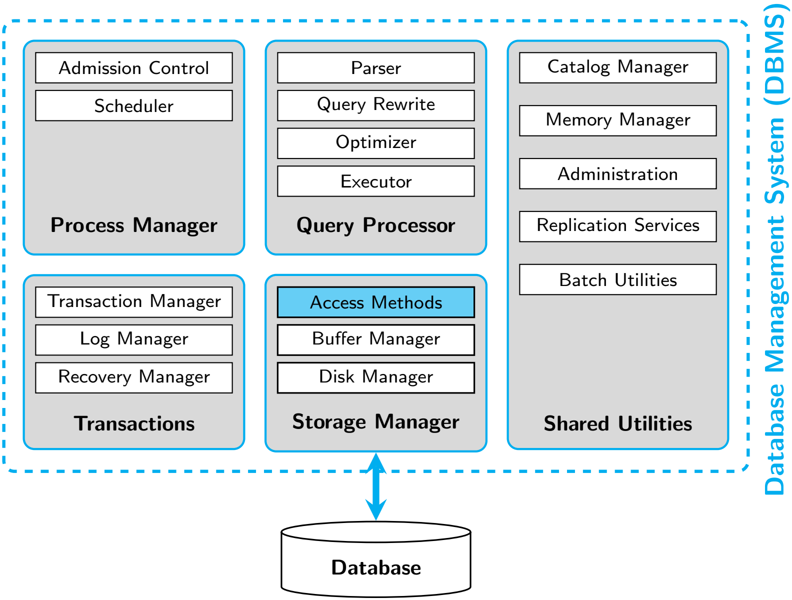
Índices Multidimensionales

Clase 07

IIC 3413

Prof. Cristian Riveros

Índices multidimensionales



Motivación

Supongamos la siguiente consulta:

```
SELECT   *
FROM     Customers
WHERE    cAge BETWEEN 30 AND 60          AND
         cSalary BETWEEN 1MM AND 3.5MM
```

- Tabla *Customers* cuenta con un B+tree-index sobre *cAge*.
- Tabla *Customers* cuenta también con un B+tree-index sobre *cSalary*.

¿qué tan útiles son estos índices para responder la consulta?

Índices multidimensionales o “spatial”

Índices 1-dimensionales (B+-trees, static hashing, etc):

- busqueda basada en un search key **único**.

Índices **multidimensionales** (IM):

- multiples search key para una misma tupla.
- cada search key es igualmente importante.

¿qué ventajas tenían los índices 1-dimensionales en comparación a IM?

Consultas típicas de índices multidimensionales

1. Partial match queries.

Ejemplo

```
SELECT   *
FROM     Customers
WHERE    cAge = 31 AND cSalary > 3MM
```

Consultas típicas de índices multidimensionales

1. Partial match queries.
2. Range queries.

Ejemplo

```
SELECT   *
FROM     Customers
WHERE    cAge BETWEEN 30 AND 60                AND
         cSalary BETWEEN 1MM AND 3.5MM
```

Consultas típicas de índices multidimensionales

1. Partial match queries.
2. Range queries.
3. Nearest-neighbor queries.

Ejemplo

Dada una BD de ciudades y una ubicación en un mapa:

- "¿cuáles son las ciudades mas cercanas a mi ubicación actual?"

Consultas típicas de índices multidimensionales

1. Partial match queries.
2. Range queries.
3. Nearest-neighbor queries.
4. Where-am-I queries.

Ejemplo

Dada una base de datos de polígonos en un plano 2D (o 3D).

- "¿qué polígonos contienen mi punto (x_0, y_0)?"

Aplicaciones de índices multidimensionales

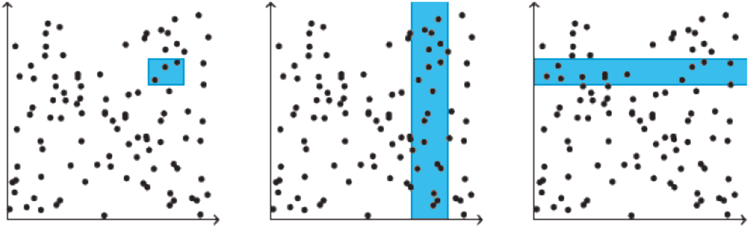
1. Consultas SQL con filtros sobre varios campos.
2. Datos geográficos.
   - GIS: Geographical Information Systems.
3. Datos astronómicas.
4. Diseño de circuitos.
5. Diseño asistido por computadora.

¿podemos usar B+-trees para estas aplicaciones?

```
SELECT   *
FROM     Customers
WHERE    cAge BETWEEN 30 AND 60             AND
         cSalary BETWEEN 1MM AND 3.5MM
```

Podemos responder esta consulta:

- ¿consultando un índice para *cAge*? ¿o un índice para *cSalary*?



¿podemos usar B+-trees para estas aplicaciones?

```
SELECT   *
FROM     Customers
WHERE    cAge BETWEEN 30 AND 60          AND
         cSalary BETWEEN 1MM AND 3.5MM
```

Podemos responder esta consulta:

- ¿consultando un índice para *cAge*? ¿o un índice para *cSalary*? ✗
- ¿haciendo la intersección de ambos índices? ✗
- ¿consultando un índice sobre la llave compuesta $(cAge, cSalary)$? ✗

B+-trees como índices multidimensionales

B+-trees están restrigidos a 1-dimension.

Objetivo de índices multidimensionales:

1. **simétrico** con respecto a cada dimension.
2. agrupe los datos según su posición en el **espacio**.
3. provee la mayor cantidad de **consultas distintas**.

Muchas propuestas para índices multidimensionales

- Quad Tree [Finkel 1974]
- R-tree [Guttman 1984]
- R+-tree [Sellis 1987]
- R*-tree [Geckmann 1990]
- Vp-tree [Chiueh 1994]
- UB-tree [Bayer 1996]
- SS-tree [White 1996]
- M-tree [Ciaccia 1996]
- Pyramid [Berchtold 1998]
- DABS-tree [Böhm 1999]
- Slim-tree [Faloutsos 2000]
- P-Sphere-tree [Goldstein 2000]
- K-D-B-Tree [Robinson 1981]
- Grid File [Nievergelt 1984]
- LSD-tree [Henrich 1989]
- hB-tree [Lomet 1990]
- TV-tree [Lin 1994]
- hB-pi-tree [Evangelidis 1995]
- X-tree [Berchtold 1996]
- SR-tree [Katayama 1997]
- Hybrid-tree [Chakrabarti 1999]
- IQ-tree [Böhm 2000]
- Landmark file [Böhm 2000]
- A-tree [Sakurai 2000]

Ninguna de ellas entrega
una solución 100% satisfactoria en todos los casos.

Tipos de índices multidimensionales

Es posible dividir estas propuestas en dos categorías:

- Basados en Hashing.
- Basados en árboles.
- Otros.

Nosotros veremos una de ellas: **Bitmap index**

Bitmap index

- Permite consultas multidimensionales.
- Basado en una codificación alternativa de las tuplas de una relación.
- Distinto a otros índices multidimensionales.

Estructura de un bitmap index

Para una relación R:

1. **Enumeramos** sus tuplas del 1 a $|R|$ de forma permanente:

$$R = \{t_1, t_2, \ldots, t_{|R|}\}$$

2. Para un atributo c de R y un valor $v \in \pi_c(R)$ definimos un **string binario** b_v^c de largo $|R|$ tal que:

$$b_v^c[i] = \begin{cases} 1 & \text{si } t_i(c) = v. \\ 0 & \text{en otro caso.} \end{cases}$$

3. Un **bitmap index** sobre un atributo c se define como el conjunto:

$$B^c = \{\, b_v^c \mid v \in \pi_c(R) \,\}$$

Estructura de un bitmap index

Ejemplo

Suponga la relación de alumnos y notas:

	sName	sMark
1	Pedro	6,0
2	Juan	6,0
3	Diego	4,0
4	Pedro	5,0
5	Diego	6,0
6	Juan	7,0

$$
\begin{array}{cccccccc}
 & & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\
b_{Pedro}^{sName} & = & 1 & 0 & 0 & 1 & 0 & 0
\end{array}
$$

B^{sName}:

Valor	Bitmap
Pedro	100100
Juan	010001
Diego	001010

Aplicación de bitmap index

```
SELECT   *
FROM     Students
WHERE    sName = 'Pedro' AND sMark = 6,0
```

¿cómo calculamos esta consulta usando B^{sName} y B^{sMark}?

Resultado:

$$b^{sName}_{Pedro} \ \texttt{AND} \ b^{sMark}_{6,0}$$

donde `AND` es aplicado **bitwise** (por cada bit):

$$(b \ \texttt{AND} \ b')[i] \ = \ b[i] \wedge b'[i] \quad \text{para todo } i.$$

Aplicación de bitmap index

```
SELECT   *
FROM     Customers
WHERE    cAge BETWEEN 30 AND 60
         cSalary BETWEEN 1MM AND 3.5MM
```

¿cómo calculamos esta consulta usando B^{cAge} y $B^{cSalary}$?

Resultado:

$$\texttt{OR}\left\{b_x \in B^{cAge} \mid 30 \leq x \leq 60\right\}$$

$$\texttt{AND}$$

$$\texttt{OR}\left\{b_y \in B^{cSalary} \mid 1\text{MM} \leq y \leq 3.5\text{MM}\right\}$$

¿cuáles son los inconvenientes de un Bitmap Index?

1. Tamaño de índice usa mucho espacio.
2. ¿cómo insertamos/eliminamos una tupla?
3. ¿cómo buscamos el bitmap que necesitamos en B^c?
4. ¿cómo encontramos (al final) las tuplas del resultado?

Compresión de bitmaps

Para n tuplas con m valores distintos tenemos:

$$\begin{array}{lccc} \text{Tamaño de cada bitmap} & = & n & \text{bits} \\ \text{Tamaño índice} & = & n \times m & \text{bits} \end{array}$$

Peor caso $m = n$ y tenemos que el tamaño del índice es n^2 , pero:

- si m es **grande**, entonces cada bitmap b se ve como:

$$b \;=\; 0\,0\,0 \cdots 0\,0\,0\,\mathbf{1}\,0\,0\,0 \cdots 0\,0\,0\,\mathbf{1}\,0\,0\,0 \cdots 0\,0\,0$$

- si m es **pequeño**, entonces $|B^c| \approx n$ bits.

Conclusión: bitmaps son altamente comprimibles!

Run-length-encoding (RLE)

Deseamos comprimir largas secuencias de 0's delimitados por 1:

$$b = \underbrace{\boxed{0 \quad 0 \quad 0 \quad 1}}_{3} \underbrace{\boxed{1}}_{0} \underbrace{\boxed{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1}}_{5} \underbrace{\boxed{0 \quad 0 \quad 0}}_{3}$$

Podemos guardar los largos de cada secuencia y así comprimir b.

$$\mathit{comprimir}(b) = 3053 \quad o \quad (11)(0)(101)(11)$$

¿cuál es el problema de guardar los largos en binario?

Run-length-encoding (RLE)

Solución: cada largo n lo guardamos como:

$$\underbrace{1\ 1\ 1 \cdots 1\ 1}_{k-1 \text{ veces}} \cdot 0 \cdot \mathrm{bin}(n)$$

- Largo k de $\mathrm{bin}(n)$ en unario.
- Seguido de $\mathrm{bin}(n)$.

Ejemplo

$$b = \underbrace{\boxed{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1}}_{13\ (=1\,1\,0\,1)}\ \underbrace{\boxed{1}}_{0}\ \underbrace{\boxed{0\ 0\ 0\ 1}}_{3}$$

$$13 \rightarrow \mathbf{1\ 1\ 1\ 0}\ 1\ 1\ 0\ 1 \qquad 0 \rightarrow \mathbf{0}\ 0 \qquad 3 \rightarrow \mathbf{1\ 0}\ 1\ 1$$

$$RLE(b) = \boxed{1\ 1\ 1\ 0\ 1\ 1\ 0\ 1}\ \boxed{0\ 0}\ \boxed{1\ 0\ 1\ 1}$$

Eficiencia de Run-length-encoding

Para n tuplas con m valores distintos.

- Cada secuencia de i-ceros y un uno, se codifica en:

$$2 \cdot \log_2(i) \text{ bits}$$

- Si $m \approx n$, entonces cada bitmap tiene aproximadamente un único 1.

$$2 \cdot \log_2(n) \text{ bits} \quad (\textit{approx.})$$

- Por lo tanto, B^c queda de tamaño $2 \cdot n \cdot \log_2(n)$. (asumiendo $m \approx n$)

$$2 \cdot n \cdot \log_2(n) << n^2$$

Operaciones `AND` y `OR` en Run-Length-Encoding

Para dos bitmaps b_1 y b_2:

- Descomprimimos $RLE(b_1)$ y $RLE(b_2)$ **en línea** (de izq. a der.).
- Comparamos bit a bit ejecutando `AND` u `OR`.

¿cuáles son los inconvenientes de un Bitmap Index?

1. Tamaño de índice usa mucho espacio. ✓
   - Solución: bitmaps son altamente comprimibles.

2. ¿cómo insertamos/eliminamos una tupla?

3. ¿cómo buscamos el bitmap que necesitamos en B^c?

4. ¿cómo encontramos (al final) las tuplas del resultado?

¿cómo insertamos/eliminamos una tupla en bitmap index?

Optimización de RLE para bitmap indexes

- Dado que conocemos la cantidad de tuplas, entonces podemos omitir la última secuencia de 0 en cada bitmap.
- Esto nos permite
  agregar un 1 en cualquier posición mayor o igual al último 1.

Explotamos esta propiedad para hacer insert.

¿cómo insertamos/eliminamos una tupla en bitmap index?

¿cómo **insertamos** una tupla?

- Para cada atributo c, buscamos el bitmap b_v^c con v el nuevo valor.
- Insertamos un nuevo 1 al final de b_v^c.

¿cómo **eliminamos** una tupla?

- Marcamos con un **tombstone** su posición.

¿cuáles son los inconvenientes de un Bitmap Index?

1. Tamaño de índice usa mucho espacio. ✓
   - Solución: bitmaps son altamente comprimibles.
2. ¿cómo insertamos/eliminamos una tupla? ✓
   - Solución: Run-length-encoding facilita las operaciones de update.
3. ¿cómo buscamos el bitmap que necesitamos en B^c? ✓
   - Solución: B+-tree sobre c en B^c para encontrar bitmaps.
4. ¿cómo encontramos (al final) las tuplas del resultado? ✓
   - Solución: índice sobre la posición asignada a cada tupla.

Conclusiones sobre bitmap index

Muy útil para :

- consultas con filtros complejos.
- columnas con baja cardinalidad.
- datos con poco updates (e.g. OLAP).

Deficientes:

- datos con updates frequentes.

Implementado por DB comerciales (Oracle)