Documento anónimo

Soluciones.pdf

Soluciones

2º Fundamentos de Bases de Datos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Soluciones a las consultas SQL y AR de las Bases de Datos de Bike Stores, de agentes/clientes y de la liga de baloncesto

Manuel Pegalajar Cuéllar

11 de mayo de 2020

1. Consultas sobre la Base de Datos Bike Stores

NOTA ACLARATORIA: LAS SOLUCIONES QUE SE MUESTRAN A CONTINUACIÓN SON POSIBLES SOLUCIONES, Y NO LAS ÚNICAS SOLUCIONES POSIBLES. HAY MULTIPLES FORMAS DE ESCRIBIR UNA CONSULTA. EN EL CASO DEL AR, SE HA SEGUIDO EL PRINCIPIO DE REALIZAR UNA SELECCIÓN Y/O UNA PROYECCIÓN TAN PRONTO COMO SEA POSIBLE.

SE OMITEN LAS SOLUCIONES DEL AR QUE NO PUEDEN REALIZARSE.

1.1. Consultas básicas

1. Seleccionar todos las filas de la tabla de clientes

   SQL:

   SELECT * FROM CUSTOMERS;

   AR:

   CUSTOMERS

2. Seleccionar sólo las columnas first_namey last_name de la tabla de clientes

   SQL:

   SELECT first_name, last_name FROM CUSTOMERS;

   AR:

   $\pi_{first\_name,last\_name}(customers)$

3. Selección de todas las filas y columnas de la tabla de clientes de New York, ordenadas por apellidos y nombre

   SQL:

SELECT * FROM CUSTOMERS WHERE

city='New York'

ORDER BY last_name, first_name;

AR: No se puede ordenar. Salvo eso, sería:

$\sigma_{city='NewYork'}(customers)$

4. Selección de todos (filas y columnas) los empleados

   SQL:

   SELECT * FROM STAFFS;

   AR:

   staffs

5. Selección de todas (filas y columnas) las marcas

   SQL:

   SELECT * FROM BRANDS;

   AR:

   brands

6. Selección de todas (filas y columnas) las categorías

   SQL:

   SELECT * FROM CATEGORIES;

   AR:

   categories

7. Selección de todas los productos (filas y columnas) cuyo precio esté entre 100 y 500, ordenado por precio descendentemente

   SQL:

   SELECT * FROM PRODUCTS WHERE

   list_price BETWEEN 100 AND 500

   ORDER BY LIST_PRICE DESC;

   AR: No se puede ordenar. Salvo eso, sería:

   $\sigma_{list\_price \geq 100 \wedge list\_price \leq 500}(products)$

8. Seleccionar productos con precio superior a 1000

   SQL:

   SELECT * from products where

   list_price>1000

   AR:

   $\sigma_{list\_price > 1000}(products)$

9. Seleccionar nombres y correos electrónicos de tiendas de Santa Cruz

   SQL: SELECT store_name, email FROM STORES WHERE

   city='Santa Cruz';

   AR:

   $\pi_{store\_name,email}(\sigma_{city='SantaCruz'}(stores))$

10. Seleccionar precio+descuento de todos los items vendidos

    SQL:

    SELECT LIST_PRICE+DISCOUNT FROM order_items;

    AR: No se puede. AR no tiene operadores aritméticos

11. Seleccionar el email del personal cuyo nombre y apellidos es Fabiola Jackson

    SQL:

    SELECT email FROM STAFFS WHERE

    first_name='Fabiola' and last_name='Jackson';

    AR:

    $\pi_{email}(\sigma_{first\_name='Fabiola' \wedge last\_name='Jackson'}(staffs))$

12. Seleccionar el personal que no tenga manager. Es obligatorio hacer uso de la condición IS NULL en la consulta

    SQL:

    SELECT * FROM STAFFS WHERE

    manager_id IS NULL;

    AR: No se puede. No dispone de IS NULL

13. Consulta todos los atributos de empleados cuyos jefes se llaman Mireya

    SQL:

    SELECT staffs.* FROM staffs, staffs jefes WHERE

    staffs.manager_id=jefes.staff_id AND jefes.first_name='Mireya';

    AR:

    $\rho(staffs) = jefes$

    $\pi_{staffs.*}(staffs \bowtie \sigma_{first\_name='Mireya'}(jefes))$

14. Nombre, apellidos, código postal, fecha de pedido y estado del pedido de clientes que han realizado pedidos con fecha superior a 2017 (del 1 de Enero de 2018 en adelante) ordenados por fecha de pedido

    SQL:

    SELECT first_name, last_name, zip_code, order_date, order_status FROM CUSTOMERS, ORDERS WHERE

CUSTOMERS.customer_id=ORDERS.customer_id AND
order_date>= TO_DATE('2018/01/01', 'yyyy/mm/dd')
ORDER BY order_date;

AR:

15. Nombre, apellidos y nombre de la tienda donde trabajan los empleados, ordenados por nombre de tienda ascendentemente, y por apellidos descendentemente en caso de empate

    SQL:

    SELECT first_name, last_name, store_name FROM staffs, stores WHERE
    staffs.store_id=stores.store_id
    ORDER BY store_name, last_name DESC;

    AR: No se puede ordenar. Salvo eso, sería:

    $\pi_{first\_name,last\_name,store\_name}(staffs \bowtie stores)$

16. Seleccionar nombre y apellidos de personal (staffs) que hayan procesado pedidos de clientes de la ciudad de New York

    SQL:

    SELECT distinct STAFFS.first_name, STAFFS.last_name FROM STAFFS, ORDERS, CUSTOMERS WHERE
    staffs.staff_id=orders.staff_id AND orders.customer_id=customers.customer_id AND customers.city='New York';

    AR:

    $\pi_{staffs.first\_name,staffs.last\_name}(\sigma_{customers.city='NewYork'}(curtomers \bowtie (staffs \bowtie orders)))$

17. Seleccionar nombres de tiendas que tienen productos de la categoría Mountain Bikes

    SQL:

    SELECT distinct STORES.store_name FROM STORES, PRODUCTS, STOCKS, CATEGORIES WHERE
    stores.store_id=stocks.store_id AND products.product_id=stocks.product_id AND products.category_id=categories.category_id AND
    categories.category_name='Mountain Bikes';

    AR:

    $\pi_{store\_name}(stores \bowtie stocks \bowtie products \bowtie \sigma_{category\_name='MountainBikes'}(categories))$

18. Seleccionar nombres de productos vendidos, y cantidad, a clientes que viven en Bronx

    SQL:

SELECT PRODUCTS.product_name, ORDER_ITEMS.quantity FROM PRODUCTS, ORDER_ITEMS, CUSTOMERS, ORDERS WHERE

orders.order_id=order_items.order_id AND products.product_id=order_items.product_id AND orders.customer_id=customers.customer_id AND customers.city='Bronx';

AR:

$\pi_{product\_name,quantity}(products \bowtie order\_items \bowtie orders \bowtie \sigma_{city='Bronx'}(customers))$

19. Seleccionar nombre y apellidos de clientes que han comprado productos de la categoría Mountain Bikes

    SQL:

    SELECT distinct customers.first_name, customers.last_name FROM CUSTOMERS, PRODUCTS, CATEGORIES, ORDERS, ORDER_ITEMS WHERE

    customers.customer_id=orders.customer_id AND orders.order_id=order_items.order_id AND orders.customer_id=customers.customer_id AND

    order_items.product_id=products.product_id AND products.category_id=categories.category_id AND categories.category_name='Mountain Bikes';

    AR:

    $\pi_{customers.first\_name,customers.last\_name}(customers \bowtie orders \bowtie order\_items \bowtie products \bowtie \sigma_{category\_name='MountainBikes'}(categories))$

20. Seleccionar las marcas que ha vendido el personal de la tienda localizada en Santa Cruz

    SQL:

    SELECT brands.* FROM brands,stores,orders,order_items,products WHERE

    stores.city='Santa Cruz' AND orders.store_id=stores.store_id AND

    orders.order_id=order_items.order_id AND order_items.product_id=products.product_id AND products.brand_id=brands.brand_id;

    AR: $\pi_{brands.*}(\sigma_{city='SantaCruz'}(stores) \bowtie orders \bowtie order\_items \bowtie products \bowtie brands)$

21. Seleccionar el precio promedio, su desviación típica, precio mínimo y precio máximo de productos

    SQL:

    SELECT AVG(list_price), STDDEV(list_price), MIN(list_price), MAX(list_price) FROM PRODUCTS;

    AR: No se puede, no tiene agregadores

22. Seleccionar el nombre del producto de mínimo precio, junto con su marca, categoría y precio

SQL:

SELECT product_name, brand_name, category_name, list_price FROM PRODUCTS, BRANDS, CATEGORIES WHERE

PRODUCTS.brand_id=BRANDS.brand_id AND

PRODUCTS.category_id=CATEGORIES.category_id AND

list_price<=ALL(

SELECT P2.list_price FROM PRODUCTS P2

);

AR: Idea: A todos los productos se les quitan aquellos que tienen algún otro producto de precio menor. Quedan los de mínimo precio.

$\rho(products) = p2$

$\pi_{product\_name,brand\_name,category\_name,list\_price}($

$\pi_{product\_id,product\_name,list\_price}(products) -$

$\pi_{p2.product\_id,product\_name,list\_price}(\sigma_{products.list\_price<p2.list\_price}(products \times p2))) \bowtie brands \bowtie categories)$

23. Seleccionar el nombre del producto de máximo precio, junto con su marca, categoría y precio

SQL:

SELECT product_name, brand_name, category_name, list_price FROM PRODUCTS, BRANDS, CATEGORIES WHERE

PRODUCTS.brand_id=BRANDS.brand_id AND

PRODUCTS.category_id=CATEGORIES.category_id AND

list_price>=ALL(

SELECT P2.list_price FROM PRODUCTS P2

);

AR: Idea: A todos los productos se les quitan aquellos que tienen algún otro producto de precio mayor. Quedan los de máximo precio

$\pi_{product\_name,brand\_name,category\_name,list\_price}($

$\pi_{product\_id,product\_name,list\_price}(products) -$

$\pi_{p2.product\_id,p2.product\_name,p2.list\_price}(\sigma_{products.list\_price>p2.list\_price}(products \times p2))) \bowtie brands \bowtie categories)$

24. Seleccionar el nombre de productos que no estén en stock (que no exista una fila en la tabla stocks para esos productos)

SQL:

SELECT product_name from products WHERE NOT EXISTS(

SELECT * FROM stocks WHERE stocks.product_id=products.product_id

);

AR:

$\pi_{product\_name}((\pi_{product\_id}(products) - \pi_{product\_id}(stocks)) \bowtie products)$

25. Seleccionar productos vendidos a precio diferente del producto

    SQL 1: Procedente del AR

    SELECT product_name FROM products, order_items WHERE

    order_items.product_id=products.product_id AND order_items.list_price <> products.list_price;

    SQL 2: Procedente del CRT

    SELECT product_name FROM products WHERE EXISTS(

    SELECT * FROM order_items WHERE

    order_items.product_id=products.product_id AND order_items.list_price <> products.list_price

    );

    AR:

    $\pi_{products.*}(\sigma_{order\_items.list\_price<>products.list\_price}(products \bowtie order\_items))$

26. Seleccionar Descuento mínimo, máximo y promedio que se hace de los items vendidos

    SQL:

    SELECT min(discount), max(discount), avg(discount) FROM ORDER_ITEMS;

27. Seleccionar Descuento mínimo, máximo y promedio que se hace de los items vendidos a personas de New York

    SQL:

    SELECT min(discount), max(discount), avg(discount) FROM ORDER_ITEMS, ORDERS, CUSTOMERS WHERE

    order_items.order_id=orders.order_id AND orders.customer_id=customers.customer_id AND customers.city='New York';

28. Seleccionar marcas que tengan algún producto que no esté en stock

    SQL:

    SELECT brands.* FROM BRANDS, PRODUCTS WHERE

    brands.brand_id=products.brand_id AND NOT

    EXISTS(

    SELECT * FROM stocks WHERE stocks.product_id=products.product_id

    );

    AR:

    $\pi_{brands.*}((\pi_{product\_id}(products) - \pi_{product\_id}(stocks)) \bowtie products \bowtie brands)$

29. Seleccionar categorías que no tengan productos que no estén en stock (PISTA: Usar 3 consultas anidadas haciendo uso de NOT EXISTS)

    SQL:

    SELECT categories.* FROM CATEGORIES WHERE NOT EXISTS(

    SELECT * FROM products WHERE categories.category_id=products.category_id AND NOT EXISTS(

    SELECT * FROM stocks WHERE stocks.product_id=products.product_id

    )) ;

30. Seleccionar nombres de productos que no hayan sido vendidos

    SQL:

    SELECT PRODUCT_NAME FROM PRODUCTS WHERE NOT EXISTS
    (

    SELECT * FROM ORDER_ITEMS WHERE

    ORDER_ITEMS.PRODUCT_ID = PRODUCTS.PRODUCT_ID

    );

    AR:

    $\pi_{product\_name}((\pi_{product\_id}(products) - \pi_{product\_id}(order\_items)) \bowtie products)$

31. Seleccionar nombre de clientes que hayan comprado la máxima cantidad de algún producto, junto con el nombre del producto

    SQL:

    SELECT customers.first_name products.product_name FROM CUSTOMERS, ORDER_ITEMS, ORDERS, PRODUCTS WHERE

    order_items.order_id=orders.order_id AND orders.customer_id=customers.customer_id AND order_items.product_id=products.product_id AND order_items>=ALL
    (

    SELECT o2.quantity FROM order_items o2);

    AR:

    $\rho(order\_items) = o2$

    $AUX = order\_items - \pi_{o2.*}(\sigma_{order\_items.quantity > o2.quantity}(order\_items \times o2))$

    $\pi_{customers.first\_nameproducts.product\_name}(customers \bowtie orders \bowtie AUX \bowtie products)$

32. Seleccionar nombre y apellidos de personal que no haya vendido nada en 2016 (es decir, que no haya procesado pedidos en ese año)

    SQL:

    (SELECT first_name, last_name FROM STAFFS)

MINUS

(SELECT first_name, last_name FROM STAFFS, ORDERS WHERE staffs.staff_id = orders.staff_id AND orders.order_date BETWEEN (TO_DATE('2016/01/01','yyyy/mm/dd')) AND (TO_DATE('2016/12/31','yyyy/mm/dd'))

)

AR:

$\pi_{first\_name,last\_name}(staffs) -$

$\pi_{first\_name,last\_name}(staffs \bowtie \sigma_{order\_date>='1/1/16' \wedge order\_date<='31/12/16'}(orders))$

33. Seleccionar nombres de tiendas y clientes que sean de la misma ciudad

SQL:

SELECT stores.store_name, customers.first_name FROM STORES, CUSTOMERS WHERE

stores.city = customers.city;

AR:

$\sigma_{customers.city=stores.city}(customers \times stores)$

34. Seleccionar nombres de tiendas y clientes que sean de la misma ciudad, y donde los clientes hayan hecho pedidos a otras tiendas que no sean esas

SQL:

SELECT stores.store_name, customers.first_name FROM STORES, CUSTOMERS, ORDERS WHERE

stores.city = customers.city AND customers.customer_id=orders.customer_id AND orders.store_id <> stores.store_id

AR:

$\pi_{store\_name,first\_name}(\sigma_{customers.city=stores.city \wedge}$

$customers.customer\_id=orders.customer\_id \wedge$

$orders.store\_id<>stores.store\_id(customers \times stores \times orders))$

35. Seleccionar nombres de clientes y su ciudad que hayan comprado los productos más caros.

SQL:

SELECT first_name, city FROM CUSTOMERS, ORDERS, ORDER_ITEMS WHERE

customers.customer_id = orders.customer_id AND orders.order_id = order_items.order_id AND product_id IN (

SELECT product_id FROM PRODUCTS WHERE list_price>=ALL(SELECT P2.list_price FROM PRODUCTS P2);

)

AR:

$\pi_{first\_name,city}(customers \bowtie orders \bowtie order\_items \bowtie (\pi_{product\_id}(products) - \pi_{p2.product\_id}(\sigma_{products.list\_price > p2.list\_price}(products \times p2))))$

36. Nombre e ID de los productos que no hayan sido vendidos, usando operadores conjuntistas

SQL:

SELECT product_name, product_id FROM products WHERE

products.product_id IN(

(SELECT p2.product_id FROM products p2)

MINUS

(SELECT product_id FROM order_items)

);

AR:

$\pi_{product\_name,product\_id}((\pi_{product\_id}(products) - \pi_{product\_id}(order\_items)) \bowtie products)$

37. Selección del nombre y apellidos de clientes que vivan en una ciudad que no tiene tienda, usando operadores conjuntistas

SQL:

SELECT first_name, last_name FROM customers WHERE city IN (

(SELECT city FROM customers)

MINUS

(SELECT city FROM stores)

);

AR:

$AUX = \pi_{city}(customers) - \pi_{city}(stores)$

$\pi_{first\_name,last\_name}(\sigma_{customers.city <> AUX.city}(customers \times AUX))$

38. Selección del nombre y apellidos de clientes que vivan en una ciudad que tiene tienda

SQL:

SELECT first_name, last_name FROM customers WHERE city IN (

SELECT city FROM stores

);

AR:

$\pi_{first\_name,last\_name}(\sigma_{customers.city = stores.city}(customers \times stores))$

39. Selección del email de las tiendas y de su personal

SQL:

(SELECT email FROM stores)

UNION

(SELECT email from staffs);

AR:

$\pi_{email}(stores) \bigcup \pi_{email}(staffs)$

40. Con operadores conjuntistas: Seleccionar nombre de tiendas que tengan en stock productos de al menos dos marcas distintas

SQL:

SELECT store_name FROM stores, stocks, stocks s2, products, products p2 WHERE

stores.store_id = stocks.store_id AND stores.store_id = s2.store_id AND stocks.product_id=products.product_id AND s2.product_id=p2.product_id AND products.brand_id <> p2.brand_id

AR:

$\rho(stocks) = s2; \rho(products) = p2$

$\pi_{store\_name}(\sigma_{stores.store\_id=stocks.store\_id \wedge stores.store\_id=s2.store\_id \wedge}$

$_{stocks.product\_id=products.product\_id \wedge s2.product\_id=p2.product\_id \wedge}$

$_{products.brand\_id<>p2.brand\_id}(stores \times stocks \times s2 \times products \times p2))$

41. Con operadores conjuntistas: Seleccionar nombre y teléfono de clientes que sean de New York o de Bronx

SQL:

(SELECT first_name, phone FROM customers WHERE city='New York')

UNION

(SELECT first_name, phone FROM customers WHERE city='Bronx')

AR:

$\pi_{first\_name,phone}(\sigma_{city='NewYork'}(customers)) \bigcup$

$\pi_{first\_name,phone}(\sigma_{city='Bronx'}(customers))$

42. Con operadores conjuntistas: Encontrar el producto de precio máximo USANDO OPERADORES CONJUNTISTAS (sin usar operadores adicionales como min,max,all, etc.)

SQL: A todos los productos se le quitan aquellos que tienen algún otro producto de precio superior

(SELECT product_id, list_price FROM products)

MINUS

(SELECT p2.product_id, p2.list_price FROM products, products p2 WHERE products.list_price > p2.list_price)

AR:

$\pi_{product\_id,list\_price}(products) - \pi_{p2.product\_id,p2.list\_price}(\sigma_{products.list\_price>p2.list\_price}(products \times p2))$

43. Con operadores conjuntistas: Encontrar los nombres de productos que no estén en stock USANDO OPERADORES CONJUNTISTAS

SQL:

SELECT product_name FROM Products p2 WHERE p2.product_id IN

((SELECT product_id FROM PRODUCTS)

MINUS

(SELECT product_id FROM STOCKS))

AR:

$\pi_{product\_name}((\pi_{product\_id}(products) - \pi_{product\_id}(stocks)) \bowtie products)$

44. Con operadores conjuntistas: Encontrar los nombres de productos que hayan sido vendidos por al menos dos tiendas distintas

SQL:

SELECT product_name FROM products, order_items, order_items oi2, orders, orders o2 WHERE

products.product_id = order_items.product_id AND products.product_id = oi2.product_id AND order_items.order_id = orders.order_id AND oi2.order_id = o2.order_id AND o2.store_id <> orders.store_id

AR: $\rho(order\_items) = oi2; \rho(orders) = o2$

$\pi_{product\_name}(\sigma_{products.product\_id=order\_items.product\_id \wedge products.product\_id=oi2.product\_id \wedge order\_items.order\_id=orders.order\_id \wedge oi2.order\_id=o2.order\_id \wedge o2.store\_id<>orders.store\_id}(products \times order\_items \times oi2 \times orders \times o2))$

1.2. División

1. Marcas que tengan productos de todas las categorías

SQL (AR):

SELECT brand_id FROM BRANDS WHERE NOT EXISTS(

(SELECT category_id FROM CATEGORIES)

MINUS

(SELECT category_id FROM PRODUCTS WHERE PRODUCTS.brand_id=BRANDS.brand_id)

);

SQL (CRT):

SELECT BRAND_ID FROM BRANDS WHERE NOT EXISTS(

SELECT category_id FROM CATEGORIES WHERE NOT EXISTS

(SELECT * FROM PRODUCTS WHERE PRODUCTS.BRAND_ID=BRANDS.BRAND_ID AND PRODUCTS.CATEGORY_ID=CATEGORIES.CATEGORY_ID)

);

AR:

$\pi_{brand\_id,category\_id}(products) \div \pi_{category\_id}(categories)$

2. Categorías que tengan productos de todas las marcas

   SQL (AR):

   SELECT category_id FROM CATEGORIES WHERE NOT EXISTS(

   (SELECT brand_id FROM BRANDS)

   MINUS

   (SELECT brand_id FROM PRODUCTS WHERE products.category_id=categories.category_id)

   );

   SQL (CRT):

   SELECT category_id FROM CATEGORIES WHERE NOT EXISTS(

   SELECT brand_id FROM BRANDS WHERE NOT EXISTS(

   SELECT * FROM PRODUCTS WHERE

   products.category_id=categories.category_id AND products.brand_id=brands.brand_id)

   );

   AR:

   $\pi_{category\_id,brand\_id}(products) \div \pi_{brand\_id}(brands)$

3. Clientes que hayan realizado pedidos en todas las tiendas

   SQL (AR):

   SELECT customer_id FROM customers WHERE NOT EXISTS(

   (SELECT store_id FROM stores)

   MINUS

   (SELECT store_id FROM orders WHERE orders.customer_id = customers.customer_id));

   SQL (CRT): Seleccionar clientes para los que no exista una tienda en la que no haya hecho (no exista) pedidos

   SELECT customer_id FROM customers WHERE NOT EXISTS(

   SELECT store_id FROM stores WHERE NOT EXISTS(

   SELECT * FROM ORDERS WHERE orders.customer_id = customers.customer_id AND orders.store_id=stores.store_id)

);

AR:

$\pi_{customer\_id,store\_id}(orders) \div \pi_{store\_id}(stores)$

4. Tiendas (id y nombre) que hayan vendido todos los productos de la marca Strider

   SQL (AR):

   SELECT store_id, store_name FROM stores WHERE NOT EXISTS(

   (SELECT product_id FROM PRODUCTS p, BRANDS b WHERE p.brand_id=b.brand_id AND b.brand_name='Strider')

   MINUS

   (SELECT order_items.product_id FROM ORDER_ITEMS, ORDERS WHERE order_items.order_id=orders.order_id AND orders.store_id = stores.store_id)

   );

   SQL (CRT): Tiendas para las que no exista productos de la marca Strider que no se hayan vendido

   SELECT store_id, store_name FROM stores WHERE NOT EXISTS(

   SELECT * FROM products,brands WHERE products.brand_id=brands.brand_id AND brand_name='Strider' AND NOT EXISTS(

   SELECT * FROM order_items, orders WHERE order_items.order_id=orders.order_id AND order_items.product_id = products.product_id AND orders.store_id = stores.store_id)

   );

   AR:

   $\pi_{stores.store\_id,store\_name,product\_id}(stores \bowtie orders \bowtie order\_items) \div \pi_{product\_id}(products \bowtie \sigma_{brand\_name='Strider'}(brands))$

5. Tiendas que hayan vendido productos de todas las categorías

   SQL (AR):

   SELECT store_id FROM stores WHERE NOT EXISTS(

   (SELECT category_id FROM categories)

   MINUS

   (SELECT category_id FROM products,order_items,orders WHERE order_items.product_id=products.product_id AND order_items.order_id = orders.order_id AND orders.store_id = stores.store_id)

   );

   SQL (CRT):

   SELECT store_id FROM stores WHERE NOT EXISTS(

   SELECT category_id FROM categories WHERE NOT EXISTS(

SELECT * FROM products,order_items,orders WHERE

order_items.product_id=products.product_id AND order_items.order_id = orders.order_id AND orders.store_id = stores.store_id));

AR:

$\pi_{stores.store\_id,category\_id}(stores \bowtie orders \bowtie order\_items \bowtie products) \div \pi_{category\_id}(categories)$

6. Tiendas que hayan vendido productos de todas las marcas

   SQL (AR):

   SELECT store_id FROM STORES WHERE NOT EXISTS(

   (SELECT brand_id FROM BRANDS)

   MINUS

   (SELECT brand_id FROM PRODUCTS, ORDERS, ORDER_ITEMS WHERE order_items.product_id=products.product_id AND order_items.order_id = orders.order_id AND orders.store_id = stores.store_id));

   SQL (CRT): Tiendas para las que no existen marcas que no se hayan vendido (por la tienda)

   SELECT store_id FROM stores WHERE NOT EXISTS(

   SELECT * from brands WHERE NOT EXISTS(

   SELECT * from orders, order_items, products WHERE orders.order_id= order_items.order_id AND order_items.product_id = products.product_id AND

   orders.store_id = stores.store_id AND products.brand_id=brands.brand_id)

   );

   SQL (AR):

   $\pi_{stores.store\_id,brand\_id}(stores \bowtie orders \bowtie order\_items \bowtie products) \div \pi_{brand\_id}(brands)$

7. Categorías que hayan sido vendidas por todas las tiendas

   SQL (AR):

   SELECT category_name FROM CATEGORIES WHERE NOT EXISTS(

   (SELECT store_id FROM STORES)

   MINUS

   (SELECT store_id FROM ORDERS, ORDER_ITEMS, PRODUCTS WHERE orders.order_id= order_items.order_id AND order_items.product_id = products.product_id AND products.category_id=categories.category_id));

   SQL (CRT):

   SELECT category_name FROM CATEGORIES WHERE NOT EXISTS(

SELECT store_id FROM STORES WHERE NOT EXISTS(
SELECT * FROM ORDERS, ORDER_ITEMS, PRODUCTS WHERE orders.order_id= order_items.order_id AND order_items.product_id = products.product_id AND
orders.store_id = stores.store_id AND products.category_id=categories.category_id)
);

AR:

$\pi_{category\_id,store\_id}(orders \bowtie order\_items \bowtie products) \div \pi_{store\_id}(stores)$

2. Consultas sobre la Base de Datos de agentes

1. Conocer el nombre de los clientes cuyo agente es del mismo área (working area)

   SQL:

   SELECT cliente.NAME FROM cliente, agentes WHERE
   cliente.agent_code=agentes.code AND cliente.working_area=agentes.working_area;

   AR:

   $\pi_{cliente.name}(\sigma_{cliente.working\_area=agentes.working\_area}(cliente \bowtie agentes))$

2. Agentes con la comisión más alta

   SQL:

   SELECT NAME FROM AGENTES WHERE
   COMMISSION>=ALL(
   SELECT COMMISSION FROM AGENTES
   );

   AR:

   $\rho(agentes) = ag$

   $\pi_{name}(\pi_{ag.agent\_code,ag.name}(agentes) -$
   $\pi_{ag.agent\_code,ag.name}(\sigma_{agentes.commission>ag.commision}(agentes \times ag)))$

3. Nombres de clientes de agentes con la comisión más baja

   SQL:

   SELECT NAME FROM AGENTES WHERE
   COMMISSION<=ALL(
   SELECT COMMISSION FROM AGENTES
   );

   AR:

   $\rho(agentes) = ag$

$\pi_{name}(\pi_{ag.agent\_code,ag.name}(agentes) -$

$\pi_{ag.agent\_code,ag.name}(\sigma_{agentes.commission<ag.commision}(agentes \times ag)))$

4. Ciudad de clientes que realizaron los pedidos más recientes

   SQL:

   SELECT CITY, PED_DATE FROM CLIENTE, PEDIDOS WHERE

   CLIENTE.CODE=PEDIDOS.CLIENT_CODE AND PEDIDOS.PED_DATE >=ALL (

   SELECT PED_DATE FROM PEDIDOS

   );

   AR:

   $\rho(pedidos) = pe$

   $\pi_{city}(cliente \bowtie (\pi_{num,client\_code}(pedidos) -$

   $\pi_{pe.num,pe.client\_code}(\sigma_{pedidos.ped\_date>pe.ped\_date}(pedidos \times pe)))))$

5. Agentes que son responsables de los pedidos de todos los clientes de su misma área

   SQL:

   SELECT * FROM agentes WHERE NOT EXISTS(

   SELECT * FROM cliente WHERE

   cliente.WORKING_AREA=agentes.WORKING_AREA AND NOT EXISTS(

   SELECT * FROM PEDIDOS WHERE

   PEDIDOS.CLIENT_CODE=cliente.CLIENT_CODE AND

   PEDIDOS.AGENT_CODE=agentes.AGENT_CODE))

6. Nombres de Clientes y de agentes, junto con el país, que son del mismo país

   SQL:

   SELECT CLIENTE.NAME, AGENTES.NAME, CLIENTE.COUNTRY FROM CLIENTE, AGENTES WHERE

   AGENTES.COUNTRY=CLIENTE.COUNTRY;

   AR:

   $\pi_{CLIENTE.NAME,AGENTES.NAME,CLIENTE.COUNTRY}($

   $\sigma_{AGENTES.COUNTRY=CLIENTE.COUNTRY}(cliente \times agente))$

7. Listado que contenga nombres de clientes de máxima calificación (grade) y de mínima calificación

   SQL:

   (SELECT NAME FROM CLIENTE WHERE GRADE>=ALL(

SELECT GRADE FROM CLIENTE)

)

UNION

(SELECT NAME FROM CLIENTE WHERE GRADE<=ALL(

SELECT GRADE FROM CLIENTE)

)

AR:

$\rho(cliente) = c2$

$MINIMO = cliente - \pi_{c2.*}(\sigma_{cliente.grade<c2.grade}(cliente \times c2))$

$MAXIMO = cliente - \pi_{c2.*}(\sigma_{cliente.grade>c2.grade}(cliente \times c2))$

$\pi_{name}(MINIMO) \bigcup \pi_{name}(MAXIMO)$

8. Nombre del cliente que hizo el pedido más antiguo

SQL:

SELECT CLIENTE.NAME FROM CLIENTE, PEDIDOS WHERE

CLIENTE.CODE=PEDIDOS.CLIENT_CODE AND PEDIDOS.PED_DATE

<=ALL (SELECT PED_DATE FROM PEDIDOS);

AR:

$\rho(pedidos) = p2$

$\pi_{name}(cliente \bowtie (\pi_{num}(pedidos) - \pi_{p2.num}(\sigma_{pedidos.num<p2.num}(pedidos \times p2))))$

9. Nombre de agentes que sólo tienen un único cliente

SQL:

SELECT AGENTES.NAME FROM AGENTES, CLIENTE WHERE

AGENTES.CODE = CLIENTE.AGENT_CODE AND NOT EXISTS(

SELECT * FROM CLIENTE C2 WHERE AGENTES.CODE = C2.AGENT_CODE AND C2.CODE<>CLIENTE.CODE

);

AR: La idea es, en una tabla producto cartesiano involutivo de agentes, quedarse con las filas del mismo agente con clientes distintos (son los agentes con más de un cliente). Luego, quitarle a todos los agentes esos que tienen más de un cliente.

$\rho(agentes) = a2$

$\pi_{name}(agentes - \pi_{agentes.*}(\sigma_{agentes.client\_code<>a2.client\_code \wedge agentes.code=a2.code}(agentes \times a2)))$

10. Nombre de agentes que no fueron responsables de ningún pedido en agosto de 2008

    SQL:

    SELECT NAME FROM AGENTES WHERE NOT EXISTS(

    SELECT * FROM PEDIDOS WHERE

    PED_DATE>=TO_DATE('2008/08/01', 'yyyy/mm/dd') AND

    PED_DATE<=TO_DATE('2008/08/31', 'yyyy/mm/dd') AND

    AGENTES.CODE=PEDIDOS.AGENT_CODE);

    AR: La idea es quitarle a todos los agentes que hicieron pedidos aquellos que tuvieron algún pedido en 2008. Luego unir con agentes para coger el nombre.

    $\pi_{name}(agentes \bowtie (\pi_{code}(agentes) - \pi_{agent\_code}(\sigma_{ped\_date \geq 1/1/2008 \wedge ped\_date \leq 21/12/2008}(pedidos))))$

3. Sentencias de la liga de baloncesto

1. Listar el encuentro más antiguo

   SQL:

   select * from encuentro where fecha<=ALL(

   select fecha FROM encuentro)

   AR:

   $\rho(encuentro) = e2$

   $encuentro - \pi_{e2.*}(\sigma_{encuentro.fecha < e2.fecha}(encuentro \times e2))$

2. Listar el encuentro más reciente

   SQL:

   select * from encuentro where fecha>=ALL(

   select fecha FROM encuentro)

   AR:

   $\rho(encuentro) = e2$

   $encuentro - \pi_{e2.*}(\sigma_{encuentro.fecha > e2.fecha}(encuentro \times e2))$

3. Listar el nombre de los equipos que hayan jugado el encuentro más antiguo, junto con la fecha del encuentro

   SQL:

   select equipos.nombreE, e2.nombreE, fecha from encuentro,equipos,equipos e2 where

   equipos.codE=encuentro.eq1 AND e2.codE=encuentro.eq2 AND fecha<=ALL(

   select fecha FROM encuentro)

AR:

$\rho(encuentro) = e2$

$\rho(equipos) = equ2$

$ANTIGUO = encuentro - \pi_{e2.*}(\sigma_{encuentro.fecha>e2.fecha}(encuentro \times e2))$

$\pi_{equipo.nombreE,equ2.nombreE,fecha}(\sigma_{equipo.codE=ANTIGUO.eq1 \wedge equ2.codE=ANTIGUO.eq2}(equipo \times equ2 \times ANTIGUO))$

4. Para cada equipo, mostrar el número de veces que ha ganado como local (PISTA: Usando GROUP BY)

   SQL:

   SELECT codE, count(*) FROM equipos,encuentro WHERE

   equipos.codE.eq1=equipos.codE AND res1>res2

   GROUP BY codE

5. Calcular los puntos de cada equipo, sabiendo que los puntos se calculan como 3*partidosGanados (PISTA: Usando GROUP BY)

   SQL:

   SELECT codE, 3*count(*) FROM equipos,encuentro WHERE

   (equipos.codE.eq1=equipos.codE AND res1>res2) OR (equipos.codE.eq2=equipos.codE AND res1<res2)

   GROUP BY codE

6. Calcular los puntos de cada equipo, sabiendo que los puntos se calculan como 3*partidosGanados. Mostrar sólo los que tengan más de 3 puntos (PISTA: Usando GROUP BY y HAVING)

   SELECT codE, 3*count(*) FROM equipos,encuentro WHERE

   (equipos.codE.eq1=equipos.codE AND res1>res2) OR (equipos.codE.eq2=equipos.codE AND res1<res2)

   GROUP BY codE

   HAVING 3*count(*)¿3

7. Calcular los equipos que no hayan perdido ningún partido como local (es decir, que hayan ganado o empatado todos los partidos como local)

   SQL:

   SELECT * FROM equipos WHERE NOT EXISTS(

   SELECT * FROM encuentros WHERE

   encuentros.eq1=equipos.codE AND res1<res2

)

   AR:

   $\pi_{codE}(equipos) - \pi_{eq1}(\sigma_{res1<res2}(encuentros))$

8. Nombre del equipo cuyo jugador haya hecho el máximo número de faltas en algún encuentro

   SQL:

   SELECT * FROM jugadores,faltas WHERE

   jugadores.codJ=faltas.codJ AND num>=all(

   SELECT num from faltas)

   AR:

   $\rho(faltas) = f2$

   $\pi_{jugadores.*}(jugadores \bowtie (\pi_{codJ}(faltas) - \pi_{f2.codJ}(\sigma_{faltas.num>f2.num}(faltas \times f2))))$

9. Calcular el total faltas realizadas en cada encuentro

   SQL:

   select eq1,eq2, sum(num) FROM faltas

   GROUP BY eq1, eq2

10. Jugadores que no hayan hecho ninguna falta en toda la liga

    SQL:

    SELECT jugadores.* FROM jugadores WHERE NOT EXISTS(

    SELECT * FROM faltas WHERE faltas.codJ=jugadores.codJ AND num>0

)

    AR:

    $\pi_{codJ}(jugadores) - \pi_{codJ}(\sigma_{num>0}(faltas))$