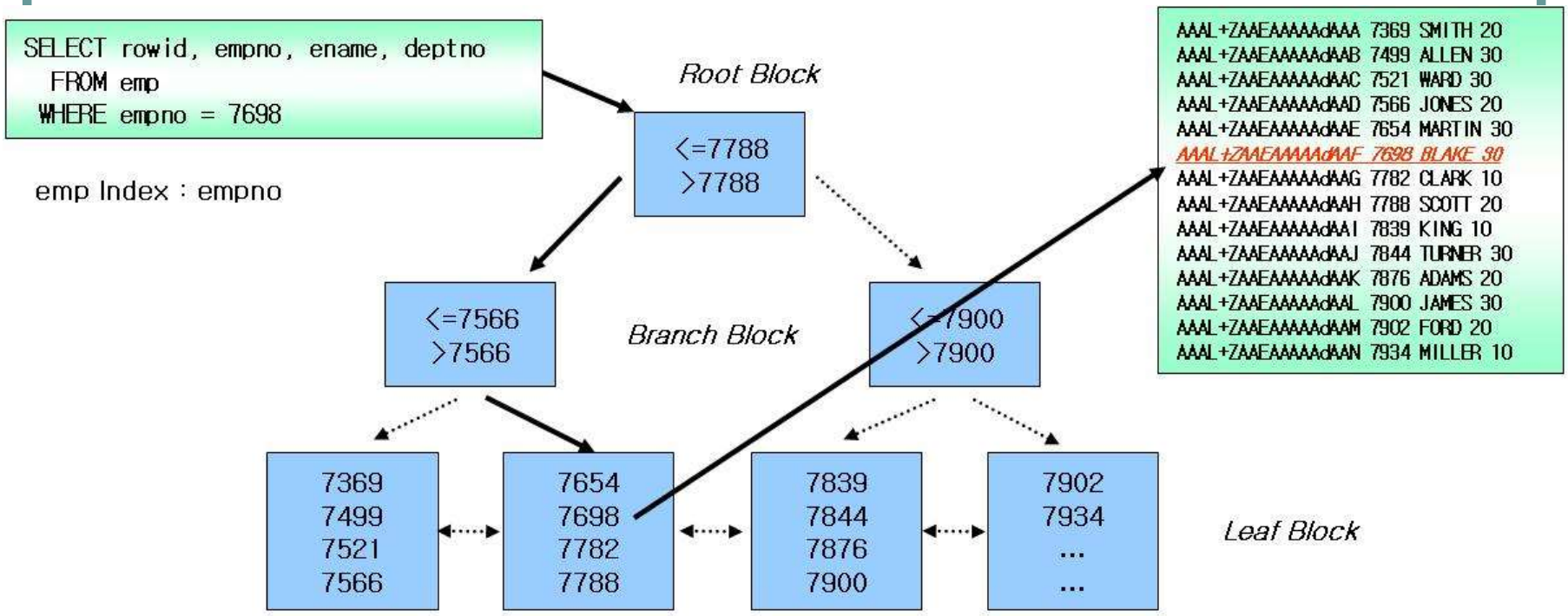
- MS Sql Server almacena las filas de datos en paginas. Cada pagina de datos contiene n filas.
- A su vez, se realiza una agrupación de paginas que se conoce con el nombre de Heaps. Cada Heap contiene m paginas.
- Existen 2 maneras de acceder a los datos:
  - Index Scan: Recorrer todas las paginas de una tabla, en lo que se denomina un recorrido de tabla. Se comienza por el principio de la tabla, recorriendo pagina por pagina y extrayendo las filas que cumplen los criterios de la consulta.
  - Index seek – range scan: Utilizar índices que apuntan a los datos de la pagina. Se recorre primero la estructura de árbol del índice para buscar filas que solicita la consulta y extrae únicamente las filas necesarias que cumplen los criterios de la consulta.




- Dada una consulta, el planificador de consultas (que es el componente responsable de generar el plan de ejecución optimo de cada consulta), determina primero si existe un índice. Luego, determina si para el acceso a los datos resulta mas eficiente recorrer toda la tabla o utilizar un índice.

# Índices

- Un índice es un estructura en disco asociada a una tabla o una vista que puede acelerar la recuperación de registros.
- Un índice basa su funcionamiento en la definición de claves, que son uno o más campos de la tabla o vista que se organizan en una típica estructura b-Tree



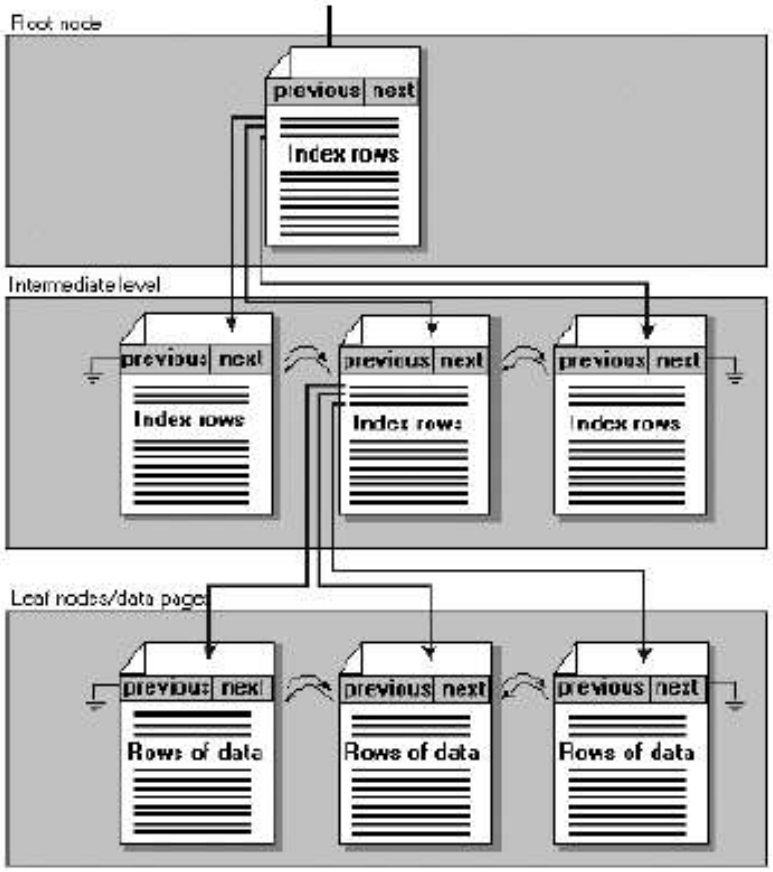
# Índices: pros y contras

- Ventajas:
  - Aceleran la recuperación de datos: si no existe un índice, se debe recorrer toda la tabla.
  - Por consecuencia aceleran las consultas que combinan tablas y que realizan operaciones de ordenamiento o agrupamiento.
  - Permiten forzar la unicidad de las filas.
- Desventajas:
  - Consumen espacio en disco
  - Requieren costo adicional de procesamiento y memoria para su mantenimiento.
- Conclusión:
  - Los índices mas adecuados son los creados con columnas que tienen un alto grado de selectividad.ç
  - No se deberían crear índices en columnas que rara vez se consulten, que no intervengan en operaciones de juntas o que no requieran de un acceso especifico.

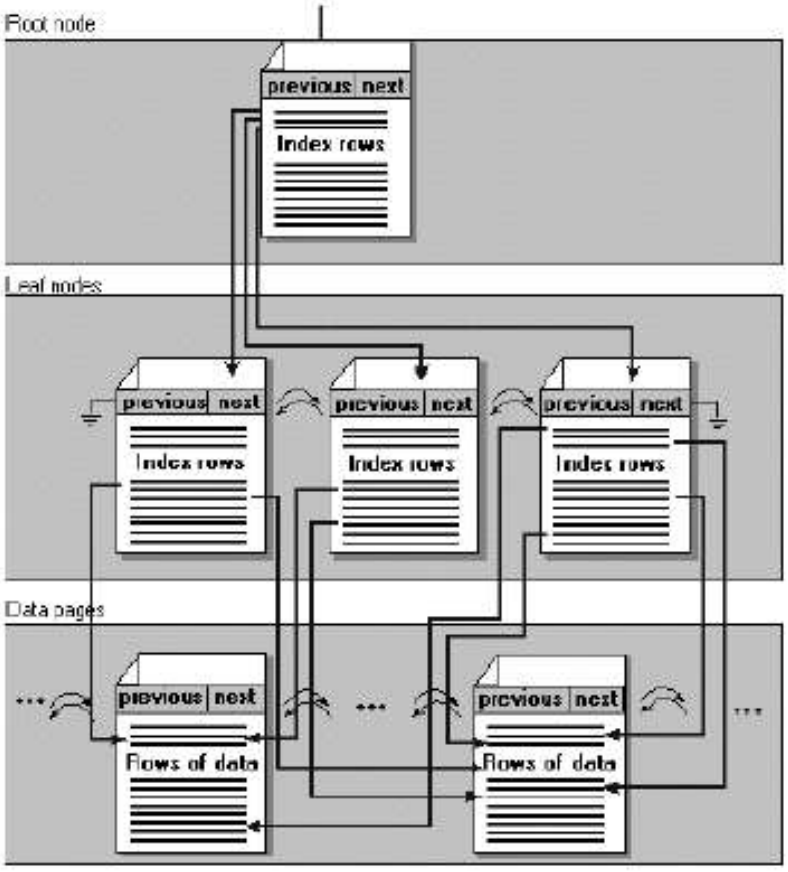
# Arquitecturas de los índices

- Indices agrupados (cluster index)
  - En un índice agrupado, el nivel de las hojas del árbol correspondiente al índice es la pagina de datos. Los datos están almacenados físicamente en orden ascendente en las paginas de datos. El orden físico es igual al orden lógico.
  - Cada tabla solo puede tener un índice agrupado. Si una tabla tiene un conjunto de atributos que forman la clave primaria, el índice que se define sobre ese conjunto de atributos es un índice agrupado.
  - Debe crearse primero el índice agrupado antes de crear los índices no agrupados, dado que el primero cambia el orden físico de las filas de la tabla.
- Indices no agrupados (non cluster index).
  - También son generados con una estructura de árbol B, pero los nodos hojas del árbol no contienen paginas de datos, sino que tienen punteros a las paginas de datos donde están ubicados los datos correspondientes a las claves de búsqueda.

# Arquitecturas de los índices




Indice agrupado




Indice no agrupado

# Creación y eliminación de índices

- Si existe una restricción de PRIMARY KEY al crear una tabla, automáticamente se genera un índice agrupado.
- De la misma, si se utiliza la restricción UNIQUE, se genera un índice no agrupado por cada restricción.
- Además de los índices mencionados, el usuario puede crear índices con la instrucción:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX
<nombre_indice> ON r(A1 [ASC | DESC],..,An [ASC | DESC])
```

- Donde:
  - r es el nombre de la tabla sobre la cual se genera el índice.
  - A1,...,An: son nombres de atributos de la tabla r que integran la clave de búsqueda del índice que se genera.
- Por default en cada atributo que forma parte del índice se toma orden Ascendente.
- Si se especifica la propiedad UNIQUE, se crea un índice de manera tal que no se permiten en la relación 2 filas con los mismos valores en los índices.

# Creación y eliminación de índices

- Ejemplo:

  CREATE INDEX ape_nom ON empleado(apellido, nombres)
- Se genera el índice ape_nom en la tabla empleado por apellido y nombres en orden ascendente.
- Para eliminar un índice, se utiliza el comando DROP INDEX, seguido del nombre de la tabla sobre la cual se definió el índice y separado por un punto el nombre del índice a eliminar.
- Ejemplo:

  DROP INDEX empleado.ape_nom