
Magatzem de columnes

Compressió de dades

PID_00253059

Jordi Conesa i Caralt

M. Elena Rodríguez González

Compresión de datos

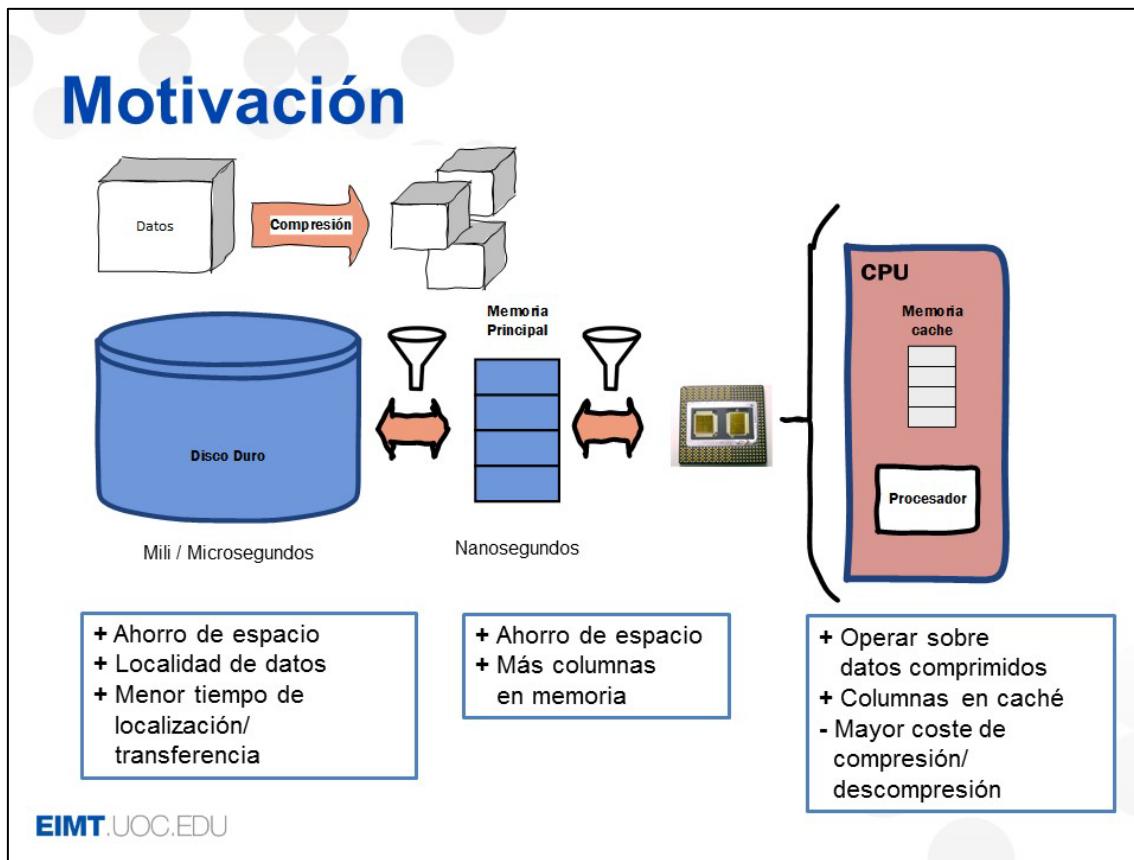
- Motivación
- Tipos de compresión de datos
- Algoritmos de compresión
- Consideraciones de diseño
- Ejemplos prácticos

EIMT.UOC.EDU

Benvinguts a la tercera presentació que tracta temes sobre els magatzems de columnes.

La presentació està dividida en 5 parts:

- Motivació: començarem veient el perquè de la compressió de dades i apuntarem les raons principals per les quals la compressió és més rellevant en els magatzems de columnes que en els magatzems de files.
- Tipus de compressió de dades: realitzarem una primera classificació dels mecanismes/algorismes de compressió de dades, en funció del seu cost i indicarem quins tipus d'algorismes són, per norma general, els més adequats en els magatzems de columnes.
- Algorismes de compressió: introduirem, mitjançant exemples, alguns dels algorismes de compressió més utilitzats (o rellevants) en els magatzems de columnes.
- Consideracions de disseny: esbossarem breument els conceptes clau a tenir en compte per a triar l'algorisme de compressió més adequat per a cada cas.
- Exemples pràctics: parlarem de la disponibilitat dels algorismes presentats en els diferents sistemes gestors de bases de dades (SGBD) que emmagatzemen les dades utilitzant magatzems de columnes.



La compressió de dades és una de les característiques principals dels magatzems de columnes. La compressió també s'usa en els magatzems de files, però de forma menys exhaustiva i amb resultats pitjors.

El motiu és que, comprimir les dades contingudes en les columnes d'una taula, sol donar millors resultats (en altres paraules, s'obté una millor ràtio de compressió) en els magatzems de columnes, ja que l'entropia de les columnes és baixa, atès que cada columna representa un mateix concepte del món real i cada columna, a més, s'emmagatzema en una estructura d'emmagatzematge (o fitxer) separada. És a dir, els diferents valors d'una mateixa columna són molt semblants entre si, comparteixen característiques (per exemple, són del mateix tipus) i no es barregen amb dades d'altres columnes que poden tenir una naturalesa diferent. Això fa que, potencialment, siguin més fàcilment comprimibles. En canvi, en el cas de les files això no passa, ja que una fila conté conjunts de valors diferents, amb tipus distints i d'una longitud potencialment diferent, és a dir, cadascun dels valors continguts en les files representa conceptes diferents del món real que tenen les seves pròpies característiques. En els magatzems de columnes, en molts casos, quan els valors d'una columna estiguin ordenats, es podrà obtenir una major compressió ja que tots els valors iguals estarán junts, és a dir, estarán emmagatzemats seqüencialment un darrere l'altre en el dispositiu d'emmagatzematge.

Algú es pot preguntar, per què és necessari comprimir les dades si avui el preu de l'emmagatzematge és molt baix? L'objectiu no és tant estalviar en el disc dur, sinó minimitzar el nombre d'operacions d'E/S, és a dir, minimitzar el nombre d'operacions de lectura i escriptura al disc. El veritable coll d'ampolla en els SGBD actuals està en els accessos al disc (i, fins i tot, en els accessos a la memòria principal). Cal tenir en compte que, en el millor dels casos (amb els discs durs més ràpids), la lectura de les dades en el disc és de l'ordre de 1.000 vegades més lenta (de l'ordre de mili o microsegons) que la lectura de les dades en la memòria principal (que és de l'ordre de nanosegons). Si comprimim una columna, reduirem el nombre de pàgines que s'han de llegir (o escriure) per a obtenir (o emmagatzemar) tots els seus valors i això estalviarà operacions d'E/S al disc, millorant així l'eficiència. A més, en alguns casos, la compressió ens permetrà emmagatzemar completament una (o més) columnes en la memòria *caché* del processador, millorant encara més el rendiment de l'SGBD.

Un altre avantatge és que, en alguns casos, és possible realitzar les operacions sobre la BD (per exemple, combinacions, operacions de selecció/filtres, etc.) directament sobre els valors comprimits. Això permetrà incrementar encara més l'eficiència del sistema, ja que es podran executar les mateixes operacions iterant sobre menys valors (ja que, com que la columna està comprimida, hi haurà potencialment menys valors a processar) en la memòria principal o en la *caché* (ja que la columna ocupa una grandària menor) i s'evitarà descomprimir les dades abans de treballar amb aquestes.

L'estalvi d'espai provocat per la compressió de les columnes sol ser aprofitat pels SGBD per a materialitzar estructures noves de dades o còpies de dades auxiliars, per exemple, per a emmagatzemar una mateixa columna diverses vegades amb ordenacions diferents, per a emmagatzemar diferents projeccions d'una taula, etc. Aquest emmagatzematge extra permet preparar millor les dades amb l'objectiu de respondre diferents consultes de forma més eficient.

L'inconvenient principal de la compressió de dades és que comprimir i descomprimir els valors d'una columna requereix temps addicional, ja que el processador (o sigui, la CPU) haurà d'invertir cicles extra per a comprimir/descomprimir les dades. Aquests cicles extra són necessaris tant en la resolució de les operacions de lectura (`SELECT`) com d'escriptura (`INSERT`, `DELETE`, `UPDATE`) sobre la BD, però seran potencialment superiors en el cas de les d'escriptura. De tota manera, i com acabem de comentar, en ocasions l'SGBD pot treballar directament sobre les dades comprimides, amb la qual cosa (en part) es mitiga aquest inconvenient.

En qualsevol cas, i a manera de resum, la compressió serà convenient quan els cicles extra del processador requerits per a la compressió/descompressió es vegin compensats per l'estalvi d'operacions d'E/S al disc. En algunes situacions, aquesta condició pot no complir-se, com seria el cas dels algoritmes de compressió més

potents, la compressió LZH, ZIP o RAR, que requereixen una càrrega de processador que pot superar l'estalvi en operacions d'E/S al disc.

Compresión de datos

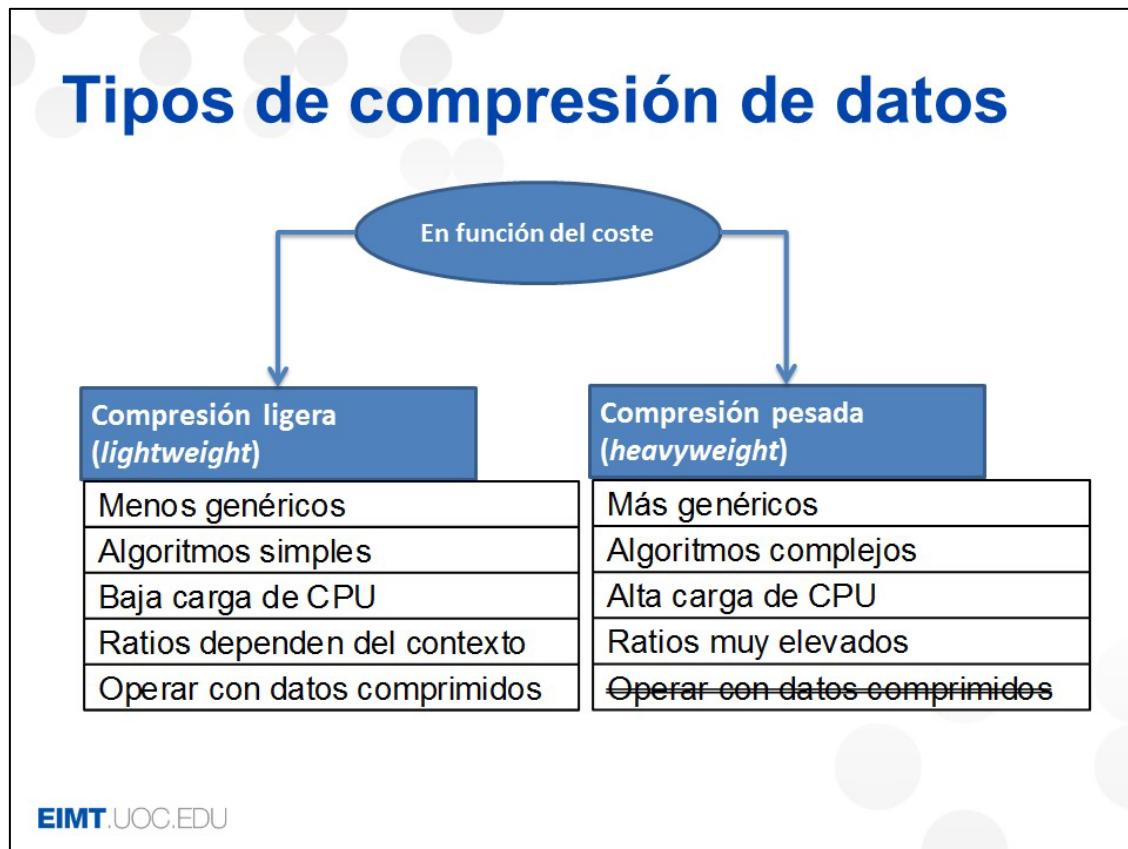
- Motivación

- Tipos de compresión de datos**

- Algoritmos de compresión
- Consideraciones de diseño
- Ejemplos prácticos

EIMT.UOC.EDU

A continuació classificarem diferents algorismes de compressió de dades en funció del seu cost computacional. Per a cada tipus d'algorisme descriurem, breument, les seves característiques principals i els seus avantatges i desavantatges quan s'apliquen sobre magatzems de columnes.



Els sistemes de compressió es poden classificar en lleugers (*lightweight*, en anglès) o pesats (en anglès, *heavyweight*), en funció del cost que implica comprimir i descomprimir les dades.

Els sistemes de compressió lleugera són aquells que utilitzen algorismes simples i, per tant, són sistemes que no requereixen una càrrega elevada de CPU (o sigui de processador) per a comprimir i descomprimir les dades. En conseqüència, la ràtio de compressió d'aquests algorismes no és tan alta com en el cas dels algorismes de compressió pesada. A més, la ràtio de compressió sol dependre molt de cada algorisme i del context en què s'apliqui. Així, alguns algorismes seran molt útils quan hi hagi valors que es repeteixen molt freqüentment, uns altres seran útils per a comprimir valors llargs o textuais i uns altres seran més adequats quan l'espai (o rang) dels valors de la columna sigui reduït. Un altre avantatge d'utilitzar la compressió lleugera és que, en alguns casos, l'SGBD pot operar directament sobre les dades comprimides, estalvant-se així el temps de descomprimir els valors, reduint les dades sobre les quals es realitzen les operacions i minimitzant el traspàs de dades al llarg de la jerarquia de la memòria (disc/memòria/caché).

Per contra, els sistemes de compressió pesats són aquells que incrementen la ràtio de compressió a costa de penalitzar el temps de compressió i descompressió. Alguns exemples són Lempel-Ziv (o LZH), ZIP o RAR. Aquests algorismes soLEN oferir elevades ràtios de compressió, independentment del context en què s'apliquin. Per tant, són la millor solució per a minimitzar la grandària de les dades. No obstant això, el seu alt

cost computacional, el fet que no es pugui descomprimir parcialment un conjunt de valors (per exemple, una pàgina de la BD que conté una part d'una columna) i el fet que no es pugui operar directament sobre les dades comprimides els fan una opció poc utilitzada en els magatzems de columnes. Encara que aquests inconvenients semblin insalvables, potser els nous canvis en maquinari permetran utilitzar aquests algorismes a curt/mitjà termini. Hi ha estudis que demostren que, per a algunes consultes, es poden utilitzar algorismes de compressió pesats, com seria el cas de l'LZH, amb resultats satisfactoris (per a més informació podeu consultar el capítol 4 de la tesi doctoral de Daniel Abadi, que és la primera referència que teniu al final d'aquesta presentació). En aquest estudi s'argumenta que, amb una bona implementació de l'algorisme LZH, es triga menys a llegir les dades comprimides i descomprimir-les del que es trigaria a llegir totes les dades no comprimides.

En aquesta material didàctic, i a causa de l'ús extensiu que tenen en els magatzems de columnes, només tractarem els algorismes de compressió lleugera.

Compresión de datos

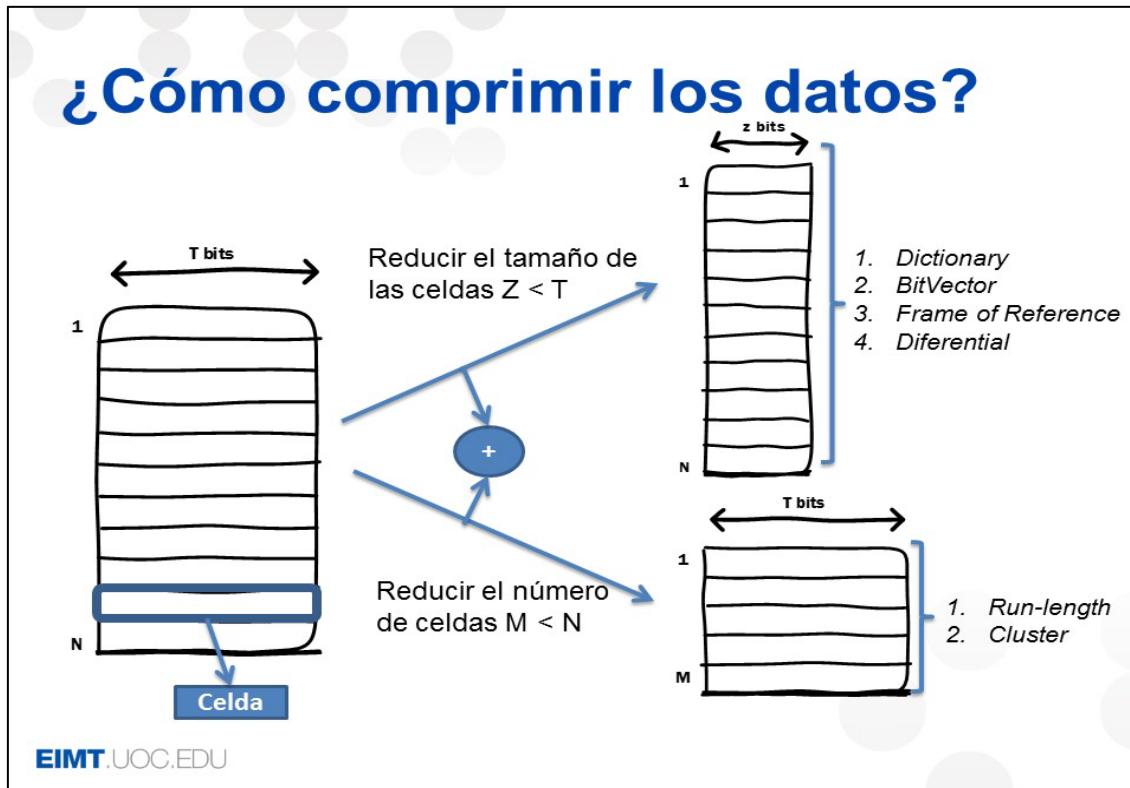
- Motivación
- Tipos de compresión de datos
- **Algoritmos de compresión**
 - Reducen el número de valores (celdas)
 - Reducen el tamaño de los valores (celdas)
- Consideraciones de diseño
- Ejemplos prácticos

EIMT.UOC.EDU

En aquest apartat presentarem, mitjançant exemples, alguns algorismes de compressió. Hem escollit els més rellevants pel seu ús o pel seu funcionament. Per a facilitar la separació dels continguts en diferents vídeos, hem classificat els algorismes en dos tipus, en funció de com comprimeixen les dades, en concret:

1. Si ho fan reduint el nombre de valors (o cel·les) d'una columna.
2. Si ho fan reduint l'espai que ocupa cada valor (o cel·la) de la columna.

Cal assenyalar que cadascun dels algorismes presentats aquí pot tenir nombroses variants i adaptacions. Per tant, abans de començar a treballar amb un magatzem de columnes, sempre serà necessari (o almenys molt aconsellable) analitzar quins algorismes proporciona l'SGBD i com els implementa.



L'espai que ocupa una columna està principalment condicionat pel nombre de valors (o cel·les) que conté (indicat mitjançant la lletra N a la diapositiva) i la grandària de cada cel·la (indicat amb la lletra T a la diapositiva). En conseqüència, la grandària total d'una columna s'obtindrà a partir de la fórmula següent (per a simplificar l'explicació i sense perdre les generalitats, hem obviat les metadades, és a dir, no considerarem els atributs que poden guardar informació de control):

$$\text{Grandària_columna} = \text{nombre_de_valors} \times \text{grandària_de_cada_valor}$$

Per a reduir la grandària d'una columna es pot reduir qualsevol dels dos operands (*nombre_de_valors* o *grandària_de_cada_valor*) que intervenen en la fórmula prèvia. És a dir, es pot reduir la grandària fent que la columna tingui menys cel·les (o sigui que emmagatzemi menys valors) o fent que cada cel·la ocipi menys espai. També es poden comprimir les dades realitzant ambdues tasques alhora, o sigui combinant diferents algorismes de compressió, per exemple, utilitzant conjuntament els algorismes *Dictionary encoding* i *Run-length encoding*, com veurem més endavant en aquesta mateixa presentació.

En aquest material, estudiarem els algorismes que comprimeixen una columna reduint el seu nombre de cel·les, com seria el cas del *Run-length encoding* i el *Cluster encoding*, o reduint la grandària de les seves cel·les, com seria el cas dels algorismes *Dictionary encoding*, *Bit-vector encoding*, el *Frame of reference encoding* i el *Differential encoding*.

Run-Length Encoding

- Reduce el número de celdas de una columna.
- Elimina valores repetidos.
- Reemplaza valores iguales que aparecen en celdas consecutivas por una tripleta con:
 - Valor
 - Posición de inicio
 - Número de valores reemplazados
- Permite obtener mejores resultados cuando la columna está ordenada.

EIMT.UOC.EDU

Aquest algorisme comprimeix les dades d'una columna eliminant gran part dels valors repetits de la mateixa. Per tant, la seva compressió es basa en la reducció del nombre de cel·les d'una columna.

En particular, l'algorisme substitueix les cel·les consecutives que tenen valors iguals per una única cel·la que conté una tripleta amb els camps següents (o metadades):

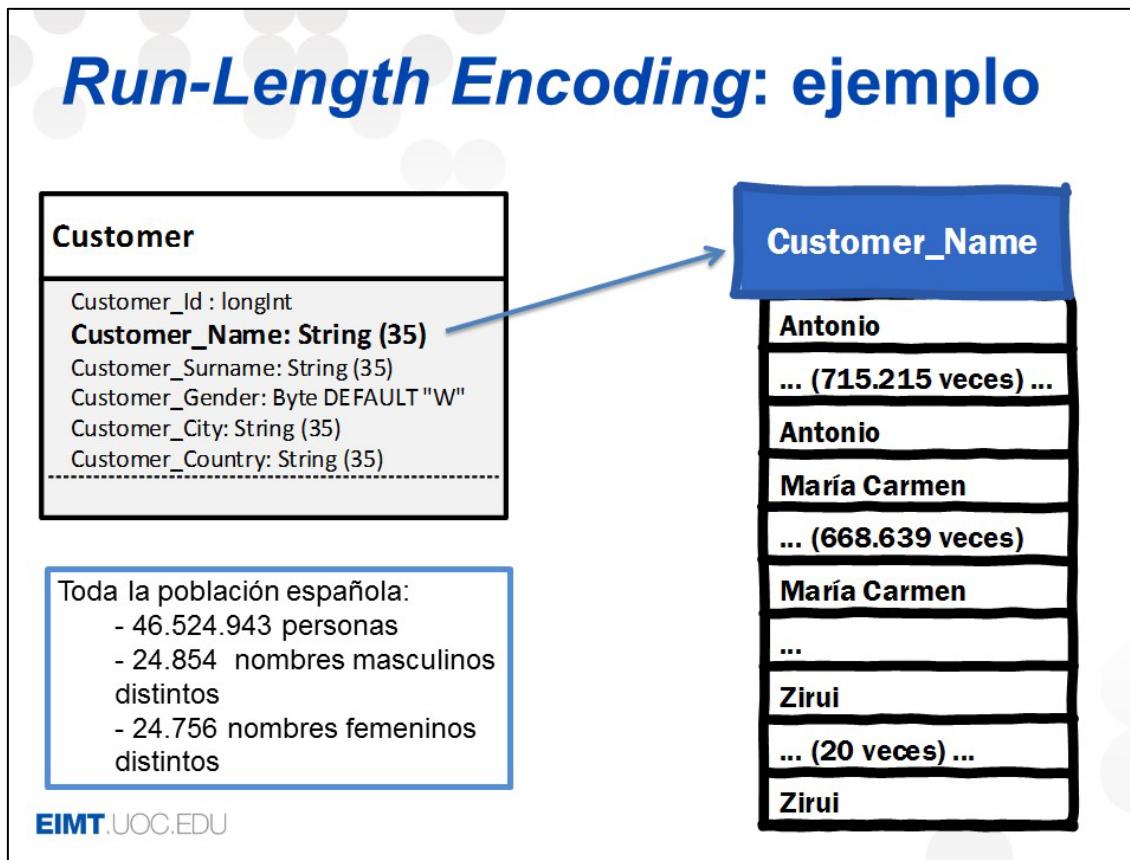
(*valor, inici, nombre_ocurrències*)

On *valor* indica el valor substituït, *inici* la posició de la primera cel·la on s'ha trobat la repetició i *nombre_ocurrències* indica el nombre de vegades que el valor estava consecutivament repetit. Hi ha variacions de l'algorisme que emmagatzemen altres tipus de metadades.

Aquest algorisme redueix el nombre de cel·les de la columna, però incrementa la grandària de les mateixes, ja que ha d'afegir les metadades *inici* i *nombre_ocurrències*.

Aquest algorisme funciona de forma eficient quan hi ha pocs valors amb un nombre elevat d'ocurrències. Com es pot intuir, l'algorisme generarà millors resultats quan els valors de la columna, a més, estiguin ordenats. El motiu és que, en aquest cas, tots els valors iguals estarán emmagatzemats en cel·les consecutives i el resultat d'utilitzar la compressió serà obtenir una nova columna amb tantes cel·les com valors diferents hi hagi.

L'algorisme es pot aplicar tant de forma aïllada, com conjuntament amb uns altres. Aquest seria el cas, per exemple, de l'algorisme *Dictionary encoding*.



Vegem com funciona l'algorisme de *Run-length encoding* mitjançant un exemple.

Recordem l'exemple que hem estat utilitzant en aquests materials dedicats dels magatzems de columnes: un petit *datamart* que conté informació de vendes d'una empresa en funció de la data de venda, els productes venuts i el client. El model de l'exemple tenia una taula de fets, la taula *Sale* i tres dimensions d'anàlisis: la data (*Date*), a qui es va vendre (*Customer*) i què es va vendre (*Product*). És important destacar que l'esquema de la BD pot haver sofert alteracions pel que fa a les presentacions prèvies, amb l'objectiu de buscar exemples més adequats al que es pretén explicar. En aquest primer exemple, comprimirem la columna *Customer_Name* de la taula *Customer*. Aquesta columna guarda el nom de pila del client, mentre que la columna *Customer_Surname* guarda el cognom (o cognoms) del client.

Per a tenir un volum significatiu de dades i mostrar la potencialitat de la compressió amb dades reals, suposarem que la nostra empresa ha realitzat vendes a tota la població espanyola. En conseqüència, la taula contindrà 46.524.943 files, una per cada persona resident a Espanya (segons dades de l'Institut Nacional d'Estadística – INE – a principis de 2016).

La columna *Customer_Name* contindrà els noms d'aquestes persones. Segons l'INE (vegeu <http://www.ine.es/daco/daco42/nombyapel/nombyapel.htm>) a Espanya hi ha un total

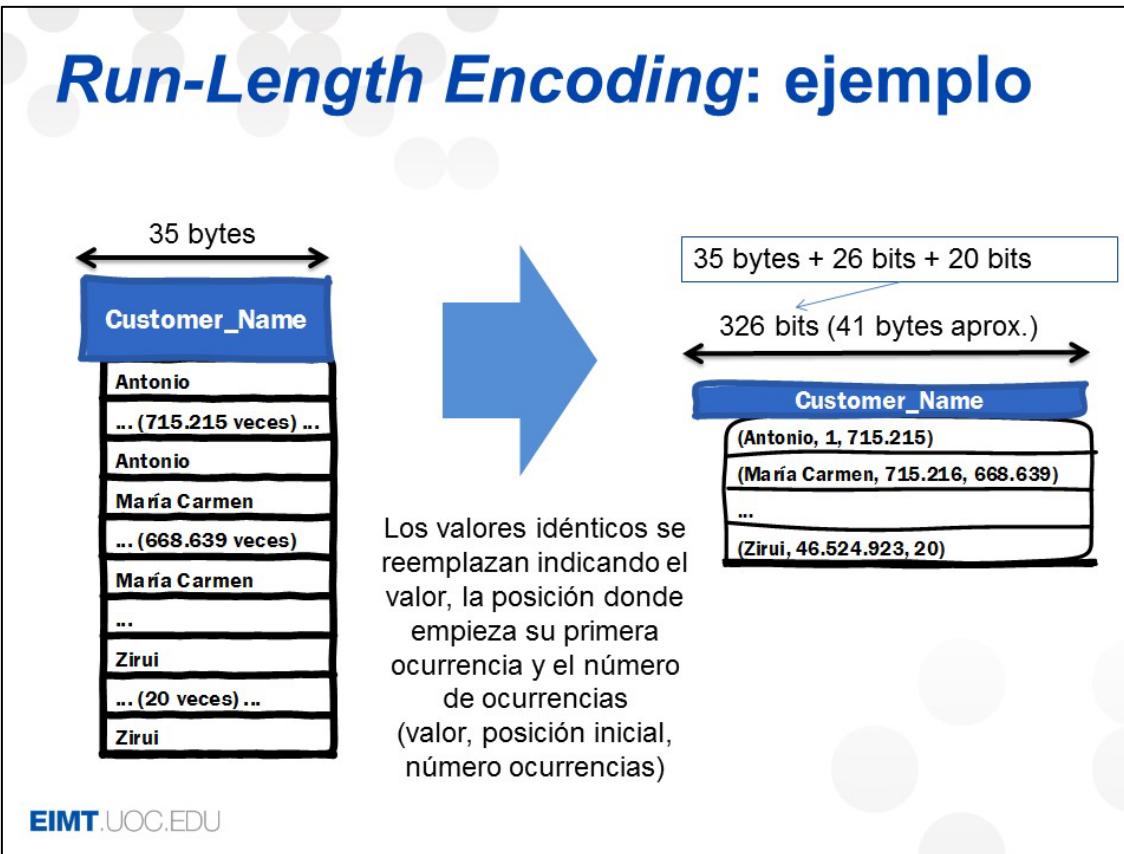
de 24.854 noms masculins i 24.756 noms femenins diferents¹. Per tant, hi haurà moltes repeticions de noms, 715.215 repetitions d'*Antonio*, 668.639 repetitions del nom *María Carmen*, 20 repetitions del nom *Zirui*, etc. També suposarem que la columna està ordenada en funció del nombre d'ocurrències dels noms dels clients².

A la transparència podem veure una representació gràfica de la columna *Customer_Name*. Per convenció, utilitzarem traços manuals (com escrits a mà) per a indicar els valors de les columnes.

Una vegada presentat el context, vegem com l'algorisme *Run-length encoding* comprimeix aquesta columna.

¹ Cal observar que només es tenen en compte noms amb una freqüència de 20 o més aparicions.

² És a dir, primer estarà *Antonio*, que és el nom que comparteixen més clients, després *María Carmen*, que és el segon més comú i, així, consecutivament. No és un ordre molt comú ni útil, però és per exemplificar millor l'algorisme.



En el context descrit, ens trobem amb el millor escenari. Com que els valors de la columna estan ordenats, totes les cel·les amb valors iguals estarán en posicions consecutives.

L'algorisme de compressió crearà una nova columna on s'emmagatzemaran els valors comprimits. L'espai de les cel·les de la nova columna haurà de ser major, ja que cada cel·la haurà d'emmagatzemar:

1. Un valor que ocupa 35 bytes.
2. La posició del valor en la columna original, que pot prendre valors entre 1 i 46.524.923. Es necessiten 26 bits per a representar aquest número.
3. El nombre d'ocurrències que, segons les dades estadístiques de l'INE, són com a màxim 715.215. Es necessiten 20 bits per a representar aquests valors.

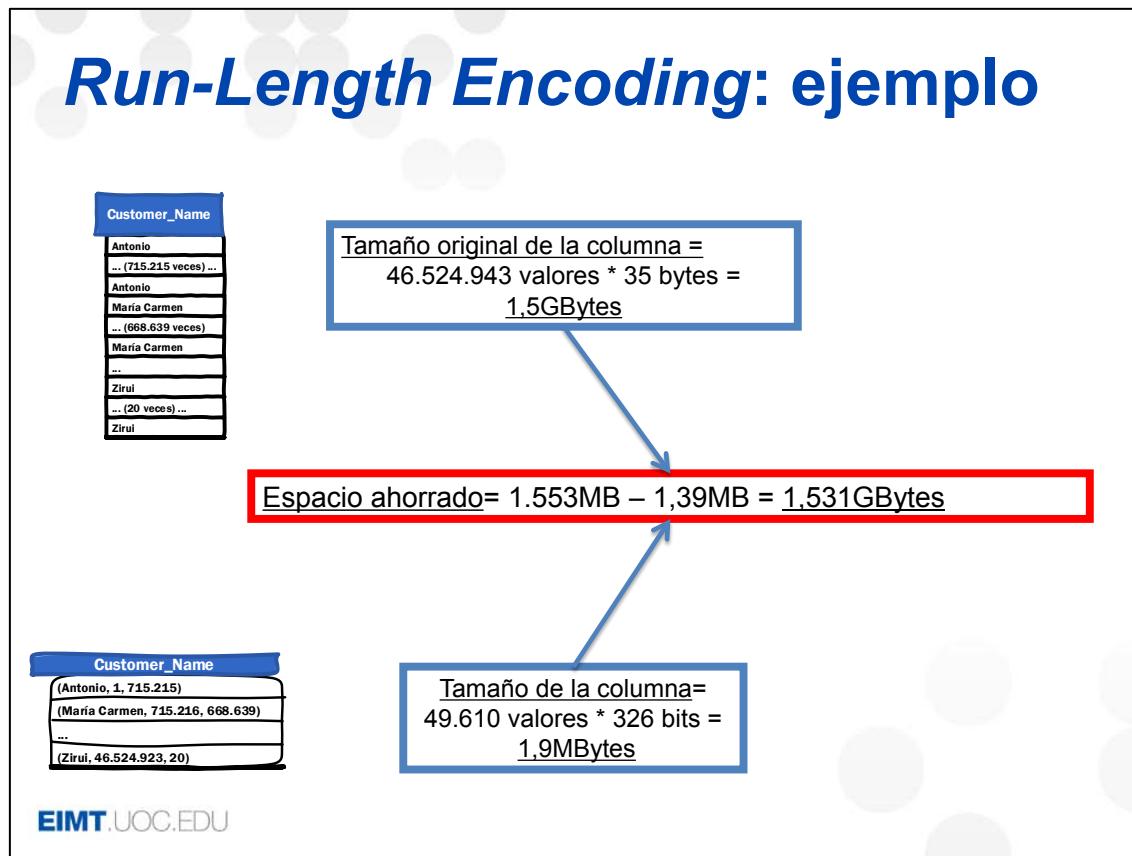
Per tant, les cel·les de la columna comprimida ocuparan 326 bits (les de la columna original n'ocupaven 280).

Per a comprimir les dades es consultarà la columna original de forma seqüencial. Per a cada cel·la de la columna (és a dir, per a cada posició i) es realitzarà el següent:

1. Es llegeix el valor de la cel·la i :

- Si la cel·la següent ($i+1$) té un valor diferent, llavors:
 - En la columna original es complirà $valor_{i+1} \neq valor_i$.
 - S'assigna a la columna comprimida la tripla ($valor_i, i, 1$).
 - La cel·la següent a llegir serà la que estigui en la posició $i+1$.
 - Si la cel·la següent ($i+1$) té el mateix valor, es consulten i es compten les cel·les consecutives i que tinguin el mateix valor (suposem que hi ha k cel·les consecutives amb el mateix valor). Llavors:
 - En la columna original es complirà $valor_{i+k} = valor_i$.
 - En la columna original es complirà $valor_{i+k+1} \neq valor_i$.
 - S'assigna a la columna comprimida la tripla ($valor_i, i, k$).
 - La cel·la següent a llegir serà la que estigui en la posició $i+k+1$.
2. Si no hem arribat al final de l'estructura que emmagatzema les dades de la columna, tornem al pas 1.

L'aplicació de l'algorisme en l'exemple de la transparència és bastant simple perquè tots els valors estan repetits i ordenats pel nombre d'ocurrències. En aquest cas, es començaria a comprimir la columna original des del principi. Com que la columna està ordenada, el valor de la primera cel·la (*Antonio*) estaria repetit 715.216 vegades (recordem que és el nombre de persones amb aquest nom segons l'INE). En conseqüència, creariem una nova tripla en la columna comprimida que tingui el valor *Antonio*, que indiqui que la primera ocurrència d'aquest valor està en la cel·la número 1 de la columna i que aquest valor es troba repetit 715.216 vegades. Veiem que l'algorisme ha permès comprimir 715.216 ocurrències d'un valor per una (1) sola ocurrència del mateix a canvi d'afegir unes quantes metades. Posteriorment, es faria el mateix amb el valor següent de la columna (*María Carmen*). En aquest cas tenim 668.639 ocurrències d'aquest nom. Totes aquestes se substituiran en la columna comprimida per una nova tripla que conté el valor *María Carmen*, la seva primera ocurrència (la 715.216, ja que ve després dels 715.216 Antonios) i el nombre d'ocurrències de la mateixa (668.639). L'algorisme aniria executant aquestes substitucions, nom a nom, fins a arribar a l'últim nom de la columna: *Zirui*, que se substituiria per la tripla (*Zirui*, 46.524.923 –posició en què apareix per primera vegada–, 20 –nombre d'ocurrències consecutives del nom–).



Com hem comentat, les cel·les de la nova columna comprimida ocupen més espai, per tant, l'estalvi d'espai d'emmagatzematge és a causa de la reducció del nombre de cel·les. A continuació vegem el resultat de la compressió en l'exemple realitzat. Veurem què ocupava la columna inicial, què ocupa la columna comprimida i quin guany (en termes d'estalvi en espai d'emmagatzematge) hem obtingut.

La grandària de la columna original és d'1,5 GBytes. Es pot calcular multiplicant el nombre de cel·les de la columna (46.524.943) per la grandària de cada cel·la (35 bytes):

$$\begin{aligned} \text{Grandària Original (Customer_Name)} &= 46.524.943 \text{ valors} \times 35 \text{ bytes} = \\ &1.628.373.005 \text{ bytes} \approx 1.553 \text{ MBytes} \end{aligned}$$

La grandària de la nova columna és de prop de 2 Mbytes. Es calcula de la mateixa forma que abans, només que ara tindrem 49.610 cel·les i cada cel·la ocuparà 326 bits.

$$\begin{aligned} \text{Grandària comprimida (Customer_Name)} &= 49.610 \text{ valors} \times 326 \text{ bits} = \\ &11.707.960 \text{ bits} \approx 1.39 \text{ MBytes} \end{aligned}$$

En conseqüència, l'estalvi final serà donat per la diferència entre la grandària inicial i el comprimit:

Estalvi (Customer_Name) = 1.553 Mbytes – 1,39 Mbytes = 1.531,61 MBytes

Podem observar que l'estalvi obtingut en aquest cas és molt bo, perquè les dades estaven ordenades i tots els valors possibles tenien repeticions. En el cas que la columna hagués contingut el nom i el cognom dels clients concatenat (tal com succeïa en la taula *Customer* usada en les presentacions prèvies), hi hagués hagut menys repetitions i els resultats haurien estat pitjors. Per tant, escollir com distribuir les dades en diferents columnes d'una taula pot tenir efectes molt significatius en el resultat (o ràtio final) de la compressió de dades.

Cluster Encoding

- Reduce el número de valores de una columna.
- Elimina valores repetidos.
- Trabaja a nivel de bloque:
 - Se distribuyen los valores de la columna en N bloques (*clusters*) de tamaño fijo.
 - Si todos los valores de un bloque son iguales se sustituyen por el valor.
 - En caso contrario, se mantiene el valor del bloque intacto.
- Útil cuando tenemos valores repetidos repartidos por distintos bloques.
- Útil en columnas que se utilizan como clave secundaria para ordenación.

EIMT.UOC.EDU

L'algorisme *Cluster encoding* comprimeix les dades d'una columna eliminant gran part dels valors repetits de la mateixa. En conseqüència, la seva compressió també es basa en la reducció del nombre de cel·les d'una columna.

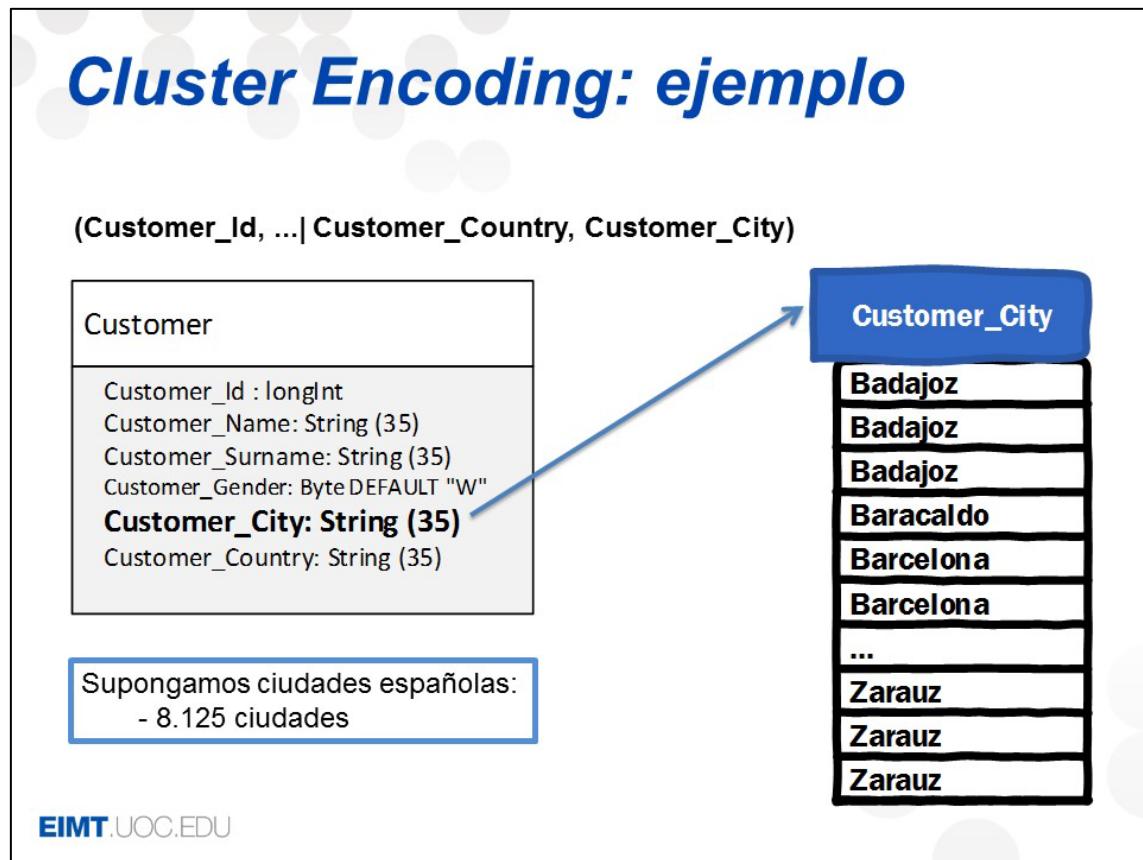
Funciona de forma similar a l'algorisme *Run-length encoding*, però treballa a nivell de clúster o blocs de cel·les, en comptes de treballar a nivell de cel·les individuals. L'algorisme divideix una columna en N blocs de la mateixa grandària. Normalment, la grandària del bloc és de 1.024 cel·les. Per a cada bloc de la columna original es comprova si tots els valors de les seves cel·les són iguals. En cas afirmatiu, se substitueix el bloc sencer per una entrada en la columna comprimida amb el valor repetit. En el cas que no totes les cel·les del bloc tinguin el mateix valor, es copia el bloc original en la columna comprimida. Per a tenir constància de quins blocs s'han comprimit i quins blocs no s'han comprimit, es guardarà un vector de bits amb tantes posicions com blocs hi hagi. A cada posició i del vector s'indicarà si el bloc i -èssim ha estat comprimit (això s'indicarà assignant un valor 1 a la posició) o no (en aquest cas s'assignarà un 0 a la posició que representa el bloc).

En aquest algorisme és important escollir bé la grandària del bloc, en funció del nombre esperat de repeticions consecutives que ens puguem trobar en la columna a comprimir. En cas contrari, el guany de la compressió pot ser poc, ja que només amb que hi hagi un valor diferent en un bloc, aquest no es comprimirà.

Aquest algorisme serà especialment útil quan les dades repetides estiguin agrupades en diferents parts de la columna. Aquest seria el cas, per exemple, que la columna fos usada com a clau secundària en una ordenació. Recordem que les taules d'un magatzem de columnes poden definir les claus d'ordenació formades per una o diverses columnes i que, en ocasions (en el cas que es permetin definir projeccions), diverses columnes de la taula es poden agrupar i reordenar els seus valors d'acord amb altres claus d'ordenació. En el cas que la clau d'ordenació inclogui diverses columnes, la primera columna és la clau primària de l'ordenació, mentre que la resta de columnes són les claus secundàries d'ordenació.

Suposem que tenim una columna que determina el país dels clients i una altra que determina la seva ciutat (serien les columnes *Customer_City* i *Customer_Country* de la nostra taula *Customer*). Si ordenem les dades en primera instància per país (aquesta columna seria la clau primària en l'ordenació) i després per ciutat (aquesta columna seria la clau secundària en l'ordenació), veurem que totes les ciutats del mateix país apareixeran consecutives en la columna *Customer_City*. Per exemple, el valor *Madrid* apareixerà consecutiu a tots els clients espanyols, però també apareixerà consecutiu en una altra part de la columna per als clients de Colòmbia, ja que en aquest país també hi ha una ciutat denominada Madrid.

L'algorisme es pot aplicar tant de forma aïllada com conjuntament amb uns altres, com seria el cas, per exemple, de l'algorisme *Dictionary encoding*.

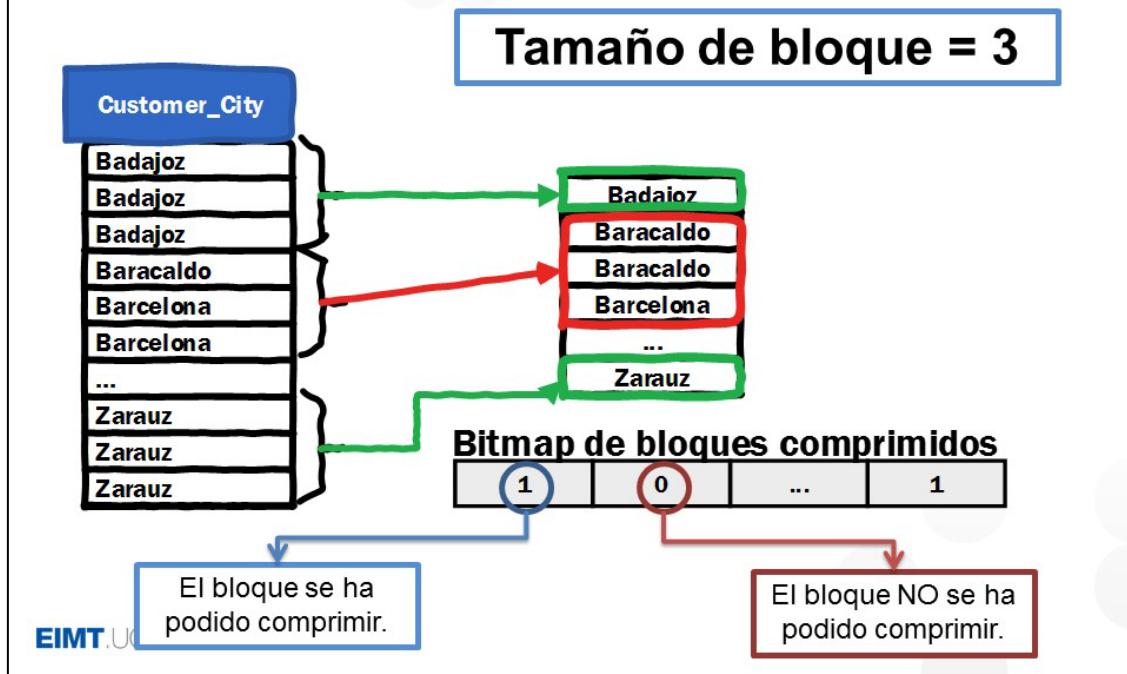


Seguidament veurem com funciona l'algorisme *Cluster encoding* a través d'un exemple.

Continuem amb la taula *Customer* utilitzada en l'algorisme anterior, que recordem té 46.524.943 files. La columna *Customer_City* contindrà els noms de les ciutats on viuen els clients. A Espanya hi ha 8.125 ciutats, en conseqüència la columna podrà tenir 8.125 valors possibles. També suposem, per a aquest exemple, que la taula està ordenada per les columnes *Customer_Country* i *Customer_City* –és a dir, el parell (*Customer_Country*, *Customer_City*) és la clau d'ordenació de la taula *Customer*–.

A la transparència podem veure una representació gràfica i simplificada de la columna *Customer_City*. Una vegada presentat el context, vegem com l'algorisme *Cluster encoding* comprimeix aquesta columna.

Cluster Encoding: ejemplo



Per a facilitar l'explicació d'aquest exemple, assumirem que s'ha escollit una grandària de bloc de 3 posicions. Això, com hem comentat, és poc realista, ja que els valors d'un bloc acostumen a ser de l'ordre de 1.024 elements o superior.

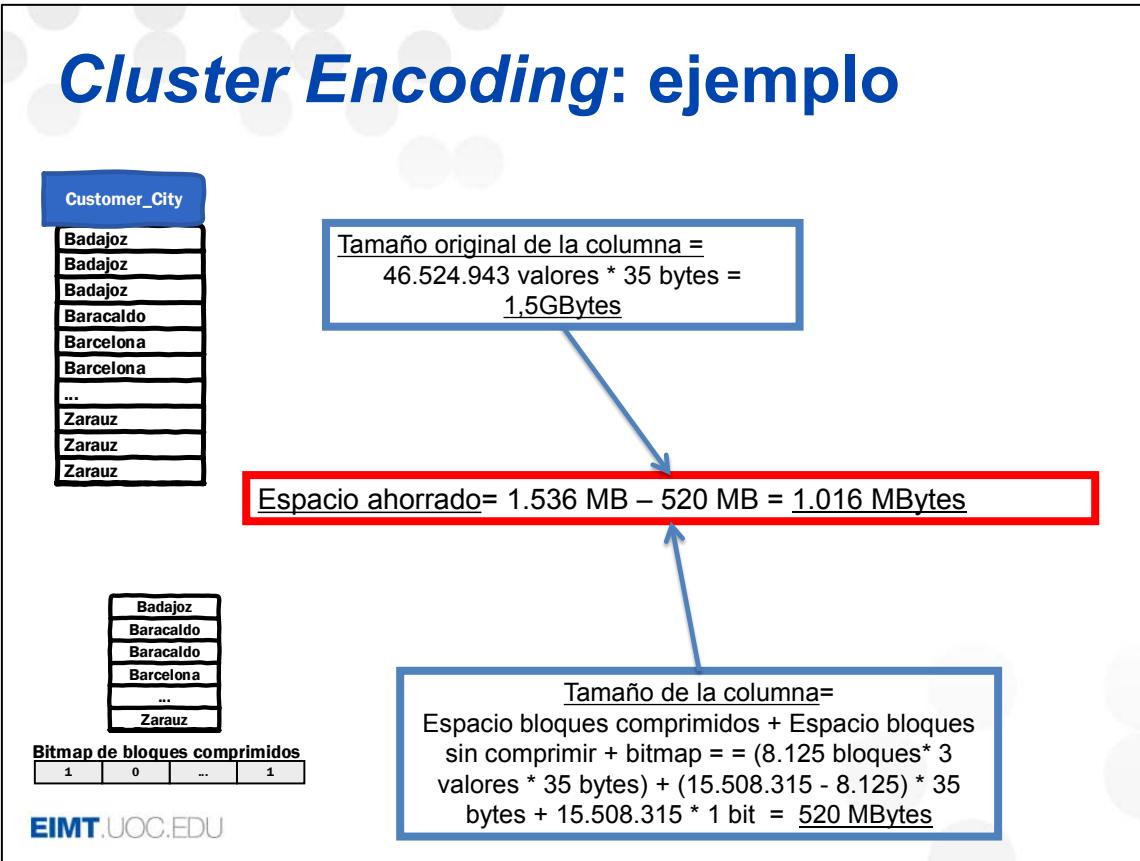
L'algorisme de compressió crearà una nova columna on s'emmagatzemaran els valors comprimits i un vector de bits que indicarà quins blocs s'han pogut comprimir. En aquest cas, l'espai de les cel·les de la nova columna serà igual al de la columna original ja que no s'han d'emmagatzemar metadades extra (és a dir, informació de control) per a cada valor.

L'algorisme classificarà la columna original en N blocs de grandària 3. Després, per a cada bloc de la taula (posició i), es realitzarà la pregunta següent: tots els valors de les cel·les del bloc i són iguals?

- En cas afirmatiu:
 - S'assigna a la columna comprimida un sol valor: el valor repetit.
 - S'assigna un 1 en la posició i -èssima del vector de bits per a indicar que el bloc i s'ha comprimit.
- En cas negatiu:

- S'assigna a la columna comprimida el bloc sencer, les tres cel·les de la columna original.
- S'assigna un 0 en la posició i -èssima del vector de bits per a indicar que el bloc i no s'ha comprimit.

En l'exemple de la transparència, es classificaria la columna en blocs de 3 cel·les. El primer bloc estaria format per 3 cel·les amb el mateix valor: *Badajoz*. Com que té el mateix valor, s'escriuria en la columna comprimida un sol valor de *Badajoz* i s'assignaria un 1 a la posició 1 del vector de bits per a indicar que el primer bloc ha estat comprimit. El segon bloc de la columna no pot ser comprimit, perquè conté valors diferents (*Barcelona* i *Baracaldo*). En aquest cas es copiaria el bloc sencer en la columna comprimida i s'indicaria en el vector de bits que no s'ha pogut comprimir (un 0 en la posició 2 del vector). Continuaríem aplicant l'algorisme fins al final. Fixeu-vos que l'últim bloc es podria comprimir perquè totes les seves cel·les comparteixen el mateix valor (*Zarauz*), substituint el bloc sencer pel valor *Zarauz* en la columna comprimida i indicant-ho inserint un 1 en l'última posició del vector de bits.



Com hem comentat, s'ha escollit una grandària de bloc de 3. Tenint en compte que la taula *Customer* conté 46.524.943 files, tindrem 15 milions de blocs:

$$\text{Nombre de blocs (Customer_City)} = 46.524.943 \text{ valors} / 3 \text{ cel·les} = 15.508.315 \text{ blocs}$$

Les cel·les de la nova columna comprimida ocupen el mateix espai que el de la columna original. Per tant, l'estalvi haurà de venir per la reducció en el nombre de cel·les. A continuació vegem el resultat de la compressió per al nostre exemple. Veurem què ocupava la columna inicial, què ocupa la columna comprimida i quin guany (en termes d'estalvi d'espai d'emmagatzematge) hem obtingut.

La grandària de la columna original és d'1,5 GBytes. Es pot calcular multiplicant el nombre de cel·les de la columna (46.524.943) per la grandària de cada cel·la (35 bytes):

$$\begin{aligned} \text{Grandària original (Customer_City)} &= 46.524.943 \text{ valors} \times 35 \text{ bytes} = \\ &1.628.373.005 \text{ bytes} \approx 1.553 \text{ MBytes} \end{aligned}$$

Per a calcular la grandària de la nova columna s'haurà d'estimar el nombre de blocs comprimits i el nombre de blocs no comprimits. Tenint en compte que la taula *Customer* emmagatzema només informació de clients d'Espanya i que està ordenada per país i ciutat (columnes *Customer_Country* i *Customer_City*), sabem que tots els

valors repetits estaran en cel·les contigües i que en total hi ha 8.125 valors diferents (és a dir, hi ha 8.125 ciutats diferents). En conseqüència, podem concloure que, en el pitjor dels casos, hi haurà 8.125 blocs sense comprimir. Una vegada feta aquesta estimació, podem calcular quina serà la grandària de la columna comprimida (d'uns 520 MBytes) mitjançant la fórmula que es mostra a continuació.

$$\begin{aligned}
 \text{Grandària comprimida (Customer_City)} &= (\text{Nombre de blocs NO comprimits} \times \\
 &\quad \text{Nombre de cel·les per bloc} \times \text{Grandària de cel·la}) + (\text{Nombre de blocs comprimits} \times \\
 &\quad \text{Grandària de cel·la}) + \text{Grandària vector de bits} = (8.125 \text{ blocs}^* 3 \text{ valors}^* 35 \text{ bytes}) + \\
 &\quad (15.508.315 \text{ blocs comprimits} - 8.125 \text{ blocs sense comprimir})^* 35 \text{ bytes} + 15.508.315 \\
 &\quad \text{posicions}^* 1 \text{ bit} \\
 &\approx 520 \text{ MBytes}
 \end{aligned}$$

Finalment, l'estalvi a l'espai d'emmagatzematge serà donat per la diferència entre la grandària inicial i el comprimit:

$$\text{Estalvi (Customer_City)} = 1.536 \text{ MB} - 520 \text{ MB} = 1.016 \text{ MBytes}$$

Podem observar que l'estalvi obtingut en el nostre exemple és molt bo, perquè les dades estaven ordenades i tots els valors possibles tenien repeticions. No obstant això, a causa que els valors repetits no estaven distribuïts al llarg de la columna, sinó que es trobaven junts, aplicar un algorisme com ara *Run-length encoding* hagués estat més eficient. Es deixa com a tasca al lector realitzar els càlculs per a veure quina millora s'obtindria amb l'algorisme *Run-length encoding* i que ho compari amb l'estalvi actual.

Compresión de datos

- Motivación
- Tipos de compresión de datos
- **Algoritmos de compresión**
 - Reducen el número de valores (celdas)
 - **Reducen el tamaño de los valores (celdas)**
- Consideraciones de diseño
- Ejemplos prácticos

EIMT.UOC.EDU

Una vegada vists els algorismes que redueixen el nombre de valors (o cel·les) d'una columna, ens centrarem en els que comprimeixen les dades reduint la grandària de cada valor (o cel·la) de la columna, és a dir, en els algorismes que redueixen la grandària necessària per a emmagatzemar cada valor d'una columna.

Dictionary Encoding

- Muy útil para comprimir columnas de tipo *string* o de tamaño considerable.
- Funcionamiento:
 - Crea un diccionario con los valores de la columna.
 - Sustituye los valores de la columna por referencias a su valor.
- Comprime los datos de valores individuales, no de conjuntos de valores:
 - La ordenación de la columna no tiene ningún impacto en la eficiencia de la compresión.
- Puede aplicarse en combinación con otros algoritmos.
- Tiene distintas variantes (diccionario ordenado, diccionarios por página, etc.).

EIMT.UOC.EDU

L'algorisme *Dictionary encoding* comprimeix les dades d'una columna reduint la grandària necessària per a representar els seus valors. La seva compressió no es basa en la reducció del nombre de cel·les d'una columna, sinó en la reducció de la grandària de cada cel·la. En conseqüència, si hi ha 100 valors iguals consecutius en una columna, l'algorisme generarà 100 entrades, una per a cada valor individual. Per això, tenir la columna ordenada no implicarà cap millora en aquest algorisme de compressió.

És un algorisme que resulta molt útil per a comprimir *strings* i columnes que tinguin associat un tipus de dades d'una grandària significativa. En el cas de columnes de gran grandària com seria el cas, per exemple, de columnes amb un tipus de dades associat BLOB, aquest algorisme pot no ser necessari, atès que, com ja sabem, l'SGBD sol guardar aquests objectes grans en estructures externes separades, de forma semblant a com es faria mitjançant aquest algorisme.

Per a comprimir les dades, l'algorisme crea un diccionari que conté els valors diferents que apareixen en la columna. Una vegada fet això, es comprimeix la columna substituint el valor de cadascuna de les seves cel·les per l'índex de la posició del diccionari que conté aquest valor. El guany està en el fet que s'ha convertit el valor original, que potencialment ocupava diversos bytes, en un nombre natural que es pot representar utilitzant molt pocs bits. Normalment el diccionari s'emmagatzema en una pàgina diferent del disc. L'estalvi apareix en els valors repetits, ja que abans

repetíem un valor que tenia un cost de *bytes* elevat múltiples vegades i ara només repetim un índex que té un cost de bits molt menor. D'altra banda, i com a aspecte potencialment negatiu, s'està afegint una nova redirecció (o indirecció) per a consultar les dades, cosa que pot fer inviable l'algorisme si el guany no és molt bo i també depenen del tipus de consultes que s'hagin de realitzar.

Aquest algorisme és un dels més utilitzats en l'actualitat. Es pot aplicar conjuntament amb altres algoritmes que redueixen el nombre de cel·les, com són els dos algoritmes de compressió que hem vist anteriorment, els algoritmes *Run-length encoding* i *Cluster encoding*.

L'algorisme té diferents variants. Una d'aquestes variants consisteix, per exemple, en emmagatzemar el diccionari de forma ordenada (això pot ser molt útil per a realitzar lectures seqüencials per valor) o bé per a definir un diccionari per a cada pàgina de la BD.

Dictionary Encoding: ejemplo

- Supongamos la tabla *Customer* con todos los habitantes de España.
- Según el Instituto nacional de estadística, en España (2016) existen:
 - 46.524.943 habitantes
 - 24.854 nombres masculinos
 - 24.756 nombres femeninos

Customer	
Customer_Id : longInt	
Customer_Name: String (35)	
Customer_Surname: String (35)	
Customer_Gender: Byte DEFAULT "W"	
Customer_City: String (35)	
Customer_Country: String (35)	

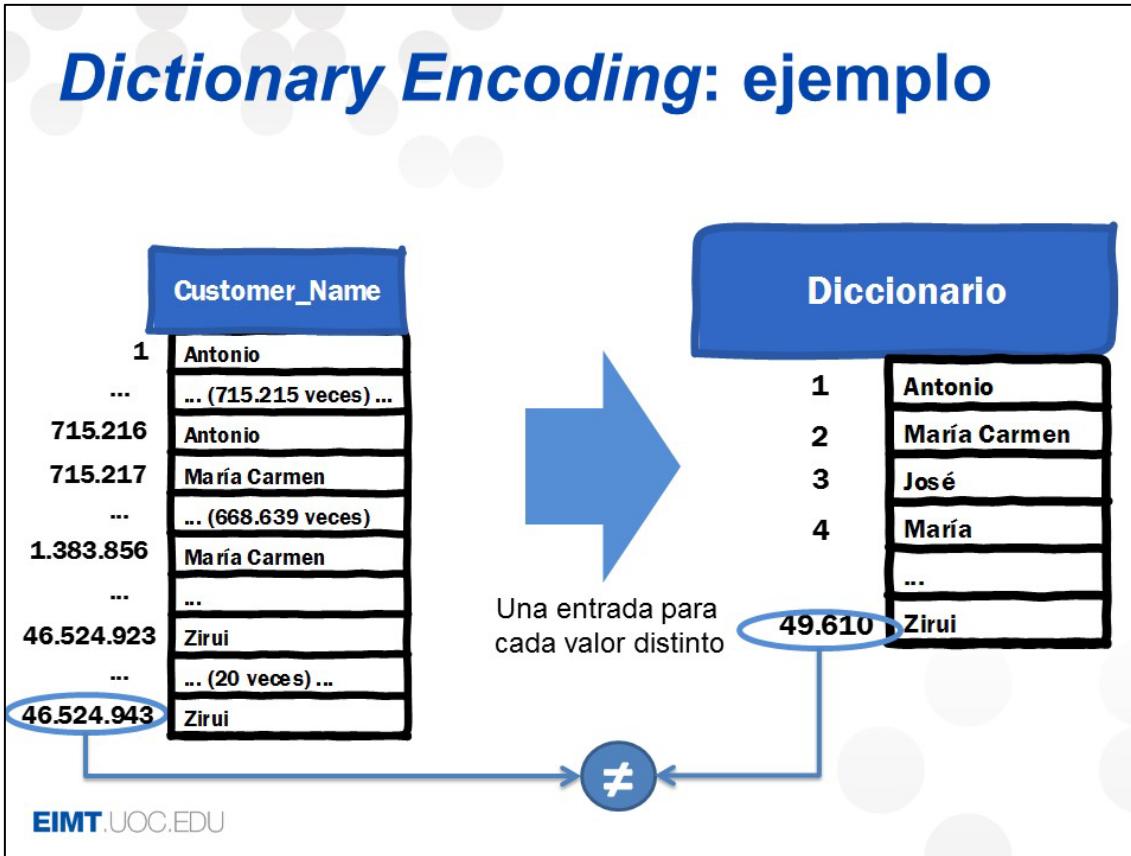
EIMT.UOC.EDU

Vegem com funciona aquest algorisme. Per a això, utilitzarem el mateix exemple que hem usat per a l'algorisme *Run-length encoding*. Recordem-lo:

En aquest exemple, comprimirem la columna *Customer_Name* de la taula *Customer*.

Suposarem que l'empresa ha realitzat vendes a tota la població espanyola. Per tant, la taula *Customer* contindrà 46.524.943 files, una per a cada persona resident a Espanya. La columna *Customer_Name* contindrà els noms d'aquestes persones. Segons l'INE³ a Espanya hi ha un total de 24.854 noms masculins i 24.756 noms femenins diferents. Per això hi haurà moltes repeticions de noms, 715.215 repeticions d'Antonio, 668.639 repeticions del nom *Maria Carmen*, 20 repeticions del nom *Zirui*, etc. També suposarem que la columna està ordenada en funció del nombre d'ocurrències dels noms dels clients.

³ Vegeu <http://www.ine.es/daco/daco42/nombyapel/nombyapel.htm>



Una vegada presentat el context, vegem com aquest algorisme comprimeix la columna *Customer_Name*.

L'algorisme realitza dues accions per a comprimir les dades, en concret:

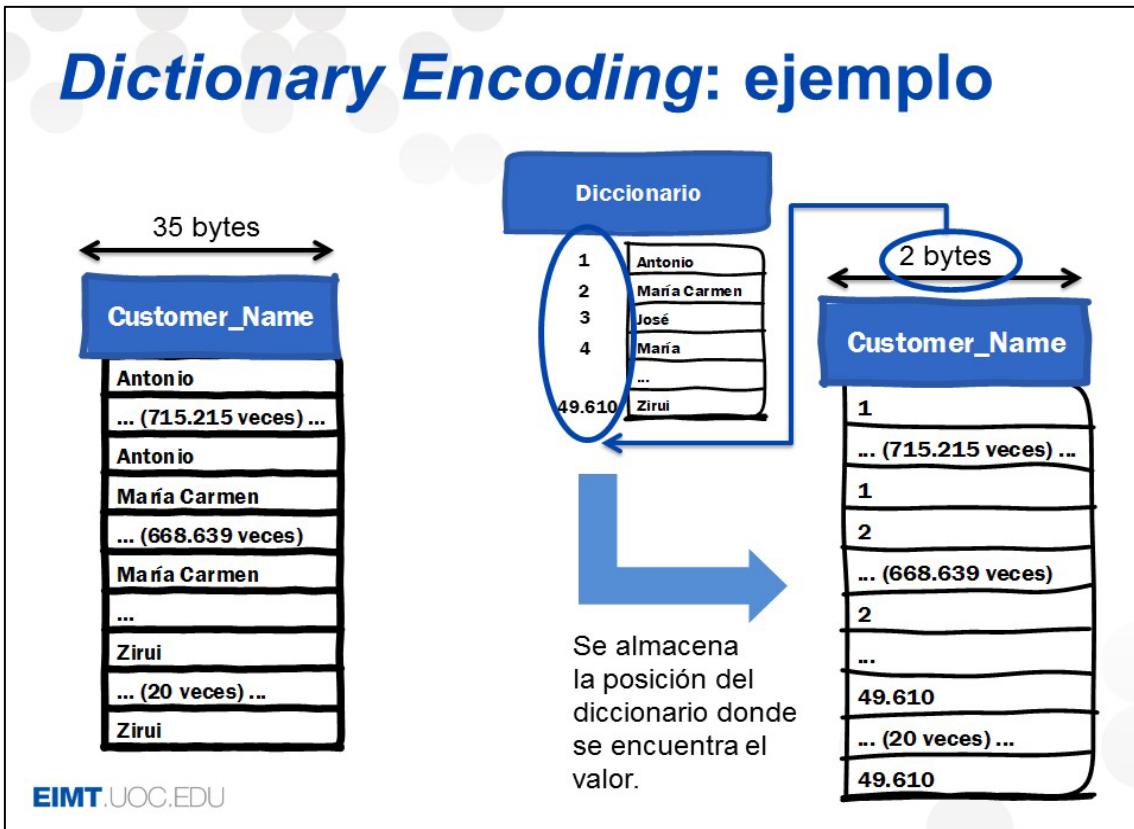
1. La creació d'un diccionari amb els valors de la columna.
2. La compressió de la columna substituint els valors originals per índexs que apunten a la ubicació d'aquests valors en el diccionari.

Ambdues accions es poden realitzar alhora, però en aquesta presentació les efectuarem de forma seqüencial per a una major claredat.

El primer que es farà és crear el diccionari. Per a això, cal saber la grandària que ha de tenir cada entrada del diccionari, així com el nombre d'entrades que haurà de tenir. La grandària de cada entrada del diccionari serà de 35 bytes ja que ha d'emmagatzemar noms de clients. D'altra banda, el diccionari ha d'emmagatzemar tots els valors que apareixen en la columna. Segons les dades ofertes per l'INE, tindrem 49.610 valors possibles, per tant el diccionari tindrà 49.610 entrades. Saber el nombre d'entrades del diccionari és important, perquè condicionarà la grandària de les cel·les de la columna comprimida. En particular, en el cas que ens ocupa, necessitem 16 bits (2 bytes) per a encaminar fins a 65.000 valors. En conseqüència, la nova columna comprimida contindrà cel·les de 2 bytes.

Una vegada creat el diccionari, es poblarà amb el contingut de la columna. Per a això, s'iterarà sobre totes les cel·les de la columna. Per a cada cel·la, es comprovarà si el seu valor ja està emmagatzemat en el diccionari. Si no fos així, s'afegiria el seu valor en el diccionari. La manera en què s'emmagatzemen les dades en el diccionari pot ser diferent. Es podrien emmagatzemar els valors de forma ordenada, per exemple. Sobre el nostre cas d'exemple concret, per a simplificar, s'afegiran en ordre d'ocurrència.

Fixeu-vos que, gràcies al diccionari, hem pogut reduir les repeticions de valors, ja que hem passat de la representació de 46 milions de valors a la representació de prop de 50 mil. El guany d'aquest algorisme vindrà en el cas que els valors ocupin molt més que els índexs necessaris per a accedir al diccionari, com succeeix en el cas del nostre exemple.

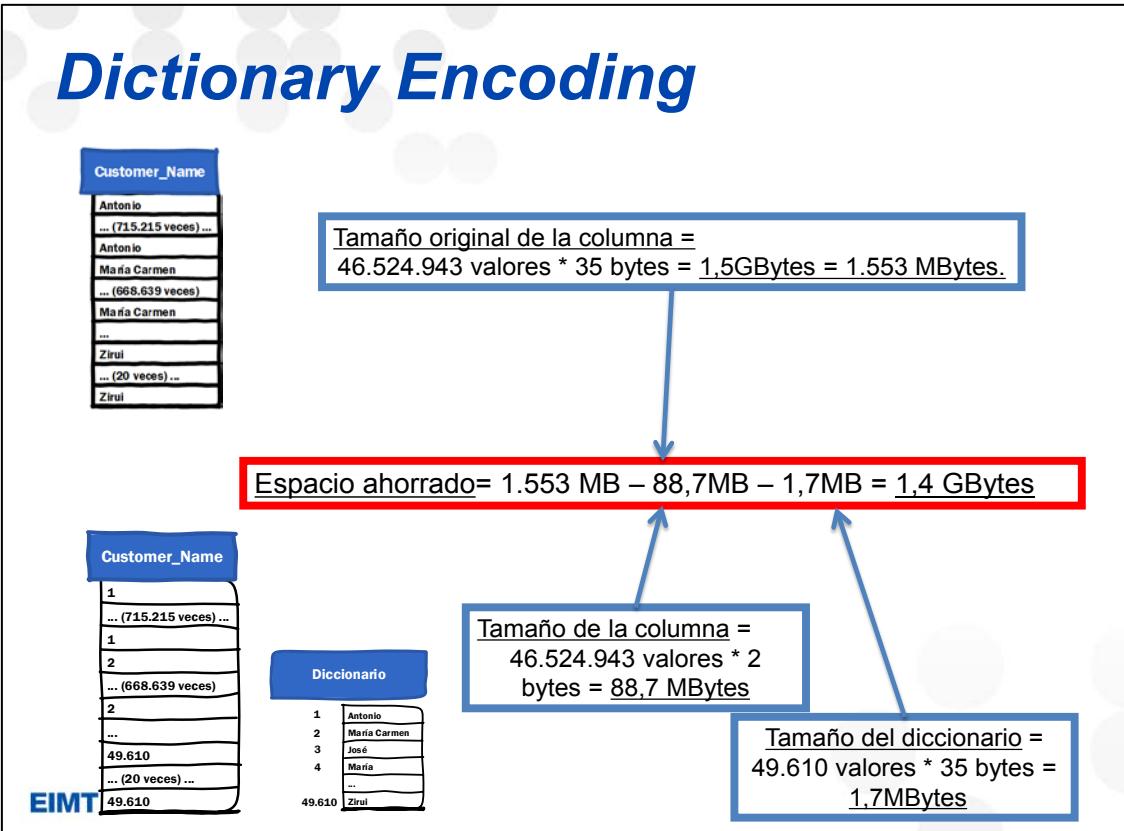


Una vegada creat el diccionari es podrà procedir a la compressió dels valors de la columna. Per a fer-ho, s'iterarà sobre la columna original i, per a cada cel·la (posició *i*-èssima) de la mateixa, es farà el següent:

- Es buscarà el seu valor en el diccionari.
- S'obtindrà l'índex (el nombre d'entrada) del diccionari on es troba el valor.
- S'afegirà aquest índex a la columna comprimida.

En el cas que ens ocupa, per a la primera cel·la, es consultaria en el diccionari el seu valor (*Antonio*). Com que *Antonio* està emmagatzemat en la primera posició del diccionari, el seu valor se substituiria pel número 1 en la columna comprimida. Això es faria per a les 715.215 cel·les següents, en què el valor d'*Antonio* es repeteix. En la cel·la 715.216, es consultaria el valor de *María Carmen* en el diccionari, i s'obtindria la seva posició (la número 2). Per tant, se substituiria el valor *María Carmen* pel número 2 en la columna comprimida. Això es faria per a totes les seves repeticions i, així, successivament per a la resta de valors de la columna i les seves ocurrències.

Al final, la columna comprimida tindrà el mateix nombre de cel·les que la columna original i el mateix nombre de repeticions. La diferència rau en el fet que s'ha canviat el tipus de la mateixa per un tipus numèric que ocupa 2 bytes, molt menys que els 35 bytes que ocupaven els noms.



A continuació vegem el resultat de la compressió en l'exemple realitzat. Veurem què ocupava la columna inicial, què ocupa la columna comprimida i quin guany (en termes d'estalvi en espai d'emmagatzematge) hem obtingut.

La grandària de la columna original és d'1,5 GBytes. Es pot calcular multiplicant el nombre de cel·les de la columna (46.524.943) per la grandària de cada cel·la (35 bytes):

$$\text{Grandària Original (Customer_Name)} = 46.524.943 \text{ valors} \times 35 \text{ bytes} = \\ 1.628.373.005 \text{ bytes} \approx 1.553 \text{ MBytes}$$

La grandària de la nova columna serà donat per la grandària de la columna més la grandària del diccionari. Aquestes grandàries es poden calcular tal com s'indica a continuació.

$$\text{Grandària diccionari (Customer_Name)} = 49.610 \text{ valors} \times 35 \text{ bytes} = \\ 1.736.350 \text{ bytes} \approx 1,7 \text{ MBytes}$$

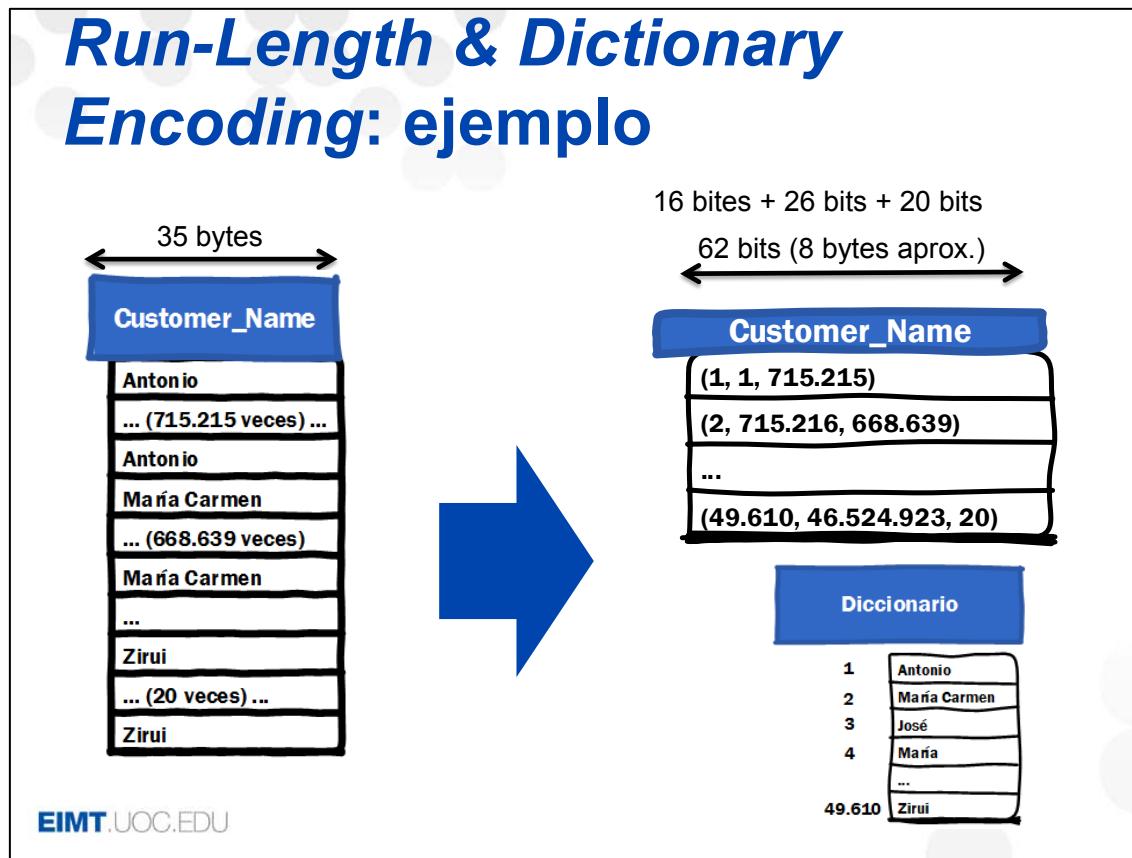
$$\text{Grandària comprimida (Customer_Name)} = 46.524.943 \text{ valors} \times 2 \text{ bytes} = \\ 93.049.886 \text{ bytes} \approx 88,7 \text{ MBytes}$$

Per tant, el valor final de la columna comprimida serà de 90,4 MBytes. L'estalvi final serà donat per la diferència entre la grandària inicial i el comprimit:

$$\text{Estalvi}(\text{Customer_Name}) = 1.553 \text{ MBytes} - 1,7 \text{ MBytes} - 88,7 \text{ MBytes} = 1.462,6 \text{ MBytes}$$

Podem observar que l'estalvi obtingut en aquest cas és molt bo, perquè hi havia moltes repeticions i les dades originals eren molt majors que els índexs necessaris per a accedir al diccionari. No obstant això, la columna resultant encara té moltes repeticions. En altres paraules, la columna pot ser susceptible de ser comprimida amb algorismes que redueixin el nombre de cel·les d'una columna per a aconseguir una millor ràtio de compressió.

A la transparència següent, mostrarem com es pot complementar la compressió realitzada amb l'algorisme *Dictionary encoding* amb l'algorisme *Run-length encoding* per a reduir encara més la grandària de la columna *Customer_Name* de la nostra taula d'exemple *Customer*.



Tal com acabem de comentar, la columna *Customer_Name* es podria comprimir combinant els algorismes *Run-length encoding* i *Dictionary encoding*. En el cas que ho féssim, ens trobaríem amb un resultat com el que es mostra a la diapositiva.

Per a això, es crearia un diccionari que emmagatzemaria els noms de les persones (que són de 35 bytes), que tindria la mateixa grandària i característiques que hem vist a les transparències anteriors. Posteriorment, es procediria a compactar les cel·les amb valors repetits, utilitzant l'algorisme *Run-length encoding*. La grandària de les cel·les de la columna comprimida en aquest cas seria de 62 bits (16 bits per a representar l'entrada del diccionari que conté el valor, 20 bits per a mostrar la posició de la cel·la on apareix el valor i 26 bits per a mostrar el nombre de repetitions consecutives).

Una vegada calculada la grandària de la nova columna, la compressió s'efectuaria de la mateixa forma que hem vist en l'apartat d'aquesta presentació dedicat a l'algorisme *Run-length encoding*, però tenint en compte que ara el valor no és de tipus *string*, sinó de tipus numèric. En aquest cas, la primera entrada de la columna comprimida no tindria el valor (*Antonio*, 1, 715.215) com abans, sinó (1, 1, 715.215). Veiem com s'ha canviat el valor d'*Antonio* per la posició d'*Antonio* en el diccionari; aquesta posició és el número 1.

Deixem com a exercici per al lector comprovar el guany realitzant la compressió mostrada en aquesta transparència. L'anitem al fet que, una vegada calculada, la

compari amb els guanys d'aplicar el *Dictionary encoding* i el *Run-length encoding* de forma aïllada i reflexioni sobre els resultats obtinguts.

Bit-Vector Encoding

- Útil cuando tenemos pocos valores posibles (sexo, categoría, etc.).
- Se crean M vectores de bits de N posiciones:
 - M es el número de valores posibles.
 - Cada vector tendrá tantas posiciones como celdas tenga la columna.
 - En la posición i -esima del vector j se asigna un 1 si el valor i -esimo de la columna es j .
- Permite representar múltiples valores para cada fila.
- Los vectores de bits se podrían comprimir mejorar la compresión.

EIMT.UOC.EDU

L'algorisme *Bit-vector encoding* comprimeix les dades d'una columna reduint la grandària necessària per a representar els seus valors. Igual que en el cas del *Dictionary encoding*, la seva compressió es basa en la reducció de la grandària necessària per a emmagatzemar els valors de cada cel·la, és a dir, no redueix la grandària en funció de les repeticions dels valors, ni tindrà un millor guany si les dades estan ordenades.

L'algorisme és molt útil quan una columna té pocs valors possibles, com seria el cas del gènere d'una persona, la categoria d'un producte, el tipus de client, etc.

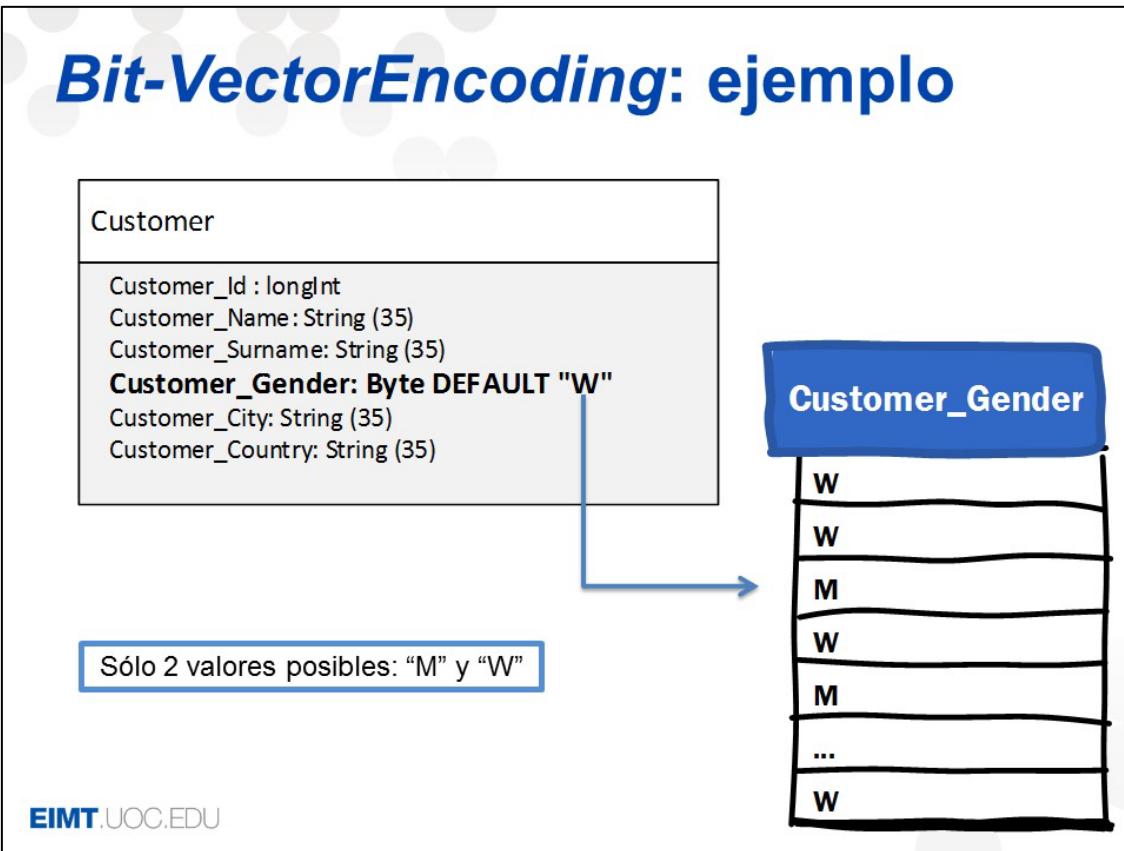
Per a comprimir les dades, es crea un vector de bits per a cada valor possible que pugui prendre la columna. Aquest vector de bits tindrà tantes posicions com cel·les tingui la columna original. Cada posició i del vector conté un bit que indica si la cel·la en la posició i té assignat aquest valor. Apart de les millores que es puguin obtenir en termes de compressió de dades, treballar amb vectors de bits és molt eficient i convenient per a realitzar algunes operacions sobre la BD (com per exemple realitzar operacions booleanes sobre un conjunt de columnes o comprovar que les files tinguin un valor determinat) i, per tant, pot representar un guany considerable en el seu rendiment. A més, l'ús del vector de bits deixa representar columnes que permeten valors múltiples, per exemple, una columna que representa el conjunt de *keywords* que descriuen un producte. Encara que (almenys des del punt de vista teòric) una BD basada en el model relacional no permet definir columnes multiavaluades, hem de

tenir en compte que hi ha altres BD basades en models de dades diferents que sí que ho permeten (per exemple, les BD NoSQL).

Com a aspecte negatiu, cal comentar que es requereix crear tants vectors de bits com valors pugui tenir una cel·la. Això pot ser inviable en funció del nombre de valors possibles.

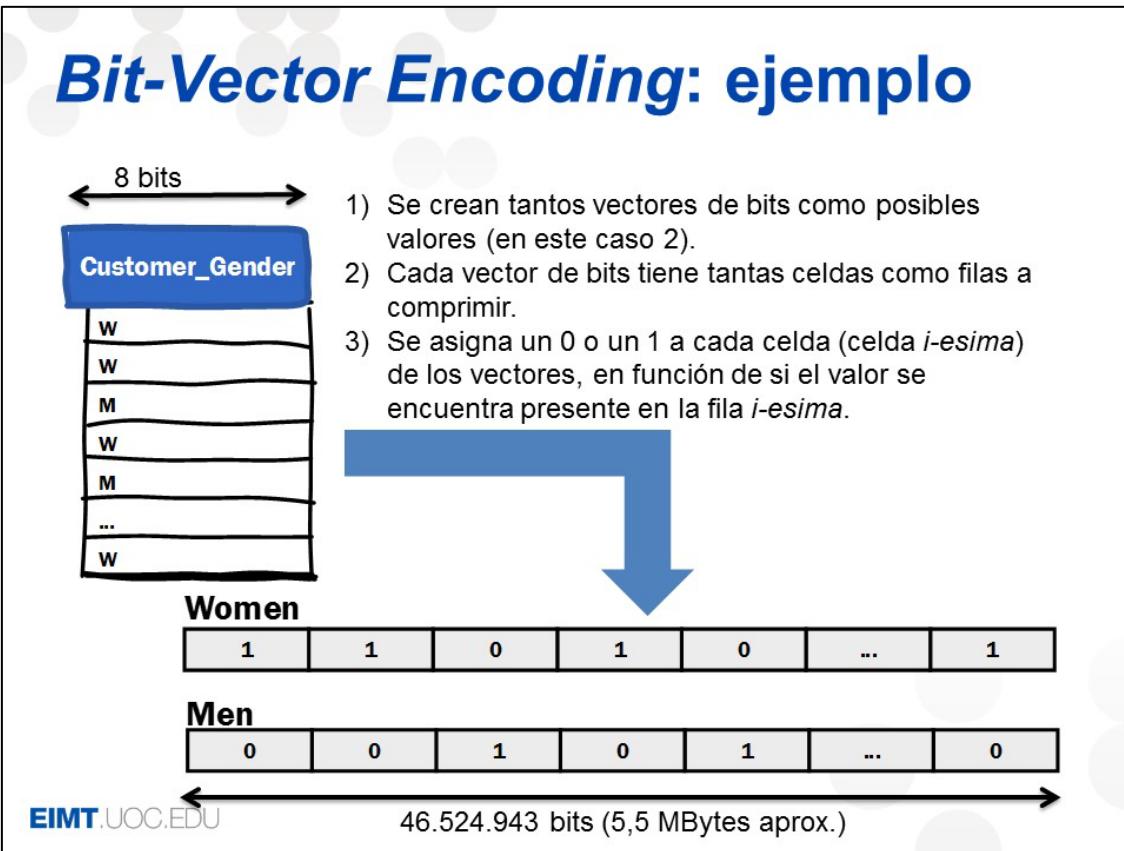
És important tenir en compte que en aquest tipus de compressió, la columna comprimida no existeix com a tal, sinó que es compon pel conjunt de vectors de bits creats.

Per les seves característiques, els vectors de bits poden tenir moltes repeticions. En el cas que es consideri necessari, es podrien comprimir usant algun dels algorismes vists anteriorment.



Seguidament, veurem com funciona aquest algorisme mitjançant un exemple. Per a això, utilitzarem la taula *Customer* que hem estat utilitzant fins ara. En aquest cas, procedirem a comprimir la columna *Customer_Gender*, que només accepta dos valors: *M* per a indicar que el client és un home (*Men*) i *W* per a indicar que el client és una dona (*Women*).

Una vegada presentat el context, vegem com es comprimeix aquesta columna mitjançant l'algorisme *Bit-vector encoding*.



Tal com hem comentat, el primer pas de l'algorisme és crear un vector de bits per a cada valor possible de la columna. Com que els valors possibles de la columna són dos (*Men* i *Women*) s'han creat dos vectors de bits:

1. *Women*: indica quins clients són de gènere femení.
2. *Men*: indica quins clients són de gènere masculí.

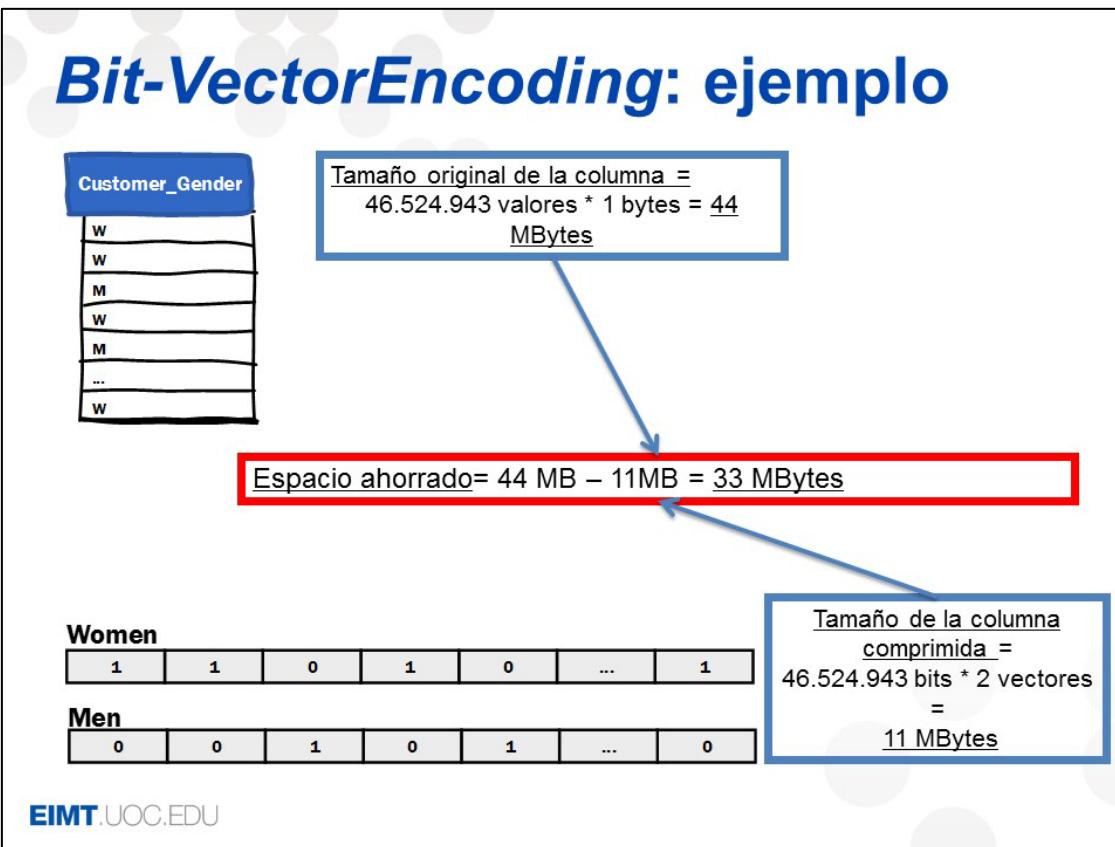
Els vectors tindran tantes posicions com cel·les tingui la columna. En aquest cas 46.524.943 bits. En conseqüència, cada vector ocuparà prop de 5,5 MBytes.

Per a assignar els valors als vectors s'iteraria sobre totes les cel·les de la columna original. Per a cada cel·la (sent i la seva posició) es realitzarien les següents accions:

- Si el $valor_i$ és *M*, llavors s'assignaria un 1 en la posició i del vector *Men* i un 0 en la posició i del vector *Women*.
- Si el $valor_i$ és *W*, llavors s'assignaria un 1 en la posició i del vector *Women* i un 0 en la posició i del vector *Men*.

A l'exemple de la transparència, la primera i segona cel·les contenen el valor *W*. Per tant, s'assignarà un 1 en la primera i en la segona posició de *Women*, i un 0 en les dues primeres posicions de *Men*. La tercera cel·la conté el valor *M*, en conseqüència s'assignarà un 1 en la tercera posició de *Men* i un 0 en la tercera posició de *Women*. I

així successivament. A la transparència podem veure el resultat d'aplicar el procediment descrit.



A continuació, vegem el resultat de la compressió en l'exemple realitzat. Veurem què ocupava la columna inicial, què ocupa la columna comprimida i quin guany (en termes d'estalvi en espai d'emmagatzematge) hem obtingut. Suposarem que la columna original emmagatzema els valors *M* i *W* mitjançant un caràcter, i, per tant, cada cel·la ocupa 1 byte. Amb aquest supòsit, la grandària de la columna original (*Customer_Gender*) és d'uns 44 MBytes.

$$\text{Grandària original (Customer_Gender)} = 46.524.943 \text{ valors} \times 1 \text{ byte} = \\ 46.524.943 \text{ bytes} \approx 44 \text{ MBytes}$$

Per la seva banda, la grandària de la nova columna serà donat per la grandària de tots els vectors de bits creats. Aquestes grandàries es poden calcular de la forma següent:

$$\text{Grandària comprimida (Customer_Gender)} = 46.524.943 \text{ valors} \times 1 \text{ bit} \times 2 \text{ vectores} = \\ 93.049.886 \text{ bits} \approx 11 \text{ MBytes}$$

Per tant, la grandària final de la columna comprimida serà d'11 MBytes. L'estalvi final serà donat per la diferència entre la grandària inicial i el comprimit:

$$\text{Estalvi (Customer_Gender)} = 44 \text{ MBytes} - 11 \text{ MBytes} = 33 \text{ MBytes}$$

Podem observar que l'estalvi obtingut en aquest cas és significatiu. No obstant això, en el cas que la columna s'hagués configurat inicialment per a ocupar només un bit no hi hauria hagut guany, més aviat al contrari.

Frame of Reference Encoding

- Útil cuando tenemos poca variabilidad alrededor de un valor medio.
- En vez de almacenar el valor total se almacena su distancia a un valor de referencia:
 - Se escoge un valor de referencia (valor medio).
 - Genérico (el mismo para toda la columna)
 - Se reescribe cada valor en función de su diferencia con el valor de referencia.
 - Si un valor está muy distante del valor de referencia, se trata como una excepción.
- Requiere de conocimiento sobre la distribución de datos dentro de una columna.
- La ordenación de los datos no afecta a su eficiencia.

EIMT.UOC.EDU

L'algorisme *Frame of reference encoding* comprimeix les dades d'una columna reduint la grandària necessària per a representar els seus valors. Per tant, no redueix la grandària en funció de les repeticions dels seus valors, ni tampoc tindrà un millor guany (o ràtio de compressió) en el cas que les dades estiguin ordenades.

Aquest algorisme és aplicable quan els valors d'una columna estan molt propers a un valor concret i mostren poca variabilitat. En aquest cas, els valors es podrien reescriure en funció de la seva distància a aquest valor de referència i així reduir, potencialment, l'espai necessari per a representar-los. Aquesta és la filosofia de treball subjacent d'aquest algorisme.

Per a executar la compressió, s'escull un valor de referència que és fix (o sigui, el mateix) per a tota la columna. Aquest valor de referència pretén ser el valor mitjà sobre el qual es distribueixen les dades. Després es reescriu el valor de cada cel·la en funció de la distància que el separa del valor de referència. Per a reduir l'espai necessari per a emmagatzemar els valors, es defineix una finestra (o marc) de referència en funció dels bits que es vulguin utilitzar per a representar la distància dels valors. És possible que hi hagi valors extrems, és a dir, valors que estan lluny del valor de referència i, per tant, la seva diferència queda fora de la finestra establerta. En aquest cas, es poden representar com a excepcions, indicant el seu valor original i una metadada (és a dir, una informació de control) que indiqui que aquest valor és una excepció (o sigui, no és la distància pel que fa al valor de referència, sinó el valor original).

És necessari escollir un bon valor i un marc de referència perquè l'algorisme aconsegueixi uns resultats satisfactoris. Això implica conèixer com són i com es distribueixen les dades dins de la columna. En el millor dels casos, l'SGBD proporcionarà una certa flexibilitat perquè l'usuari defineixi el valor i el marc de referència (almenys a nivell de *byte*).

Frame of Reference Encoding: exemple

Customer

Customer_Id : longInt
Customer_Name: String (35)
Customer_Surname: String (35)
Customer_Gender: Byte DEFAULT "W"
Customer_PostalCode: Number (5)
Customer_City: String (35)
Customer_Country: String (35)

- Partición horizontal en la tabla *Customer* en función de la provincia
- Los dos primeros dígitos de *Customer_PostalCode* definen la provincia.
- Por cada provincia con código XX, el valor medio es XX500:
 - Tomaremos XX500 como valor de referencia.
 - Tomaremos [-500, +500] como marco de referencia.

EIMT.UOC.EDU

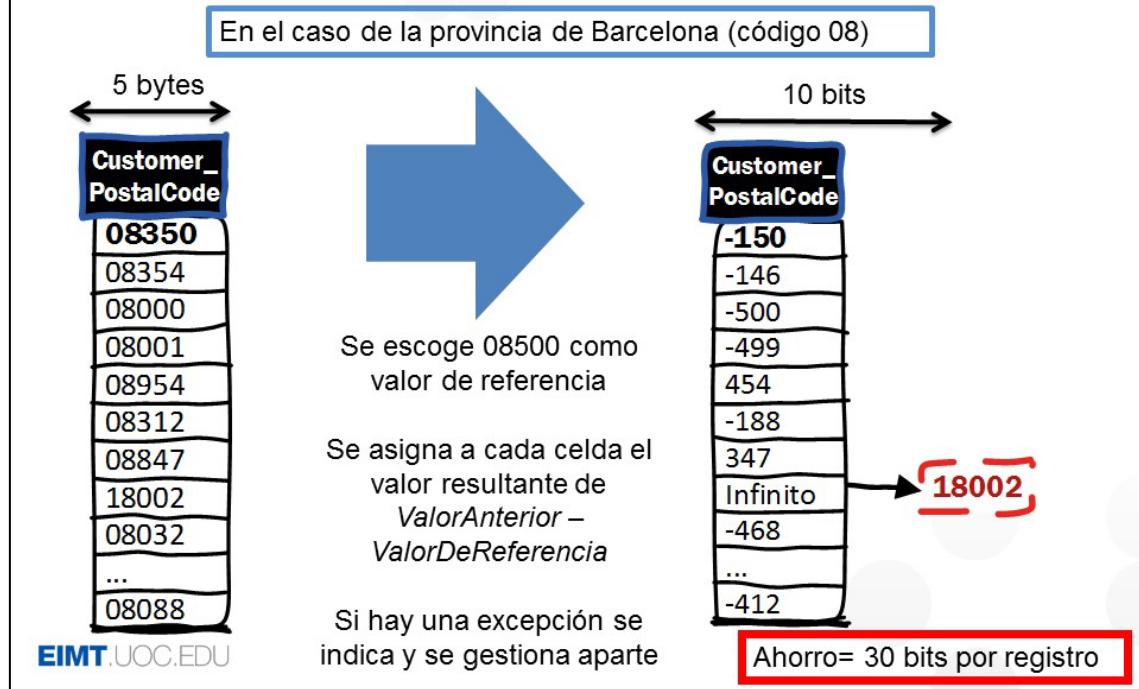
Vegem com funciona aquest algorisme mitjançant un exemple. De nou, ens basarem en la taula *Customer*. En aquest cas, procedirem a comprimir la columna *Customer_PostalCode*, el rang de valors de la qual estarà entre el 00000 i el 99999.

Perquè l'exemple sigui més clar, suposarem que la taula està dividida en particions (o fragmentada) horitzontalment per província. Cada partió horitzontal contindrà llavors tots els clients d'una província determinada, sent els dos díigits de més pes de la columna els que identificaran la província (08XXX correspon a Barcelona, 17XXX a Girona, 36XXX a Pontevedra, etc.).

Suposem que volem comprimir la columna en cadascuna de les particions utilitzant l'algorisme *Frame of reference encoding*. Sabem que donada una província XX tots els valors oscil·larien entre XX000 i XX999. En conseqüència, podríem definir el valor de referència en 500 per a cada província i el marc de referència en una distància de [-500, 500].

Una vegada presentat el context, vegem com es comprimeix aquesta columna.

Frame of Reference Encoding: exemple



En concret, vegem en aquesta transparència com es comprimiria la columna relativa a la partició horitzontal que correspon a la província de Barcelona.

El codi de Barcelona és 08, en conseqüència tots els codis postals seran del tipus 08XXX, on el valor XXX anirà des de 000 a 999. Tal com hem comentat, s'establirà el valor de referència en 08500 i el marc de referència en l'interval [-500,500]. Això vol dir que la columna comprimida haurà de ser capaç d'emmagatzemar valors que vagin del -500 a 500. Per a emmagatzemar aquests valors és necessari usar 10 bits. Per tant, la columna comprimida contindrà 10 bits en el cas general.

Per a cadascuna de les cel·les de la columna original, es calcularà la seva homòloga en la columna comprimida. Per a fer-ho es calcularà la distància del valor original amb el valor de referència (*valor original - valor de referència*). El valor resultant s'emmagatzemarà en la columna comprimida, tal com podem veure en la transparència.

En l'exemple de la transparència també podem veure com el valor 8350 es converteix en el valor -150 (resultat de l'operació 8350-8500), el valor 8354 en el valor comprimit -146, el valor 8000 en el valor -500, i així successivament. En el cas que un valor estigui més distant del valor de referència de l'esperat, es crearà una excepció. Aquesta excepció es podria representar de formes diferents, una d'aquestes seria la d'utilitzar un valor per defecte per a indicar que la distància és infinita i guardar en una altra estructura alternativa (o auxiliar) el valor original. A la transparència podem veure

que, per error, un client d'una altra província o amb un codi erroni (amb un codi que comença per 18) es troba en la partició de Barcelona. Aquest valor està clarament fora del marc de referència, ja que la seva distància al valor de referència és de 9.502 (valor molt superior al valor 512 representable amb 10 bits). En aquest cas, s'ha indicat que la distància és infinita (està més enllà del marc de referència) i el valor original s'emmagatzema apart.

Com que no sabem amb certesa el nombre d'excepcions que ens trobarem, no té molt sentit analitzar el guany total de la compressió de la columna. En aquest cas particular, creiem que és més útil indicar el guany per cel·la. Per a cada valor original de la columna dins del marc de referència, tindrem un estalvi de 30 bits (en el cas que el codi postal s'emmagatzemi mitjançant 5 caràcters) o de 7 bits (en el cas que el codi postal s'emmagatzemi com a valor numèric).

Differential Encoding

- Útil cuando tenemos poca variabilidad con los registros anteriores (*timestamps*, fechas de venta, etc.)
- Parecido al *Frame of Reference*, pero:
 - El valor de referencia no es fijo: corresponde al valor anterior en la columna.
 - Se reescribe el valor de la columna (celda *i-esima*) en función de su diferencia con el valor anterior de la misma (celda *(i-1)-ésima*)
- Buen ratio de compresión cuando los valores siguen una secuencia: *timestamps* (o fechas) en columnas ordenadas.

EIMT.UOC.EDU

L'algorisme *Differential encoding* comprimeix les dades d'una columna reduint la grandària necessària per a representar els seus valors. No obstant això, en aquest cas, per la forma de realitzar la compressió s'obté una millor ràtio de compressió quan hi ha repeticions i els valors estan ordenats.

Aquest algorisme és aplicable quan els valors d'una columna segueixen una seqüència, sia creixent o decreixent. Quan això ocorre, podem utilitzar un algorisme semblant al *Frame of reference encoding*, però utilitzant el valor de la cel·la anterior com a valor de referència. En conseqüència, seria com utilitzar un *Frame of reference encoding* però amb un valor de referència dinàmic que canvia per a cada valor de la columna. En resum, la filosofia no és emmagatzemar la distància al valor de referència, sinó emmagatzemar la diferència amb el valor anterior.

Aquest algorisme té ràtios de compressió molt altes quan comprimeixen dates, marques de temps (*timestamps*) o altres seqüències de valors que estan ordenades.

L'algorisme sempre comença replicant el valor de la primera cel·la de la columna original en la columna comprimida. A partir d'aquí, es comença a realitzar la compressió. En les cel·les següents es calcula el valor comprimit de la cel·la a partir de la diferència entre el seu valor i el valor de la cel·la anterior. En el cas que les dades estiguin ordenades és molt probable que les diferències siguin mínimes i, en conseqüència, es podrà reduir molt la grandària requerida. Igual que en l'algorisme anterior, és possible que hi hagi valors extrems (*outliers*), és a dir, valors que estan

lluny del valor de referència i, en conseqüència, fora de la finestra establerta. En aquest cas també s'estableix un marc de referència que indica quins valors seran considerats extrems. Els valors extrems es poden representar seguint una estratègia similar a l'algorisme anterior (el *Frame of reference encoding*).

Differential Encoding: Ejemplo

SALE
SALE_ID
CUSTOMER_ID
PRODUCT_ID
SALE_DATE
QUANTITY
TOTAL_PRICE

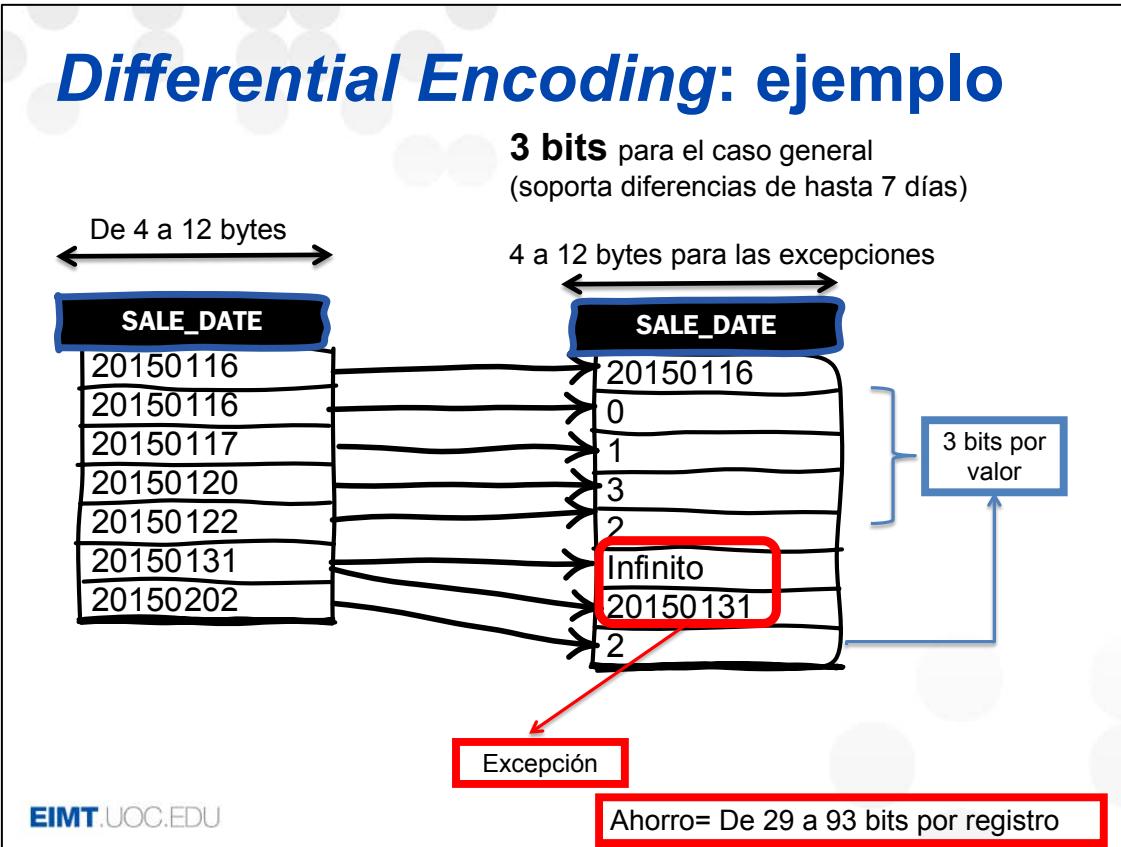
- Supongamos una proyección de la tabla ventas ordenada por fecha de venta:
(SALE_ID, PRODUCT_ID, QUANTITY, TOTAL_PRICE| SALE_DATE)
- Debido a las frecuencias de las compras, se dará la situación de que, para la columna SALE_DATE:
 - Habrá multitud de valores iguales consecutivos (ventas del mismo día).
 - Las diferencias con el valor anterior, si las hay, serán muy pequeñas (del orden de 1 día o 3 días en fines de semana).

EIMT.UOC.EDU

A continuació, vegem com funciona l'algorisme *Differential encoding* a través d'un exemple. Aquesta vegada ho farem mitjançant la taula *SALE* que, si recordeu, emmagatzema les vendes realitzades. En particular, procedirem a comprimir la columna *SALE_DATE*, que indica la data en què s'efectua la venda. Addicionalment, suposarem que les dades estan ordenades ascendentment en funció de la data de la venda.

Tenint en compte que les dates estan ordenades, els valors de la columna seguiran una seqüència ascendent, minimitzant les distàncies entre els valors consecutius dins de la columna. Aquest fet, juntament amb el volum de vendes de l'empresa (és d'esperar que hi hagi més d'una venda per dia) implicarà que, en la majoria dels casos, la distància entre una data i la següent serà molt petita. De 0 dies en el cas general, i d'1 dia a 3 dies (com a molt) en el cas dels caps de setmana.

Tenint en compte tot això, vegem com es comprimeix aquesta columna mitjançant la compressió *Differential encoding*.



Tal com hem comentat, la columna *SALE_DATE* està ordenada ascendentment. Com que hi ha més d'una venda per dia, la distància entre els valors contigus seria 0 en la majoria dels casos. En aquest exemple hem afegit més distància entre les dates per a veure exemples més diferents.

El primer que cal fer és identificar el marc de referència, és a dir, triar a partir de quina distància considerarem els elements com a elements atípics. En aquest cas hem decidit, de forma arbitrària, que la distància màxima entre dates sigui 8. És a dir, les dates amb més de 7 dies de diferència es representaran com a excepcions.

Com que tenim un marc de referència de 8 dies, necessitem només 3 bits per a representar els valors de la columna comprimida per al cas general. En els casos excepcionals, necessitarem la mateixa grandària que la data original (de 4 a 12 bytes, que és el que pot ocupar una data en funció del seu tipus).

Prenent com a base la columna de la part esquerra de la figura, la compressió es realitzaria de la forma següent. Es copiaria el primer valor de la columna en la columna comprimida. Per a calcular el segon valor, es calcularia la diferència entre el primer valor de la columna (*16 de gener de 2015*) i el segon (*16 de gener de 2015*). Com que ambdues dates són iguals, la diferència és 0, i s'afegiria el valor 0 en la segona cel·la de la columna comprimida. El tercer valor de la columna original (*17 de gener de 2015*) és un dia posterior al *16 de gener de 2015* (el segon valor de la columna original). En conseqüència, s'inseriria un 1 en la tercera cel·la de la columna comprimida. El nou

valor de referència és la data *17 de gener de 2015*, que és 3 dies anteriors al valor de la quarta cel·la de la columna (*20 de gener de 2015*). Per tant, s'assignaria un 3 en la quarta cel·la comprimida. El mateix procediment s'usaria per a la cinquena cel·la, que implicaria escriure el número 2 en la columna comprimida.

Per la seva banda, el valor de la sisena cel·la suposa una excepció, ja que la seva diferència amb el valor de la cel·la anterior és 9, que és superior a 7, valor màxim que podem representar amb 3 bits. En aquest cas, s'indicaria que és una excepció i s'emmagatzemaria el valor original de la cel·la, passant a ser aquest valor el nou valor de referència. En el cas següent, el valor correspon al *2 de febrer de 2015*, que difereix 2 dies del valor de referència. En conseqüència, s'emmagatzemaria el valor 2 en la cel·la comprimida i finalitzaria el procés de compressió.

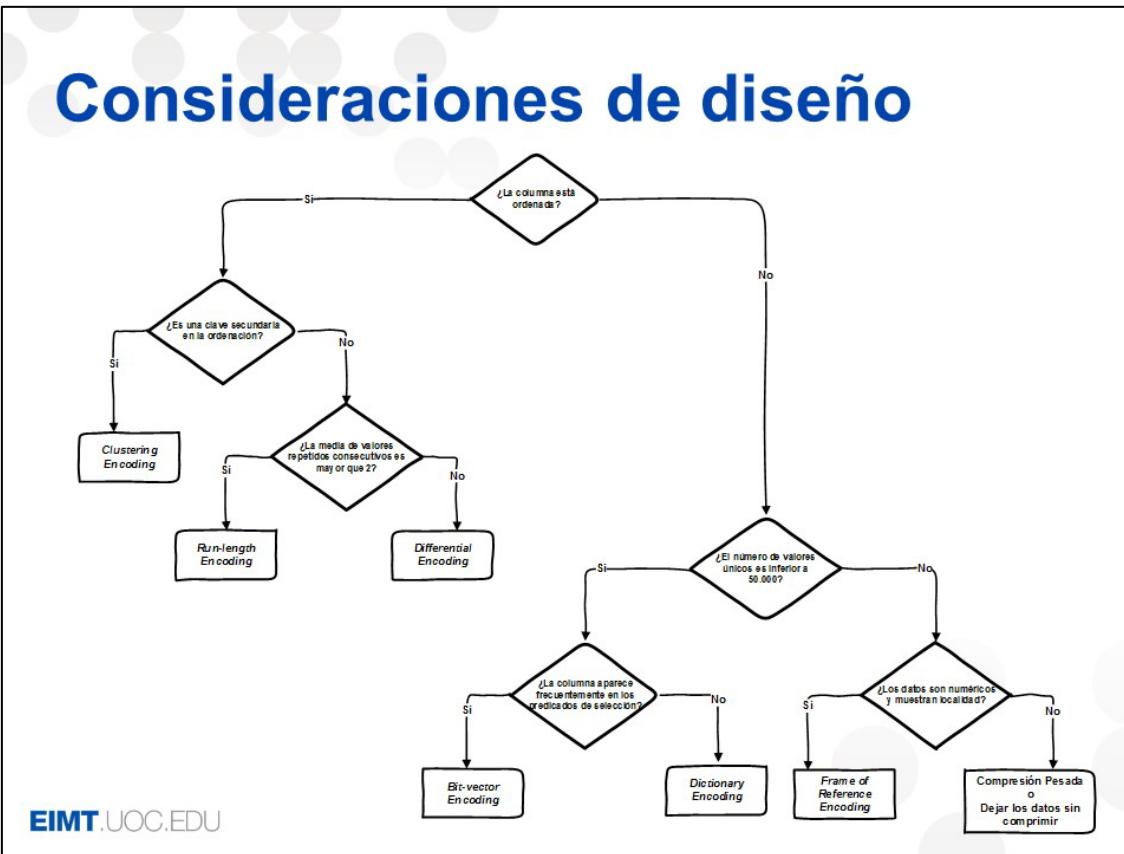
Com que no sabem amb certesa el nombre d'excepcions que trobarem, tampoc no té molt sentit, en aquest cas, analitzar el guany total (en termes d'estalvi d'espai d'emmagatzematge) de la compressió de la columna. De nou, com en el cas del *Frame of reference encoding*, creiem més útil indicar el guany per cel·la. Per a cada valor original de la columna dins del marc de referència, tindrem un estalvi de 29 a 93 bits (en funció de com s'hagi emmagatzemat la data en la columna original).

Compresión de datos

- Motivación
- Tipos de compresión de datos
- Algoritmos de compresión
- Consideraciones de diseño**
- Ejemplos prácticos

EIMT.UOC.EDU

Una vegada coneixem els algorismes de compressió més representatius, és important saber quan és aconsellable aplicar cadascun d'aquests. A continuació, presentarem algunes regles bàsiques basades en les referències presentades al final de la presentació.



Tal com hem apuntat en aquests materials didàctics, quan s'usen mètodes de compressió lleugera, és molt important conèixer les característiques de les dades a comprimir per a garantir una bona compressió de les mateixes. Recordem que en els casos extremos, els algorismes de compressió poden tenir l'efecte contrari al desitjat i poden generar columnes de més grandària si s'apliquen en contextos inadequats.

L'arbre de decisió presentat en aquesta transparència permet, de forma ràpida, fer-se una idea de les característiques principals a tenir en compte abans d'escolhir quin algorisme de compressió és el més adequat per a cada columna. Aquest arbre ha estat adaptat del treball de la tesi de Daniel Abadi i enriquit amb els suggeriments i comentaris dels autors indicats en l'apartat de les referències bibliogràfiques que teniu al final d'aquesta presentació. Es tracta de consideracions generals que serà necessari matisar en cada cas o context concret. També poden canviar amb el transcurs del temps, ja que algunes de les condicions proposades indiquen valors constants que es poden veure condicionats per avenços en la tecnologia (maquinari) i els SGBD (programari).

Dit això, la primera pregunta que ens hem de fer abans de decidir com (o si hem de) comprimir una columna és la següent: la columna està ordenada? En cas afirmatiu, sembla que podrem aprofitar la localitat de dades i repeticions per a comprimir la columna (sembla interessant usar el *Run-length encoding*, el *Cluster encoding* i el *Differential encoding*).

A partir d'aquí, caldria estudiar cada situació més a fons, però algunes regles que podríem considerar serien les següents: en el cas que la columna no sigui la clau primària en l'ordenació, sinó la clau secundària, pot ser interessant utilitzar el *Cluster encoding*. Un exemple seria quan volem comprimir una columna que indiqui el cognom dels nostres clients i la taula de clients (*Customer*) estigués ordenada primer pel nom i després pels cognoms. En el cas que la mitjana de valors consecutius sigui major que 2, la bibliografia proposa utilitzar el *Run-length encoding*. En cas contrari, s'aconsella utilitzar el *Differential encoding*.

Quan les dades de la columna no estiguin ordenades, no podem garantir que els valors iguals apareixeran de forma consecutiva en les estructures d'emmagatzematge i, en conseqüència, no tenim garanties que els algorismes anteriors funcionin adequadament. En aquestes situacions caldria considerar el nombre de valors únics que apareixen en la columna. En el cas que el nombre de valors no sigui molt elevat (es considera que el nombre de valors és molt elevat quan supera els 50.000 valors diferents) serà factible utilitzar la compressió utilitzant diccionaris i mapes de bits. L'ús d'un o l'altre dependrà bàsicament de si la columna s'utilitza freqüentment en els predicats de la selecció (per exemple, en les clàusules `WHERE` de les sentències SQL que s'executin sobre la BD). Si això fos així, d'acord amb la bibliografia, la millor alternativa seria l'ús de l'algorisme *Bit-vector encoding*. En cas contrari, es proposa usar el *Dictionary encoding*.

En columnes no ordenades on hi hagi més de 50.000 valors diferents, no es recomana usar diccionaris ni vectors de bits. Gestionar més de 50.000 vectors de bits o diccionaris de molts elements pot ser molt complex i pot excedir l'espai que es pot allotjar en la memòria, complicant la gestió de les dades. En aquests casos es proposa utilitzar l'algorisme *Frame of reference encoding* quan les dades siguin numèriques i mostrin localitat. En cas contrari, es proposa deixar les dades sense comprimir o usar algorismes de compressió pesada.

Ejemplos prácticos

- Hay multitud de sistemas de compresión y codificación:
 - Aquí hemos visto sólo los más relevantes.
 - Podemos encontrar variantes de los vistos aquí en los SGBD.
- Algunos SGBD permiten definir cómo comprimir las distintas columnas al definir las tablas:
 - Es importante estudiar si permite compresión y de qué tipo.
- Por ejemplo, en el caso de MonetDB:
 - No permite que el usuario indique cómo comprimir los datos.
 - Utiliza algunos algoritmos de compresión interna y automáticamente:
 - Representa los datos de forma compacta incluyendo el valor nulo en el espacio de valores de las columnas.
 - Representa los *strings* utilizando *dictionary encoding*.

EIMT.UOC.EDU

Ja per acabar, parlarem una mica de com podem trobar aquests algorismes de compressió en els SGBD amb els quals treballem. Primerament, cal tenir en compte que aquí hem vist alguns algorismes de compressió, però que n'hi molts més i no només això, sinó que hi ha moltes versions diferents de cada algorisme. De fet, és difícil trobar en els SGBD exactament la mateixa implementació de cadascun dels algorismes de compressió que hem explicat aquí.

En alguns casos, els SGBD utilitzen algorismes de compressió de forma automàtica sense necessitat que l'usuari hi intervingui. Alguns exemples són els vectors de bits que puguin utilitzar per a l'optimització de les consultes de diferents SGBD (sia magatzems de files o de columnes), com PostgreSQL, per exemple.

En altres casos, l'usuari no intervé en l'elecció dels algorismes de compressió a utilitzar. En altres paraules, els magatzems de columnes poden utilitzar la compressió internament i sense el control de l'usuari, per a millorar la seva eficiència. Un exemple és MonetDB, que no permet que l'usuari indiqui com comprimir les dades, però que internament aplica alguns algorismes de compressió de forma automàtica. Un d'aquests és utilitzar una variant del *Dictionary encoding* per a representar les columnes que té associat un tipus de dades *string*.

Finalment, els SGBD també poden permetre que el dissenyador i l'administrador de la BD pugui especificar els algorismes de la compressió a aplicar a les columnes en el moment de la creació de les taules o amb posterioritat (sobre la base que inicialment

es va decidir que les dades de la taula no estiguessin comprimides). En aquestes situacions, és important identificar quines opcions de compressió permet l'SGBD i analitzar-les a fons per a veure en quins casos són adequades. Per a cada algorisme suportat caldrà veure quin marge de maniobra ofereix a l'usuari per a indicar els paràmetres de compressió (si permet definir el valor i grandària del marc de referència, la grandària del diccionari, etc.). A continuació, i a manera d'exemple, vegem els diferents algorismes de compressió que permet utilitzar Redshift.

Ejemplos prácticos

- En Redshift al crear una tabla podemos indicar los siguientes parámetros de compresión:

Tipo de compresión	Tipos de datos soportados	Compresión
Raw (no compression)	All	Ninguna
Byte dictionary	All except BOOLEAN	<i>Dictionary Encoding</i> (diccionario por página)
Delta	SMALLINT, INT, BIGINT, DATE, TIMESTAMP, DECIMAL INT, BIGINT, DATE, TIMESTAMP, DECIMAL	Variante de <i>Differential Encoding</i>
LZO	All except BOOLEAN, REAL, and DOUBLE PRECISION	Compresión pesada
Mostlyn	SMALLINT, INT, BIGINT, DECIMAL INT, BIGINT, DECIMAL BIGINT, DECIMAL	NA
Run-length	All	<i>Run-length Encoding</i>
Text	VARCHAR only	<i>Dictionary Encoding</i>

Tal com hem comentat, amb Redshift, quan definim una taula, podem indicar si volem que les dades de les diferents columnes es guardin comprimides i amb quin esquema de compressió. A la transparència es poden observar els tipus (o algorismes) de compressió disponibles en aquest SGBD, sobre quin tipus de dades es poden aplicar i a quin dels algorismes que hem vist en aquesta presentació s'assemblen més.

El *Byte dictionary*, per exemple, permet crear un diccionari amb 255 entrades per a cada pàgina del disc. Per tant, i segons la documentació del fabricant, seria una adaptació de l'algorisme *Dictionary encoding* en què hi ha un diccionari per pàgina i amb un límit d'entrades. La compressió del tipus *Text* funciona de forma similar, però permet augmentar el nombre d'entrades del diccionari.

L'algorisme de compressió *Delta* sembla ser una adaptació del *Differential encoding* en què l'espai de valors pot ser d'1 a 2 bytes, segons s'indiqui.

El mètode de compressió *Mostlyn* és semblant al de *Frame of reference* però sense un valor de referència i s'usa quan els tipus de dades d'una columna permeten un rang de valors superior al que realment s'emmagatzemen en la columna. En aquest cas, l'SGBD permet redefinir la columna com *Mostlyn*, on *N* indica el nombre de bits que s'utilitzarà en la compressió (8, 16 o 32) i l'SGBD utilitzarà menys bits per a representar cadascun dels valors quan sigui possible. Quan no sigui possible, s'emmagatzemarà el valor original.

Finalment, el mètode *Run-length* de Redshift és una adaptació de l'algorisme *Run-length encoding* expliat en aquest material didàctic.

En cas que es decideixi que es vol aplicar la compressió de dades en el moment de la creació de la taula, però es desitja que l'elecció dels esquemes de compressió sigui decidida automàticament per l'SGBD, com és el cas de Redshift, és necessari executar l'ordre `COPY` amb l'opció `COMPUPDATE` activada (és a dir, amb l'opció `ON`) en el moment d'inserir les dades en la nova taula.

Quan es decideix especificar manualment els esquemes de compressió i per a ajudar en la presa de decisions, els SGBD incorporen utilitats que assisteixen el dissenyador o l'administrador de la BD. En el cas concret de Redshift, per exemple, l'ordre `ANALYZE COMPRESSION`, suggereix esquemes de compressió per a les columnes d'una taula que conté dades. Pot ser utilitzada per a decidir els esquemes de compressió d'una taula sobre la qual inicialment s'havia decidit que les dades no estiguessin comprimides o per a decidir els esquemes de compressió d'una nova taula que té característiques similars a la que s'examina.

Compresión de datos

- Motivación
- Tipos de compresión de datos
- Algoritmos de compresión
- Consideraciones de diseño
- Ejemplos prácticos

EIMT.UOC.EDU

I fins aquí aquests materials sobre la compressió de dades en magatzems de columnes. Hem vist el que pot aportar la compressió de dades en els magatzems de columnes, quins tipus de compressió de dades existeixen i quins són els més usats en els magatzems de columnes. Després hem explicat com funcionen alguns dels algorismes més representatius per a comprimir les dades en els magatzems de columnes i quins criteris de disseny cal tenir en compte per a escollir els algorismes més adequats en funció de les característiques de cada columna. Finalment, hem vist un parell d'exemples de com SGBD concrets utilitzen la compressió de dades.

Esperem que hagiu trobat els materials amens, útils i el tema interessant. A continuació trobareu un conjunt de referències rellevants sobre el tema, per si us interessa aprofundir més en aquestes qüestions.

Referencias

- D.J. Abadi (2008). *Query Execution in Column-Oriented Database Systems*. PhD Dissertation in Computer Science and Engineering at the MIT. Advisor: Samuel Madden. Chapter 4.
- D.J. Abadi, S. Madden, N. Hachem (2008). Column-stores vs. Row-stores: How different are they really? *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 967–980.
- S. Harizopoulos, D. Abadi, Peter Boncz (2009). Column-oriented Database Systems. VLDB 2009 Tutorial. Transparencias 49 a 60.
- D.J. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, S. Madden (2012). The Design and Implementation of Modern Column-Oriented Database Systems. *Foundations and Trends in Databases*, 5(3), pages 197-280.
- H. Plattner (2013). *A Course in In-Memory Data Management*. Editorial Springer. Capítulos 6 y 7.
- G. Harrison (2015). *Next Generation Databases*. Editorial Apress. Capítulo 6.