1. Consider the following relation instance r of the relational schema $R(A, B, C, D)$.

R

A	B	C	D
0	0	0	1
2	1	2	0
1	1	2	0
0	0	1	2

(a) Assuming that r is a *valid* relation instance of R, write down all the *possible* superkeys of R.

(b) Additionally, suppose that it is also known that $\{A, C\}$ is *definitely* a superkey of R. Based on the additional information, write down all the *possible* candidate keys of R. Which of these (*if any*) must be the candidate key of R?

Solution: In this question, we are mainly *inferring* the properties of the relation schema from an instance instead of *enforcing* them. We start with the assumption that the relation instance r is valid, otherwise there are no solution to this. We then look for *violations* by considering all subset of attributes of R and performing the check below. We denote the subset of attributes of interest as a and we let $b = R - a$ (*i.e.*, the set of attributes in R but not in a).

> If there are rows where all values in a are the same, but the values in b are not the same then a is not a superkey.

(a) The possible superkeys of R are $\{A, C\}$, $\{A, D\}$, $\{A, B, C\}$, $\{A, B, D\}$, $\{A, C, D\}$, and $\{A, B, C, D\}$.

(b) If $\{A, C\}$ is indeed a superkey of R, then $\{A, C\}$ must also be a candidate key of R since neither $\{A\}$ nor $\{C\}$ is a superkey of R (*based on* r). Furthermore, any superkey which is a superset of $\{A, C\}$ cannot be candidate key of R since they include $\{A, C\}$ and therefore not minimal.

The remaining two *possible* superkeys are $\{A, D\}$ and $\{A, B, D\}$. Both must be *possible* candidate keys of R since we do not know if $\{A, D\}$ is really a superkey of R or not. If $\{A, D\}$ is not a superkey then $\{A, B, D\}$ is a candidate key. Hence, we cannot discount the possibility that $\{A, B, D\}$ is a *possible* candidate key.

2. Consider a relational database consisting of two relations with schema $R(A, B)$ and $S(W, X, Y, Z)$ such that A is the primary key of R and W is the primary key of S (*for future use, we denote this with* $R(\underline{A}, B)$ *and* $S(\underline{W}, X, Y, Z)$ *where the attributes being underlined are parts of primary key*). Let r and s be the current instances of R and S, respectively, as shown below.

R

A	B
3	0
2	1
1	1
0	0

S

W	X	Y	Z
0	4	0	NULL
1	NULL	2	NULL
2	1	2	NULL
3	0	1	NULL

Based on the current database instance above, write down all the *possible* foreign keys in S that refer to attribute A in R.

Solution: Recap that the definition of *foreign key* requires only one of the following:

1. The value exists in the attributes being referenced.
2. The value is `NULL`.

Similar to the previous question, we are *inferring* for possibilities. As such, we are looking for violations to discount the possibilities. Therefore, the *possible* foreign keys are W, Y, and Z.

3. Two queries Q_1 and Q_2 on a relational database with schema D are defined to be **equivalent queries** (denoted by $Q_1 \equiv Q_2$) if for *every* valid instance d of D, both Q_1 and Q_2 always compute the same results on d.

   Consider a database with the following relational schema: $R(\underline{A}, C)$, $S(\underline{A}, D)$, and $T(\underline{X}, Y)$, with primary key attributes underlined. Assume all the attributes have integer domain. For each of the following pairs of queries Q_1 and Q_2, state whether or not $Q_1 \equiv Q_2$.

   (a) $Q_1 = \pi_A(\sigma_{A<10}(R))$ **and** $Q_2 = \sigma_{A<10}(\pi_A(R))$
   (b) $Q_1 = \pi_A(\sigma_{C<10}(R))$ **and** $Q_2 = \sigma_{C<10}(\pi_A(R))$
   (c) $Q_1 = \pi_{D,Y}(S \times T)$ **and** $Q_2 = \pi_D(S) \times \pi_Y(T)$
   (d) $Q_1 = \pi_{D,Y}(S \times T)$ **and** $Q_2 = \pi_{D,Y}(T \times S)$
   (e) $Q_1 = (R \times \pi_D(S)) \times T$ **and** $Q_2 = R \times (\pi_D(S) \times T)$
   (f) $Q_1 = \pi_A(R \cup S)$ **and** $Q_2 = \pi_A(R) \cup \pi_A(S)$
   (g) $Q_1 = \pi_A(R - S)$ **and** $Q_2 = \pi_A(R) - \pi_A(S)$

Solution:

(a) **Equivalent:** This is because it does not matter if you do projection on A first or selection on A first, the result will have all rows where $A \geq 10$ removed *and* only contains attributes A.

(b) **Not equivalent:** This is because the second relational algebra is *invalid* since the selection condition refers to a non-existent attribute. Even if it is not invalid, then the condition is *trivially* satisfied. We choose to make this invalid instead.

(c) **Equivalent:** The result will have the same attributes $\{D, Y\}$ in the exact same order (*i.e.*., (D, Y)). Furthermore, in both cases, for every element in S, it will be paired with an element in T. After removing duplicates (either after Cartesian product as in Q_1 or after Cartesian product as in Q_2), the remaining elements will be equivalent.

(d) **Equivalent:** The intermediate result is *not equivalent* (*i.e.*, $S \times T \not\equiv T \times S$), but after rearranging the attributes, they will be equivalent.

(e) **Equivalent:** Since we treat Cartesian product as *associative* by virtue of *canonical isomorphism*.

(f) **Equivalent:** Similar reasoning to part (c).

(g) **Not equivalent:** Consider $r = \{(10, 10)\}$ and $s = \{(10, 20)\}$ where r is an instance of R and s is an instance of S.

Then

$$\begin{aligned} Q_1 &= \pi_A(R - S) \\ &= \pi_A(\{(10, 10)\} - \{(10, 20)\}) \\ &= \pi_A(\{(10, 10)\}) = \{(10)\} \end{aligned}$$

On the other hand

$$\begin{aligned} Q_2 &= \pi_A(R) - \pi_A(S) \\ &= \pi_A(\{(10, 10)\}) - \pi_A(\{(10, 20)\}) \\ &= \{(10)\} - \{(10)\} = \{\} \end{aligned}$$

Hence, $Q_1 \not\equiv Q_2$.

4. Consider the following relational database schema ~~discussed in lecture~~ given in `T01.sql`. We further add the primary keys of each relation in underline.

   - `Pizza(pizza)`: *All the pizzas of interest.*
   - `Customers(cname, area)`: *The name and location of each customer.*
   - `Restaurants(rname, area)`: *The name and location of each restaurant.*
   - `Contains(pizza, ingredient)`: *The ingredients used in each pizza.*
   - `Sells(rname, pizza, price)`: *Pizzas sold by restaurants and the prices.*
   - `Likes(cname, pizza)`: *Pizzas that customers like.*

   Additionally, we have the following foreign key constraints on the database schema:

   - `Contains.pizza` is a foreign key that refers to `Pizzas.pizza`
   - `Sells.rname` is a foreign key that refers to `Restaurants.rname`
   - `Sells.pizza` is a foreign key that refers to `Pizzas.pizza`
   - `Likes.cname` is a foreign key that refers to `Customers.cname`
   - `Likes.pizza` is a foreign key that refers to `Pizzas.pizza`

   Answer each of the following queries using relational algebra.

   (a) Find all pizzas that Alice likes but is not liked by Bob[1].

   (b) Find all customer-restaurant pairs (C, R) where C and R both located in the same area and C likes some pizza that is sold by R.

   (c) Suppose the relation `Likes` contains all information about all customers. In other words, if the pair (`cname`, `pizza`) is not in the relation `likes`, it means that the customer `cname` *dislikes* the pizza `pizza`. Write a relational algebra expression to find for all customers, the pizza that they dislike. The result should be of the form (`cname`, `pizza`).

   (d) Consider having a relation `Dislikes(cname, pizza)` that are created based on the query from Part (c). Find all customer pairs $(C1, C2)$ such that $C1$ likes some pizza that $C2$ does not like.

   (e) Find all customer pairs $(C1, C2)$ such that $C1 < C2$ and they like *exactly* the same pizzas. You may assume that you have a relation `LikesDislikes(C1,C2)` that are created based on the query from Part (d). *Exclude* pairs of customers who do not like any pizza.

   (f) For each restaurant, find the price of the most expensive pizzas sold by that restaurant. Exclude restaurants that do not sell any pizza.

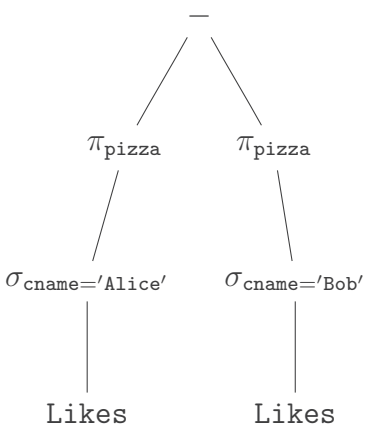
   (g) Find all customer-pizza pairs (C, P) where the pizza P sold by some restaurant that is located in the same area as that of the customer C. Include customers whose associated set of pizzas is empty.

[1] The intention is to state it as *"all pizzas that Alice likes but Bob does not like"*, however, this may indicate dislike which we have not discussed the underlying assumption yet.

Solution: Due to the complexity of some of the answers, we split the answers into subqueries. The final answer is always given as Q_{ans}.

(a) Let Q_1 be all pizza that Alice likes and Q_2 be all pizza that Bob likes. Then Q_{ans} is the difference.

$$\begin{aligned} Q_1 &:= \pi_{\texttt{pizza}}(\sigma_{\texttt{cname='Alice'}}(\texttt{Likes})) \\ Q_2 &:= \pi_{\texttt{pizza}}(\sigma_{\texttt{cname='Bob'}}(\texttt{Likes})) \\ Q_{ans} &:= Q_1 - Q_2 \end{aligned}$$



The following answer is incorrect:

$$\begin{aligned} Q_1 &:= \texttt{Likes} \times \rho_{\texttt{cname}\leftarrow\texttt{cname2},\texttt{pizza}\leftarrow\texttt{pizza2}}(\texttt{Likes}) \\ Q_2 &:= \sigma_{(\texttt{pizza=pizza2})\wedge(\texttt{cname='Alice'})\wedge(\texttt{cname2<>'Bob'})}(Q_1) \\ Q_{ans} &:= \pi_{\texttt{pizza}}(Q_2) \end{aligned}$$

The above answer will compute exactly the set of pizzas that Alice likes (*independent of whether Bob likes them*) since for each pizza p that Alice likes, we have $(\texttt{'Alice'}, p, \texttt{'Alice'}, p) \in \texttt{Likes} \times \texttt{Likes}$.

(b) *Solution 1:* We join `Restaurants`, `Sells`, `Likes` and `Customers` tables since we need all these information. Then, we equate the restaurant names, customer names, pizza and the area.

$$\begin{aligned} Q_1 &:= \texttt{Customers} \times \rho_{\texttt{rname}\leftarrow\texttt{rname2},\texttt{area}\leftarrow\texttt{area2}}(\texttt{Restaurants}) \\ &\quad \times \rho_{\texttt{rname}\leftarrow\texttt{rname3},\texttt{pizza}\leftarrow\texttt{pizza3}}(\texttt{Sells}) \\ &\quad \times \rho_{\texttt{cname}\leftarrow\texttt{cname4},\texttt{pizza}\leftarrow\texttt{pizza4}}(\texttt{Likes}) \\ Q_2 &:= \sigma_{(\texttt{area=area2})\wedge(\texttt{rname2=rname3})\wedge(\texttt{cname=cname4})\wedge(\texttt{pizza3=pizza4})}(Q_1) \\ Q_{ans} &:= \pi_{\texttt{cname},\texttt{rname}}(Q_2) \end{aligned}$$

Solution 2: Solution 1 is actually equivalent to natural join of all the tables involved.

$$Q_{ans} = \pi_{\texttt{cname},\texttt{rname}}(\texttt{Customers} \bowtie \texttt{Restaurants} \bowtie \texttt{Sells} \bowtie \texttt{Likes})$$

Solution 3: Another way to think about this is to decompose the problem into two components. Q_1 is the table consisting of all customers and restaurants in the same area. Q_2 is the table consisting of all customers and restaurants such that the restaurant sells some pizza that the customer likes. The answer is the intersection of these two tables.

$$\begin{aligned} Q_1 &:= \pi_{\texttt{cname,rname}}(\texttt{Customers} \bowtie \texttt{Restaurants}) \\ Q_2 &:= \pi_{\texttt{cname,rname}}(\texttt{Sells} \bowtie \texttt{Likes}) \\ Q_{ans} &:= Q_1 \cap Q_2 \end{aligned}$$

(c) Let Q_1 be all customer-pizza pairs (*i.e.*, all possible (C, P)). Then the final answer is $Q_1 - \texttt{Likes}$.

$$\begin{aligned} Q_1 &:= \pi_{\texttt{cname}}(\texttt{Customers}) \times \texttt{pizzas} \\ Q_{ans} &:= Q_1 - \texttt{Likes} \end{aligned}$$

(d) Let Q_1 contains all likes-dislikes pair. We need this to know which customer likes what other customer dislikes. To compute that, we can then equate the pizza that is liked with the pizza that is disliked.

$$\begin{aligned} Q_1 &:= \texttt{Likes} \times \rho_{\texttt{cname}\leftarrow\texttt{cname2},\texttt{pizza}\leftarrow\texttt{pizza2}}(\texttt{Dislikes}) \\ Q_{ans} &:= \pi_{\texttt{cname},\texttt{cname2}}(\sigma_{\texttt{pizza=pizza2}}(Q_1)) \end{aligned}$$

(e) What is `LikesDislikes`? If the pair (`cx`, `cy`) is in `LikesDislikes` then `cx` likes some pizza `cy` dislikes. If we want to have $C1$ likes *exactly* what $C2$ likes, then $(C1, C2)$ must not be in `LikesDislikes`. Therefore, we remove any rows from `LikesDislikes`.

One problem is `LikesDislikes` is *incomplete*. Instead of just removing entries where $C1$ likes some of what $C2$ dislikes, we also need to remove entries where $C2$ likes some of what $C1$ dislikes (*i.e.*, `LikesDislikes` but swap $C1$ and $C2$).

$$\begin{aligned} Q_1 &:= \pi_{\texttt{cname},\texttt{cname2}}(\sigma_{\texttt{cname}<\texttt{cname2}}(\texttt{Likes} \times \rho_{\texttt{cname}\leftarrow\texttt{cname2}}(\texttt{Likes}))) \\ Q_2 &:= \texttt{LikesDislikes} \cup \pi_{\texttt{cname2},\texttt{cname}}(\texttt{LikesDislikes}) \\ Q_{ans} &:= Q_1 - Q_2 \end{aligned}$$

(f) The set $\pi_{\texttt{rname},\texttt{price}}(\texttt{Sells})$ can be partitioned into two disjoint subsets $\pi_{\texttt{rname},\texttt{price}}(\texttt{Sells}) = S_1 \cup S_2$ where S_1 is the set of restaurant name and price pairs (r, pr) such that p is the price of some pizza sold by r that is not the most expensive pizza sold by r.

Then we have S_2 which is the set of restaurant name and price pairs (r, pr) such that p is the price of the most expensive pizza by r. Thus the required answer is given by $S_2 = \pi_{\texttt{rname},\texttt{price}}(\texttt{Sells}) - S_1$. Note that $(r, pr) \in S_1$ if there exists some tuple $(r_2, p_2, pr_2) \in \texttt{Sells}$ where $r_2 = r$ and $pr_2 > pr$.

$$\begin{aligned} Q_1 &:= \texttt{Sells} \times \rho_{\texttt{rname}\leftarrow\texttt{rname2},\texttt{price}\leftarrow\texttt{price2}}(\texttt{Sells}) \\ S_1 &:= \pi_{\texttt{rname},\texttt{price}}(\sigma_{(\texttt{rname=rname2})\wedge(\texttt{price<price2})}(Q_1)) \\ S_2 &:= \pi_{\texttt{rname},\texttt{price}}(\texttt{Sells}) - S_1 \end{aligned}$$

(g) To include customers whose associated set of pizzas is empty, we will need *outer joins*. Here we use natural outer joins since we omit the conditions.

$$\pi_{\texttt{cname,pizza}}(\texttt{Customers} \rightarrowtail\!\!\!\!\bowtie (\texttt{Restaurants} \bowtie \texttt{Sells}))$$

The following answer is *incorrect* (note the missing parentheses):

$$\pi_{\texttt{cname,pizza}}(\texttt{Customers} \rightarrowtail\!\!\!\!\bowtie \texttt{Restaurants} \bowtie \texttt{Sells})$$

The above answer is equivalent to

$$\pi_{\texttt{cname,pizza}}((\texttt{Customers} \rightarrowtail\!\!\!\!\bowtie \texttt{Restaurants}) \bowtie \texttt{Sells})$$

However, outer joins are generally not associative with inner joins. In particular,

$$(\texttt{Customers} \rightarrowtail\!\!\!\!\bowtie \texttt{Restaurants}) \bowtie \texttt{Sells} \not\equiv \texttt{Customers} \rightarrowtail\!\!\!\!\bowtie (\texttt{Restaurants} \bowtie \texttt{Sells})$$

To see this, consider a customer tuple $c \in$ `Customers` who is not co-located with any restaurant. Therefore, c will appear exactly once in the result of `Customers` $\rightarrow$ `Restaurants` with a `NULL` value for attributes `rname`. Due to the `NULL` value for `rname`, this tuple with c will be a dangling tuple w.r.t. the natural join with `Sells`.

In contrast, by performing the outer join *after* the natural join, c will be preserved in the result of `Customers` $\rightarrow$ (`Restaurants` $\bowtie$ `Sells`).

⚠ You should pay attention to the join ordering when using outer joins!

The following is another incorrect answer:

$$\pi_{\texttt{cname,pizza}}(\texttt{Customers} \rightarrowtail\!\!\!\!\bowtie (\texttt{Restaurants} \rightarrowtail\!\!\!\!\bowtie \texttt{Sells}))$$

By using full outer joins for both joins, this answer will also preserve dangling tuples from `Sells` and `Restaurants` relations. Thus, the query result could have tuples with a `NULL` value for `cname` but such tuples are not required by the question.

To see this, consider a tuple $(r, p, pr) \in$ `Sells` where the restaurant r is not co-located with any customer. Thus, (r, p, pr) will join with exactly one tuple from `Restaurants` (say, (r, a)), but the resultant tuple (r, a, p, pr) from the first join computation will not join with any tuple from `Customers`. This will procude $(\texttt{NULL}, p)$ in the query result, which is incorrect.

Moreover, if there is some tuple $r \in$ `Restaurants` where r is not co-located with any customer and r does not sell any pizza, then this will produce $(\texttt{NULL}, \texttt{NULL})$ in the query result, which is also incorrect.

⚠ The type of joins matters: you should use appropriate outer joins to preserve the required dangling tuples only when necessary!

5. Consider the following relational algebra query expressed on the database schema in Question 4.

$$
\begin{aligned}
R_1 &:= \pi_{\texttt{pizza}}(\sigma_{\texttt{cname='Maggie'}}(\texttt{Likes})) \\
R_2 &:= \pi_{\texttt{rname}}(\texttt{Sells}) \times R_1 \\
R_3 &:= \pi_{\texttt{rname}}(R_2 - \pi_{\texttt{rname,pizza}}(\texttt{Sells})) \\
R_4 &:= \pi_{\texttt{rname}}(\texttt{Sells}) - R_3 \\
R_5 &:= \pi_{\texttt{pizza}}(\sigma_{\texttt{cname='Ralph'}}(\texttt{Likes})) \\
R_6 &:= \pi_{\texttt{rname}}(\sigma_{\texttt{pizza5=pizza}}((\texttt{Sells} \times \rho_{\texttt{pizza}\leftarrow\texttt{pizza5}}(R_5)))) \\
R_7 &:= R_4 - R_6
\end{aligned}
$$

For each of the relational algebra expression R_i, write down a concise English sentence to precisely describe the information retrieved by R_i.

Solution:

R_1 : The set of pizzas that Maggie likes.

R_2 : The set of restaurant-pizza pairs (r, p) where r is a restaurant that sells some pizza and p is some pizza that Maggie likes.

R_3 : The set of restaurants that don't sell some pizza that Maggie likes.

R_4 : The set of restaurants that sell all the pizzas that Maggie likes.

R_5 : The set of pizzas that Ralph likes.

R_6 : The set of restaurants that sells some pizza that Ralph likes.

R_7 : The set of restaurants that sell all the pizzas that Maggie likes and don't sell any pizza that Ralph likes.