Exercici 3

Enunciat 1

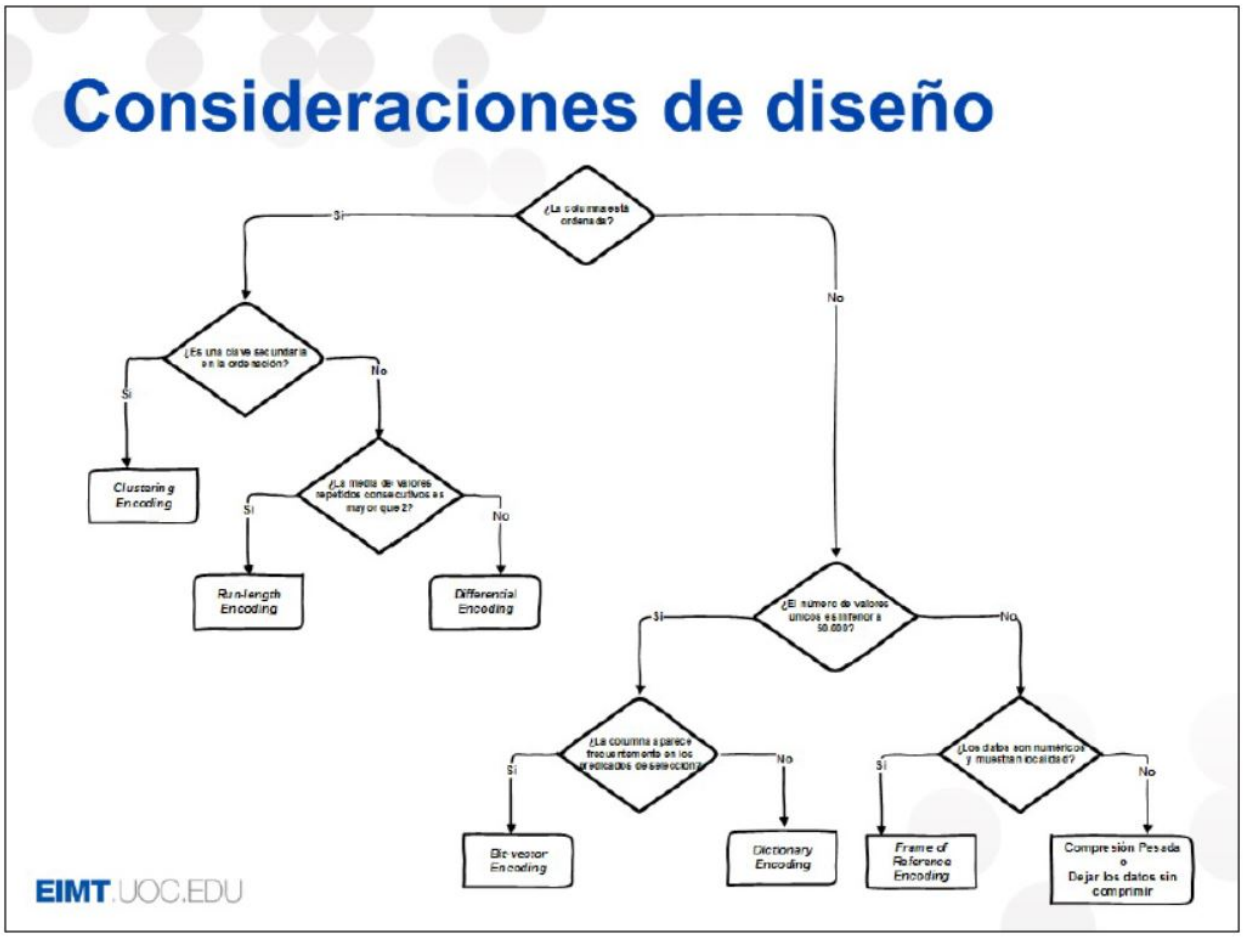
Sabem que la nostra dimensió de productes té un nombre de files elevat, concretament 10 milions de files (recordeu que aquesta dimensió manté l'històric de productes segons han canviat categories, subcategories, etc.).

Volem crear una projecció que contingui les columnes category_name i subcategory_name, i volem aplicar algorismes de compressió sobre totes dues columnes.

Es demana proporcionar quins algorismes de compressió poden donar-nos un bon resultat per a reduir l'espai d'emmagatzematge de totes dues columnes en la nova projecció i expliqueu breument el per què. Proporcionar diferents suposats si ho considereu oportú (no és necessari realitzar i proporcionar els càlculs d'espai de les noves columnes sobre la base dels algorismes que presenteu).

Abans de res, dir que descartem els algorismes heavyweigth. Podrien funcionar bé, però no els hem considerat per les raons explicades al material docent: són complexes, requereixen de molta CPU i no permeten ser tractats sense descomprimir. Dins del lightweigth, considerarem tant els de reducció de files com els de reducció de valors. S'especificarà, a cada cas, de quin tipus es tracta.

Tenint en compte l'arbre de decisió proporcionat al material docent:



Feim les següents assumpcions:

1. **Les columnes category_name i subcategory_name no estan ordenades.** Això ens fa descartar algorismes que, en principi, ens podrien funcionar com "Run-Length Encoding" o "Cluster Encoding". Si els volguessim utilitzar, hauriem de canviar el funcionament del procediment sp_load_product_dim.
2. **Farem les consideracions sobre el nombre de valors diferents:** entendrem que el nombre de categories i subcategories és baix. Si fos alt, l'alternativa és deixar la columna sense comprimir o aplicar un algorisme heavyweigth.
3. **Respecte a si apareix freqüentment als predicats de seleccio,** considerarem les dues alternatives: sí i no. Veurem quin és el resultat en cada cas.

Si les columnes apareixen freqüentment als predicats where, utilitzarem l'algorisme "bit-vector encoding". És un algorisme de reducció de valor, no de nombre de registres. Per a fer això:

Es crea un vector de bits per a cada category_name i subcategory_name.

Si considerem les dades insertades a l'exercici 3, són 6 categories i 15 subcategories. Això suposaria crear 21 arrays d'un bit, amb tant de valors com registres té la taula (10 milions). Els arrays serian de la forma:

Categoria					
Frutas	0	1	0	...	0
Bebidas	1	0	0	...	1
Congelados	0	0	1	...	0

Subcategoria					
Cítricos	0	1	0	...	0
Vinos	1	0	0	...	1
Pizzas	0	0	1	...	0

El tamany agrupat de les dues columnes actualment és complicat de calcular, ja que hauriem de conèixer l'encoding de la base de dades. Si pensem que és de 2 Bytes per lletra, pasem de:

Tamany actual = 2 camps * 40 caràcters * 2 Bytes * 10 milions de registres

A:

Tamany comprimit: 21 vetors * 1 bit * 10 milions de registres

En qualsevol cas, hem de pensar que el VARYING només ocupa l'espai que necessita (no els 40 caràcters) i que aquesta estratègia es pitjor com més valors diferents tenim. A un entorn real de milers de categories i subcategories pot ser un desastre.

Sembla més natural seguir l'estratègia de "Dictionary encoding", especialment si no feim moltes consultes per aquests camps. L'estratègia és:

Cream un dccionari per a cada valor diferent que es troba a la taula. Seria de la forma (igual amb subcategories):

Categoria	
0	Frutas
1	Bebidas
2	Congelados
3	Lacteos
4	Carne
5	Higiene

I assignem a cada valor de la taula el seu codi en lloc de la descripció. Així, si la taula té un "category_name" de "Lácteos", sustituirem la cadena "Lácteos" per un 3. Pensem que necessitem 3 bits per a representar les categories i 4 les subcategories. Així, el tamany comprimit per ambdues columnes seria aproximadament de:

Tamany comprimit: (21 valors als diccionari * 40 caràcters * 2 Bytes) + (10 milions de registres * 3 bits) + (10 milions de registres * 4 bits)

Enunciat 2

Suposant que els nostres usuaris realitzen les següents consultes:

• El 65% de les consultes mostra informació de surt per país del client i client, de forma anual sobre la base de la data de comanda (order_date).

• Un 15% de les consultes es mostra informació de tots els productes (nom, categoria, subcategoria, preu, data d'inici i data de fi), que s'utilitzarà per a importar a una base de dades de gestió de dades mestres.

• La resta de consultes que llancen els usuaris són consultes ad hoc (consultes no predefinides), és a dir, no sabem el tipus de consulta que s'espera sobre aquest model dimensional.

Definir les projeccions necessàries per a donar una solució de rendiment òptima al tipus de consultes que s'han especificat anteriorment. No heu de definir més de 3 projeccions en total (sense comptar les superproyecciones de cadascuna de les taules). Explicar breument la clau d'ordenació seleccionada i per a quines consultes es crea la projecció.

Recorem que, seguint les tècniques de Database Cracking, no definim índexos fixos a la base de dades. La projecció és adaptativa a la consulta específica que llença l'usuari. Així, si definim una projecció per a optimitzar les consultes, aquesta projecció s'adaptarà als cracks (zones específiques de la base de dades) quan es llencin consultes noves.

Jo crearia dues projeccions per a la consulta 1, ja que representa el 65% de les consultes. Crearia una per a enguany i una altra per a l'any anterior. És d'esperar que siguin els dos cracks més consultats.

Com que són dues taules diferents, entenc que s'han de juntar a una única taula. A mode d'exemple, escribim la query amb una join.

Seguint la sintaxi Vertica:

CREATE PROJECTION orders_2019

AS

SELECT CLI.client_code, CLI.country

FROM tb_order_line_fact OLF

JOIN tb_client_dim CLI ON (

`OLF.client_key = CLI.client_code`

)

WHERE OLF.order_date BETWEEN TO_DATE('01/01/2019', 'dd/mm/yyyy') AND TO_DATE('31/12/2019 23:59:59', 'dd/mm/yyyy hh24:mi:ss')

ORDER BY OLF.order_date;

CREATE PROJECTION orders_2018

AS

SELECT CLI.client_code, CLI.country

FROM tb_order_line_fact OLF

JOIN tb_client_dim CLI ON (

`OLF.client_key = CLI.client_code`

)

WHERE OLF.order_date BETWEEN TO_DATE('01/01/2018', 'dd/mm/yyyy') AND TO_DATE('31/12/2018 23:59:59', 'dd/mm/yyyy hh24:mi:ss')

ORDER BY OLF.order_date;

Finalment, crearia una tercera projecció per a la consulta del 15%. Com que es vol per a una taula mestra seria normal fer un group by per a tenir els valors diferents. No ho farem perque s'inclou la data d'inici i la data fi. Un mateix producte no pot tenir dos registres iguals. Orden per data perque entenc que és millor per a fer càrregues incrementals (permet filtrar millor pels productes a partir del moment de la darrera càrrega).

CREATE PROJECTION master_data

AS

SELECT PRO.product_name, PRO.category_code, PRO.subcategory_code, PRO.price, PRO.start_date, PRO.end_date

FROM surt.tb_product_dim PRO

ORDER BY PRO.start_date DESC;