CSE-3421: Exercises
with answers

1. **Independence**

   Answer #1.2 (page 23) from the textbook: What is *logical data independence* and why is it important?

   A short paragraph is sufficient.

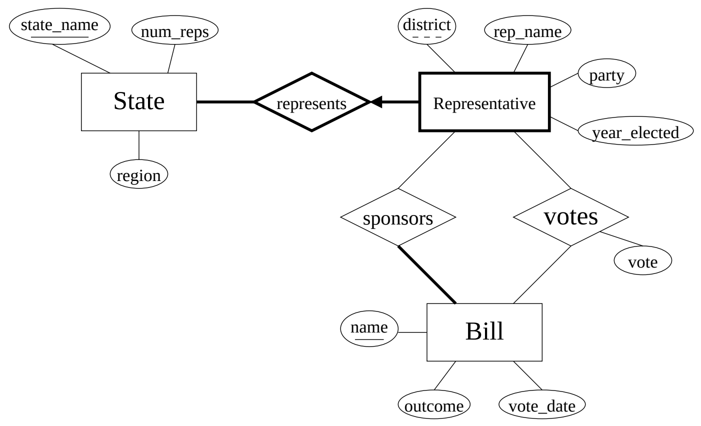
   The textbook defines both the concepts logical data independence *and* physical data independence. *The latter is something more often brought up when listing the merits of relational database systems.*
   The relational model allows for logical data independence in that we can insulate users—and queries and application programs written on a database—from changes made to the database's schema. The view facility allows us to write views *(which act as virtual tables) on the schema. We can then somewhat guarantee that these views will stay the same over time. When we change the logical schema itself (the definitions of the tables that compose the database), we only need to modify the views' definitions accordingly. The changes then are transparent to queries and application programs written beforehand.*
   Relational databases provide then tools for supporting logical data independence, while many other database models (non-relational) do not. This is quite useful in practice. However, the textbook may oversell it. A huge problem with any database is that changing the schema typically does *affect the users, the application programs, and pre-written queries. We would have to have been very judicious as designers to have properly insulated the schema via views in all the ways we should have done in hindsight. That is why it is so important to make a solid design for the database schema initially, so that the changes needed on it over time are fewer.*
   The relational model allows for physical data independence too, which states that how the data is stored is independent from how it is presented logically to the users (and queries and application programs). This "presentation" is simply the (logical) schema. Any changes to how the data is physically organized are transparent to the users under the relation model simply because they can only interact with the database "logically" via the schema (that is, writing SQL with the schema). Changes to the physical database are common, and are often made to improve the database system's performance.

2. **The House Database**

You have just moved to Washington, D.C., to work for the U.S. House of Representatives (the "House") as a database specialist. For your first job, they want you to design a database to keep track of votes taken in the House. (A counterpart to you working for the U.S. Senate is designing a similar database for tracking votes in the Senate.)

The database will track votes taken in the House during the current two-year congressional session. It should record each U.S. state (for instance, Texas) with its name, number of representatives, and *region* in which the state is located (northeast, mid-atlantic, midwest, and so forth). Each congress-creature, er, I mean representative, in the House is to be described by his or her name, the district (by district number) that he or she represents, the year when he or she was first elected, and the political party to which he or she belongs (for instance, *Republican*, *Democrat*, *Independent*, *Green*, *Reform*, *Other*). The database should track each bill (legislation) with its name, the date on which its vote was taken, whether the bill passed or failed (so the domain is *yes* and *no*), and its sponsors (the representatives who proposed the bill). The database should track how each representative voted on each bill (*yes*, *no*, *abstain*, *absent*).

a. Design an entity-relationship (E-R) schema diagram for the above enterprise. Be careful to ensure that each of the attributes would be restricted to *legal* values (no pun...). State clearly any assumptions that you make. Also state any business rules—logic to which the database should adhere—that are not captured in your E-R diagram.



Many students added an arrow from the entity box Reprensentative *to the relationship* votes *to make it a many-one rel-ship instead of a many-many rel-ship. Their argument was that a representative gets* one *vote on a given bill, not more than one. This many-one is the wrong logic, however! The* Bill *entity box represents the collection of all bills that come to the floor of the house. A representative votes on each bill, so he or she votes on many bills. And, of course, each bill is voted on by many representatives. So the rel-ship* votes *is properly many-many.*

But what about the fact that we want to ensure that a representative can only vote once on a given bill? We have it already! That is why it was natural to model votes *as a rel-ship. Any given pair of a particular representative and a particular bill can show up in the rel-ship at most once. This same pair showing up "again" is impossible by the definition of what a rel-ship is. A relationship is a* set *(not a* multi-set*, or* bag*) of these combinations (for* votes*, pairs of representatives and bills).*
There is no easy way to represent via the E-R diagram that we are restricting the values of how *a representative votes to a certain small set of legal values. (Why?) However, in a relational schema, this would be simple to do.*
A representative can be uniquely identified by the state and the district# (within the state) that he or she represents. So it makes since to make the entity Representative *a weak entity on entity* State*. First, a representative cannot exist without a state to represent. Second, the natural key for representative is the state's name plus the district#.*

b. Do you have any derived attributes in your diagram? Would these be problematic in the implemented database?

A derived attribute is any piece of information that we could derive from the other information in the database. Generally, we like to avoid derived attribute because this is a redundancy and so can cause trouble. (What are we to think when the attribute's value and the value that we could derive for it are different?)
*We have several in our design. The number of representatives for a state (*num_reps*) can be computed by counting the number of* Representative *"records" associated with that state. The* outcome *of a bill can be deduced by how many votes in favor it received.*
Relational purists will usually eliminate any derived attributes from the design. Real-world designers will sometimes allow a derived attribute in the design if they feel there is good reason for it. We just should be aware of them and weigh the advantages and disadvantages.

3. **Airplane, the SEQUEL** (no pun…)

   You have become a world-renowned specialist in airline database design. (Nothing like specializing, eh?) Scoundrel Airlines has come to you for help. Your mission, should you decide to accept it, is to design an E-R schema for the Airline's database for flight information and reservations. They provide you the following specifications.

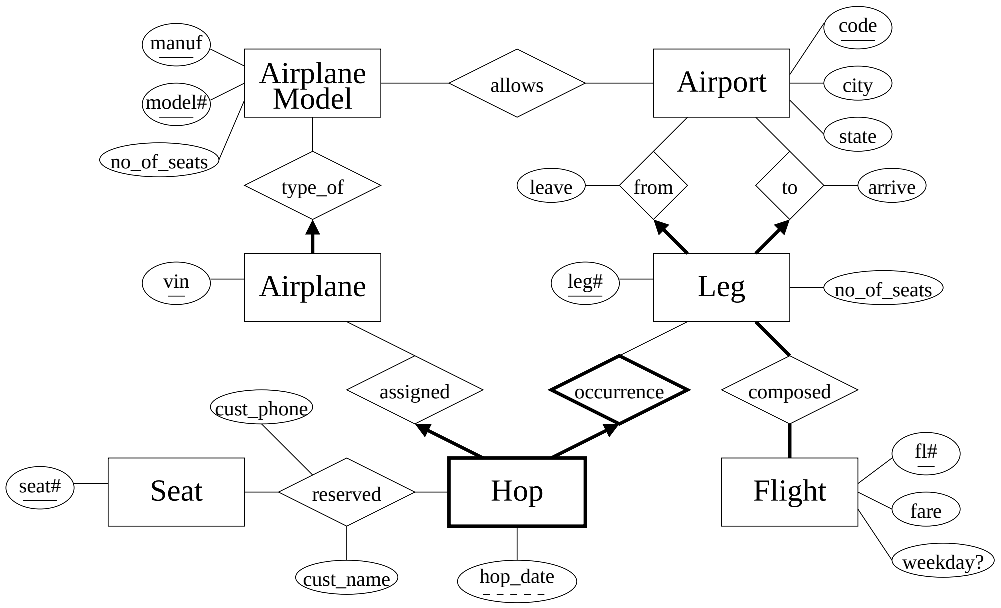
   Scoundrel Airlines wants to keep information on its *airplanes*. They want to record the *airports* that they fly into and out of. Airport information should include the airport's code (such as 'RIC' for the Richmond-Williamsburg International (?) Airport), the name of the airport, the primary city it serves, and in which state or province it is. The database should keep track of the types of airplanes that the Airline owns, capturing the information of the number of seats, the manufacturer, and the model name.

   The database should model the fact that only certain types of airplane can land at certain airports. For instance, jumbo jets can land at RIC, but not at the Newport News-Williamsburg International (???) Airport (PHF). Every airplane is of a given airplane type.

   The database needs to keep track of flights. A *leg* of a trip is denoted by its departure information (from airport X, departure time) and its arrival information (to airport Y, arrival time). A particular airplane is assigned to a given flight leg on a given day. Each flight leg has a number of seats available to be reserved. Each *seat* on the airplane is *reserved* for a given customer, recording his or her name and telephone number.

   A *flight* is made up of a sequence of legs, for which they want to record the flight number, the flight fare, and whether the flight flies on weekdays or not.

   Design an E-R schema diagram to model Scoundrel Airline's database. Again, state clearly any assumptions that you make, and state any rules that are not captured in your E-R diagram.



It is important that we distinguish between a leg*, a schedule entry of a non-stop flight between two airports, and an instance of that actual* leg *occuring on a given day. We named the occurrence of a* leg *a* hop*. It is a* hop *which uses a particular airplane, and for which we sell seats. So we had to build this distinction into our design.*
We have seen this type/instance dichotomy before. A course *is a listing in the course catalog, while a* class *is an instance of a course offerred in a given semester meeting in a given room at a given time and taught by a given instructor. In our movie-rental database, we had* copy *and* movie*. The latter was the catalog entry about the particular movie and the former referred to the physical video tape or DVD with the movie on it. We had to keep track of the copies to know which customers have returned their rentals.*

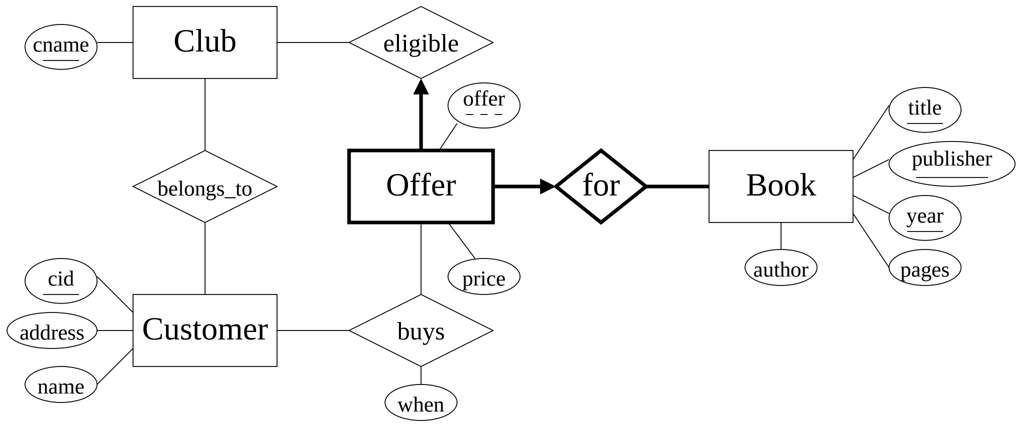


Figure 1: E-R diagram for Question 4.

4. **YRB.com**

   You have just been hired at the famous company **YRB.com** (*York River Books*) which sells books over the Internet. They have been having problems with their database designs and have hired you as a database expert to help fix them.

   Figure 1 shows the core of the E-R diagram that represents **YRB.com**'s main database to track sales of books to customers.

   Customers belong to "clubs", such as *AAA*, *student*, *professor*, *AOL subscriber*, *AARP*, and so forth. There can be several "offers" available for a book which determines its price based upon the clubs to which a customer belongs.

   a. How does one determine the price a given customer paid for a given book? Should an attribute price be added to **buys**?

      Which Offers a Customer has "bought" is recorded by the rel-ship buys. An offer is for a given book, so it is easy to tell the book the customer bought by going through the offer he or she bought it under. We can look up the assciated Offer "record" from the buys entry to see the price the customer paid for the book.
      The customer may have bought the same book several times through different offers each with a different price. So we may not get just one answer.

   b. Is it possible for two customers to buy the same book but for different prices? If so, how is this possible? If not, how does the logic of the E-R diagram prohibit this?

      Sure. They can each buy the book but through different offers (with different prices).

   c. Does the customer always pay the lowest price for which he or she is eligible (**eligible**)? If not, is there an easy way to modify the E-R diagram in order to assure this?

No. The logic of the diagram does not ensure this. In fact, there is no way that we could ensure this via E-R. The E-R model does not give us any way to refer to actual values in the database, so there is no way to "talk about" lowest price in E-R. We could handle this later in application programs we write on the database to run the bookstore, of course, if this is important.

d. Does the E-R diagram ensure that the Offer under which a customer buys a book is, in fact, legitimate? That is, an offer is for a particular club's members. Are we guaranteed that the customer belongs to the corresponding club? Why or why not?

No, it does not. The rel-ships belongs_to *and* buys *are independent. We can put entries in* buys *without having to check with* belongs_to*. The rel-ship* buys *relates a customer and an offer. There is no club information involved.*
Again, this is not to say we could not handle such a business rule outside of the design. We could check it within the application programs and queries we build on the database for running the bookstore. It just is not enforced by the design. (So if we were to allow customers to fill out their own orders using SQL, they could buy books under offers for which they are not eligible.)
Ideally, if we could redesign the E-R so that it did *protect against this, then the database system would ensure that customers only buy books through offers for which they are eligible club-wise. We would not have to be sure to handle this ourselves in our queries and APPs. Such a design would be then a more elegant solution (as long as the design were not horrendously complex).*
In this case, it is possible to fix this in E-R. A solution is provided in the class notes.

5. **The Fix**

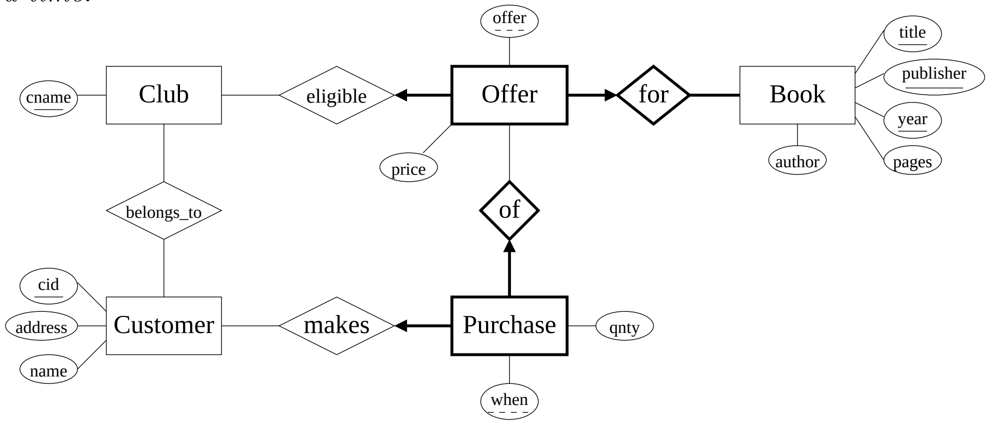
   Consider again **YRB.com**'s database (the E-R in Figure 1).

   a. There is a serious flaw (at least one) in the design in Figure 1, at least in as far as any bookseller would be concerned. What is the flaw?

   Well, there are numerous flaws, especially for a real-world bookstore.
   A big obvious one, though, is that a customer can buy a given book via a given offer only once! This is because we modeled buys *as a rel-ship. We want that a customer could buy the same book more than once. There are many natural situations for this, and from the point of view of the bookstore, is only to be encouraged.*

   b. Redesign the E-R from Figure 1 to fix this.

   To fix this, we could replace the rel-ship buys *with an entity, say named* Purchase. *We could make* Purchase *a weak entity on* Offer, *since a purchase has to be for something. We could then make the attribute* when *a partial key. (Its type in the relational implementation would most likely be* timestamp, *recording the date and time.) We should add an attribute* quantity *to store the number of copies of the book that they are buying. We would want to allow them to buy more than a single copy at a time!*



Person(ssn, *name*, gender, *job*, studio, title, year)
FK (studio) refs **Studio**,
FK (title, year) refs **Movie**
Category(cat)
Movie(title, year, length, *director*, *studio*, *cat*)
FK (director) refs **Person** (ssn),
FK (studio) refs **Studio**,
FK (cat) refs **Category**
Cast(ssn, title, year, character)
FK (ssn) refs **Person**,
FK (title, year) refs **Movie**
Studio(studio, city)

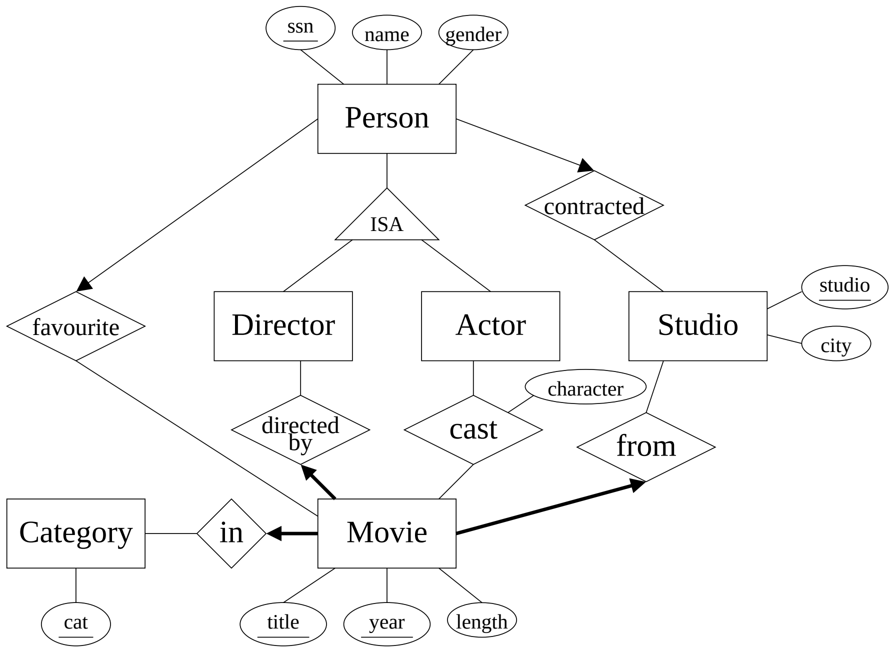
Figure 2: Movies and Casts.

6. **Reverse Engineering**

   A relational schema for movies and casts is given in Figure 2. The primary key for each relation is indicated by the underlined attributes, as ssn in **Person**. Attributes that are starred, as *director* in **Movie**, are not nullable (not null). (Likewise, any attribute which is part of the primary key is not nullable.) The foreign keys for each table are written as FK ..., and refs is shorthand for references.

   The attributes title and year in table **Person** indicate that person's favorite movie. The attribute job of **Person** is allowed two values: 'actor' or 'director'. Only actors are allowed to act in (**Cast**) movies. Only directors are allowed to direct (director). A movie is produced by a studio. A person (actor or director) can be *contracted* by a studio (but by at most one studio), as indicated by studio in **Person**.

   Reverse engineer this into a reasonable entity-relationship diagram that reflects the same logic of this relational schema and the explanation of the domain above.



There is no evidence that table **Cast** *was derived from an entity. Indeed, it is consistent with how we would render a many-many rel-ship. So we make* ***cast*** *a rel-ship in our diagram. We handled a person's job by making an ISA hierarchy which divides people into two sub-entities, directors and actors. It is sensible to do this because directors and actors participate in different rel-ships. There is no attribute* job *in the diagram since the ISA handles the distinction.*

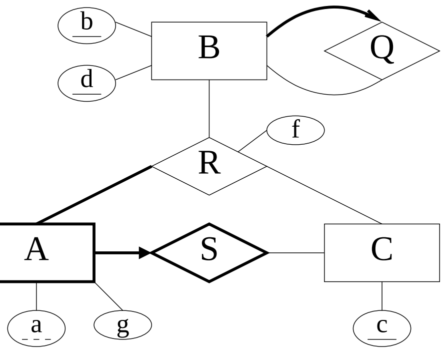


Figure 3: A, B, C's

7. **Scheming**

   Write a relational schema for the E-R in Figure 3. You may use the short-hand style like in Figure 2 from Question 6, rather than in full SQL CREATE TABLE statements. (Do this for all the components of the diagram—**A**, **B**, **C**, etc.—not just for part of it.) Be certain to capture the participation and key constraints from the E-R diagram in your relational schema, where possible. Do not introduce any tables that are not necessary. Note that **R** is a many-many-many.

A($\underline{a}, \underline{c}, g$)
FK (c) refs **C** *% Handles* **S**
B($\underline{b}, \underline{d}, x, y$)
FK (x, y) refs **B** *(b, d) % Handles* **Q**
C($\underline{c}$)
R($\underline{a}, \underline{b}, \underline{c}, \underline{d}, f$)
FK (a, c) refs **A**,
FK (b, d) refs **B**,
FK (c) refs **C**

Each entity is made a table. The many-one rel-ships **Q** *and* **S** *do not have to be made tables. Each can be handled as a FK within the table representing the entity with the arrow. Furthermore, we* have to *do it this way to capture the manditory participation in each case. We have to do it this way for* **S** *because* **A** *is a weak entity on* **C***, and we could not capture this correctly if* **S** *were its own table. For* **Q***, we could not force that every* **B** *tuple participates in the* **Q** *rel-ship, and is thus related to another* **B** *tuple, otherwise.* **Q** *seems odd since it is a recursive rel-ship, relating* **B** *to itself. But we have seen these before. For instance, imagine that* **B** *is* **Employee** *and* **Q** *is the rel-ship* **manages***. Now you might think that we could render* **B** *and* **Q** *as*

B($\underline{b}, \underline{d}$) *FK (b, d) refs* **B**

However, this would be wrong! This would only allow each **B** *tuple to be associated with itself via rel-ship* **Q***! There is no sense in this. So each employee is his or her own boss. (Well, actually, that doesn't sound so bad ....) So we have to introduce extra columns to hold the rel-ship* **Q***. Here, we introduced x and y. For* **employee***, we might introduce an attribute manager to reference, say, the managers' ssn.*
A *is weak on* **C***. So its key is the union of its attributes marked as being part of the key (a), and the attributes that form* **C***'s key (c).*
Rel-ship **R** *we have to make into a table since it is many-many-many. It relates three entities (***A***,* **B***, and* **C***), so three FKs are needed, one to each of the entities's tables. Table* **R***'s key is the union of the three entities's keys.* **R** *has an attribute of its own, f. so we include it as a column in its table.*
Note there is an ambiguity in the E-R diagram regarding **R***. Is* **R** *only allowed to relate an instance of* **A** *with an instance of* **C** *(and further with an instance of* **B***, of course) such that that* **A** *is weak on that* **C***? This is the way we modeled it above. Or should* **R** *be free to associate* any **A** *with* any **C** *with any* **B***? If it is the later, we would have to render table* **R** *as, say*

R($\underline{a}, \underline{b}, \underline{c}, \underline{d}, \underline{e}, f$)
FK (a, e) refs **A** *(a, e),*
FK (b, d) refs **B**,
FK (c) refs **C**

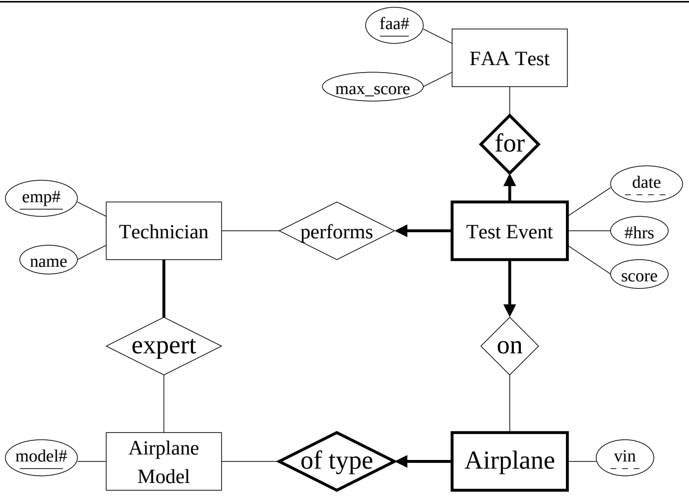
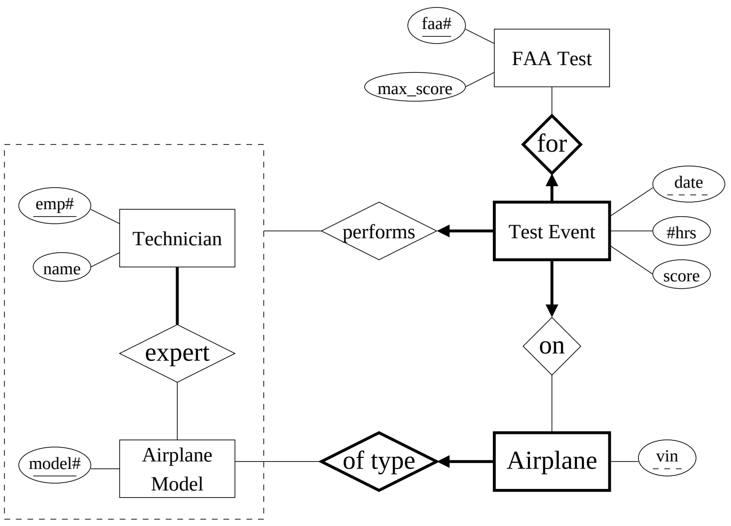


Figure 4: A, B, C's

8. **Extreme E-R**

   a. Redesign the E-R diagram from Figure 8 so that in the revised E-R diagram a technician who conducts a test on an airplane is guaranteed to be an expert on that model of airplane.



This can be done easily by making an aggregation block around the rel-ship **expert**, *and changing the rel-ship* **performs** *to be between the aggregation* **expert** *and* **Test Event**.

Under a rule of design minimality, this works: **performs** *associates a* **Test Event** *(which is associated with a particular* **Airplane** *and hence with a particular* **Airplane Model***) with an "***expert***" (who is asscoiated with a particular* **Airplane Model***). That the airplane model has to be the same in both cases is forced in* **performs**.

   b. Write a relational schema derived from your E-R design in 8a. Again, you may use the short-hand style as in Figure 2 from Question 6,

FAA_Test(*faa#*, *max_score*)
Airplane_Model(*model#*)
Airplane(*vin*, *model#*)
    FK (model#) refs **Airplane_Model** *% Handles 'of type'*
Technician(*emp#*, *name*)
Expert(*emp#*, *model#*)
    FK (emp#) refs **Technician**,
    FK (model#) refs **Airplane_Model**
Test_Event(*faa#*, *date*, *#hrs*, *score*, *vin*, *model#*)
    FK (faa#) refs **FAA_Test**, *% Handles 'for'*
    FK (vin, model#) refs **Airplane**, *% Handles 'on'*
    FK (emp#, model#) refs **Expert** *% Handles 'performs'*

9. **Normal.** *Is this normal?*

   Consider the relation schema ABCDEF with the following set of functional dependencies:

$$\begin{array}{rlrl} \mathsf{A} \mapsto \mathsf{B} & & \mathsf{AE} \mapsto \mathsf{F} \\ \mathsf{CD} \mapsto \mathsf{A} & & \mathsf{CE} \mapsto \mathsf{F} \\ \mathsf{BC} \mapsto \mathsf{D} & & & \end{array}$$

   a. What are the keys of the relationship ABCDEF above with respect to the set of functional dependencies?

   To help reduce the work, we note that neither C nor E appear on the right-hand-side of any of the FDs. So they must belong in any *candidate key.*
   We can check then to see if CE is a candidate key. If it is, we are done. It would be the only candidate key. (Any other set of attributes that acts as "key" must contain C and E, and so would be a super-key.)

$$CE \mapsto F.$$

   No, we don't infer all the attributes when we compute the closure of CE. We only get additionally F.
   So we try A, B, D, and F, in turn, with CE to see if any of those work.

$$\begin{array}{r} ACE \mapsto BDF.\ \textit{Yes!} \\ BCE \mapsto ADF.\ \textit{Yes!} \\ CDE \mapsto ABF.\ \textit{Yes!} \\ CEF \mapsto.\ \textit{No.} \end{array}$$

   Three of them are candidate keys. Next, we need to look at subsets of four attributes (with two of them being C and E). However, all are supersets of ACE, BCE, or CDE, the candidate keys we know so far. So those could only be superkeys, and we need not check them. The same is true for yet larger subsets of the attributes, so our search is complete.

   b. Is schema ABCDEF in BCNF? Why or why not?

   Consider the second to last FD, AE ↦ F. AE is not equivalent to, nor a superset of, any of the three candidate keys. Therefore, this FD violates BCNF, so the relation is not in BCNF.
   In fact, each *FD in this example violates BCNF!*

   Is schema ABCDEF in 3NF? Why or why not?

For 3NF, an FD is a violation iff *the left-hand-side is not equivalent to, nor a superset of, any candidate key,* or *the right-habd-side attribute is* prime. *(An attribute is called* prime *if it appears as part of any of the candidate keys.) For* $\mathsf{A} \mapsto \mathsf{B}$*,* $\mathsf{CD} \mapsto \mathsf{A}$*, and* $\mathsf{BC} \mapsto \mathsf{D}$*, each of* B*,* A*,* D *appear in some candidate key, so none is a problematic FD for 3NF. Attribute* F*, however, appears in none of the candidate keys and so is not prime. Thus this will not excuse the last two FDs,* $\mathsf{AE} \mapsto \mathsf{F}$ *and* $\mathsf{CE} \mapsto \mathsf{F}$*. As we observed when checking for BCNF for* $\mathsf{AE} \mapsto \mathsf{F}$*,* AE *is not equivalent to, nor a superset of, any of the candidate keys. Therefore,* $\mathsf{AE} \mapsto \mathsf{F}$ *violates 3NF. So does* $\mathsf{CE} \mapsto \mathsf{F}$*.*

Is schema ABCDEF in 2NF? Why or why not?

Again, the first three FDs cannot be problems because their right-hand-sides are prime. Is $\mathsf{AE} \mapsto \mathsf{F}$ *a problem for 2NF? 2NF says each FD must either have a prime right-hand-side, or the left-hand-side cannot be a* proper subset of *any candidate key. However,* AE *is a subset of candidate key* ACE*. So this violates 2NF too. Likewise, so does* $\mathsf{CE} \mapsto \mathsf{F}$*.*

10. **Intersection by any other name...**

Give two separate (*independent*) arguments (informal proofs) why the relational operator $\cap$ (*intersection* is algebraically redundant, given the relational operators σ (*selection*), π (*projection*), $\bowtie$ (*join*), $\cup$ (*union*), and $-$ (*set-difference*). That is, show that an R.A. query that uses intersection can always be rewritten as a query that does not.

So we want to find other R.A. ways to write $\mathsf{R} \cap \mathsf{S}$.
Let R*'s schema be* $\mathsf{R}(A, B)$ *and* S*'s schema be* $\mathsf{S}(C, D)$. *Then* $\mathsf{R} \bowtie_{A=C \wedge B=D} \mathsf{S}$ *would mean the same thing. In general, an equi-join between two tables that involves* all *the columns is the same as an intersection between them.*
An aside: Consider what $\mathsf{R} \bowtie \mathsf{S}$ *means. It is a natural join (the join columns are left implicit). So we join on the attribute names in common. There are none! When the join condition is empty, a join is equivalent then to the cross-product. So, in this case,* $\mathsf{R} \bowtie \mathsf{S}$ *is the same as* $\mathsf{R} \times \mathsf{S}$. *So at one extreme, when the join equates all the columns, the join is the same as intersection. At the other extreme, when the join involves no columns, it is equivalent to cross-product.*
Other equivalences we can pull from what we know with Venn-diagrams and set theory. $\mathsf{R} \cap \mathsf{S}$ *can be rewritten as* $((\mathsf{R} \cup \mathsf{S}) - (\mathsf{R} - \mathsf{S})) - (\mathsf{R} - \mathsf{S})$. *This works for R.A. too. Note that the parentheses are important above! Why?*
$\mathsf{R} - (\mathsf{R} - \mathsf{S})$ *is yet another equivalent formula. Of course, so is* $\mathsf{S} - (\mathsf{S} - \mathsf{R})$ *then. Here, it really is clear why the parentheses matter.*
Many would like to say $\overline{\overline{\mathsf{R}} \cup \overline{\mathsf{S}}}$. *That's fine for Venn-diagrams, but not for R.A. We do not have a "complement" operator in R.A.!*

11. **How many tuples for a tuppence?**

Problem #4.2 from the textbook.

expression	*assumption*	*min*	*max*
$\mathbf{R}_1 \cup \mathbf{R}_2$	$\mathbf{R}_1$ *&* $\mathbf{R}_2$ *are union compatable*	N_2	$N_1 + N_2$
$\mathbf{R}_1 \cap \mathbf{R}_2$	$\mathbf{R}_1$ *&* $\mathbf{R}_2$ *are union compatable*	0	N_1
$\mathbf{R}_1 - \mathbf{R}_2$	$\mathbf{R}_1$ *&* $\mathbf{R}_2$ *are union compatable*	0	N_1
$\mathbf{R}_1 \times \mathbf{R}_2$		$N_1 \times N_2$	$N_1 \times N_2$
$\sigma_{a=5}(\mathbf{R}_1)$	$\mathbf{R}_1$ *has an attribute named* a	0	N_1
$\mathbf{R}_1 \div \mathbf{R}_2$	*The set of attributes of* $\mathbf{R}_2$ *is a subset of the attributes of* $\mathbf{R}_1$	0	N_1
$\mathbf{R}_2 \div \mathbf{R}_1$	*The set of attributes of* $\mathbf{R}_1$ *is a subset of the attributes of* $\mathbf{R}_2$	0	$\lfloor N_2/N_1 \rfloor$

12. **What's that in English?!**

Problem #4.4 from the textbook.

For a. and b., the questions are bogus. I do not know what they mean. Apologies. The rest are meaningful, though.
For c., we could get bogus answers. Say Parke, Inc.*, a supplier in Toronto makes only cheap, red parts. Say* Parke, Inc.*, a different supplier in Williamsburg makes only cheap, green parts. They are different companies, but the same name. (They'd have different* sid*'s, of course.) But the query would return* Parke, Inc.*. See why?*
To protect against that problem, query d. does the right thing and intersects using the key, sid*. However, if we wanted the suppliers' names back, this does not do that. Query e. revises d. to also return the names.*

13. **One of these things is not like the other.**

Consider the schema $\mathbf{R}(\underline{\mathsf{A}}, \mathsf{B}, \mathsf{C})$, $\mathbf{S}(\underline{\mathsf{A}}, \underline{\mathsf{D}})$, and $\mathbf{T}(\underline{\mathsf{D}}, \mathsf{B}, \mathsf{E})$. Consider the following relational algebra expressions with respect to the above schema.

a. $\pi_{\mathsf{B}}((\sigma_{\mathsf{C}=5}(\mathbf{R}) \bowtie \sigma_{\mathsf{E}=7}(\mathbf{T})) \bowtie \mathbf{S})$

b. $\pi_{\mathsf{B}}(\pi_{\mathsf{A,B,D}}(\sigma_{(\mathsf{C}=5)\wedge(\mathsf{E}=7)}(\mathbf{R} \bowtie \mathbf{T})) \cap (\mathbf{S} \times \pi_{\mathsf{B}}(\mathbf{T})))$

c. $\pi_{\mathsf{B}}((\sigma_{\mathsf{C}=5}(\mathbf{R} \bowtie \mathbf{S})) \bowtie (\sigma_{\mathsf{E}=7}(\mathbf{S} \bowtie \mathbf{T})))$

d. $\pi_{\mathsf{B}}(\sigma_{\mathsf{C}=5}(\mathbf{R} \bowtie \mathbf{S})) \cap \pi_{\mathsf{B}}(\sigma_{\mathsf{E}=7}(\mathbf{S} \bowtie \mathbf{T}))$

e. $\pi_{\mathsf{B}}(\pi_{\mathsf{A,B}}(\sigma_{\mathsf{C}=5}(\mathbf{R})) \bowtie (\mathbf{S} \bowtie \pi_{\mathsf{D,B}}(\sigma_{\mathsf{E}=7}(\mathbf{T}))))$

One of these is not like the others. That is, one of them may evaluate differently from the other four. Which one?

Give an example relation for which this one does evaluate to a different answer than the other four, and show what the odd one evaluates to, and what the other four R.A. expressions evaluate to.

To see what's going on, we have to simplipy the formulas. Let us look at the selections (σ's) Each time, we select just the tuples of $\mathbf{R}$ *such that* $C = 5$*; for* $\mathbf{S}$ *such that* $E = 7$*. Note that columns* C *and* E *are not used in any other operation! They are not used in any join condition, and we ultimately throw them away. (We only keep* B *in the end, in each case.) So let* $\mathbf{R}' = \pi_{A,B}(\sigma_{C=5}(\mathbf{R}))$ *and let* $\mathbf{T}' = \pi_{D,B}(\sigma_{E=7}(\mathbf{T}))$*. Thus, the above are equivalent to*

a. $\pi_B((\mathbf{R}' \bowtie \mathbf{T}') \bowtie \mathbf{S})$

b. $\pi_B(\pi_{A,B,D}(\mathbf{R}' \bowtie \mathbf{T}') \cap (\mathbf{S} \times \pi_B(\mathbf{T})))$

c. $\pi_B(\mathbf{R}' \bowtie \mathbf{S}) \bowtie (\mathbf{S} \bowtie \mathbf{T}'))$

d. $\pi_B(\mathbf{R}' \bowtie \mathbf{S}) \cap \pi_B(\mathbf{S} \bowtie \mathbf{T}')$

e. $\pi_B(\pi_{A,B}(\mathbf{R}') \bowtie (\mathbf{S} \bowtie \pi_{D,B}(\mathbf{T}')))$

These are easier to look at and to understand. And they each mean the same thing as the corresponding original.
Now, the order that joins are done in does not matter (unlike with set-difference). We know that join is associative and it is reflexive. That is, $((\mathbf{R} \bowtie \mathbf{T}) \bowtie \mathbf{S})$ *and* $(\mathbf{R} \bowtie (\mathbf{T} \bowtie \mathbf{S})$ *mean the same thing! And* $\mathbf{R} \bowtie \mathbf{T}$ *and* $\mathbf{T} \bowtie \mathbf{R}$ *must evaluate to the same thing. Since join is associative, we often just write* $(\mathbf{R} \bowtie \mathbf{T} \bowtie \mathbf{S})$ *for* $((\mathbf{R} \bowtie \mathbf{T}) \bowtie \mathbf{S})$ *since the order in which the joins are evaluated does not matter. The ambiguity left by leaving out the parentheses is immaterial in this case.*

Note the colums of $\mathbf{R}'$ *are* A *and* B*, and of* $\mathbf{T}'$ *are* D *and* B*. So* $\pi_{A,B}\mathbf{R}'$ *is just* $\mathbf{R}'$*, and* $\pi_{D,B}\mathbf{T}'$ *is just* $\mathbf{T}'$*. So, our queries now simplify to*

a. $\pi_B(\mathbf{R}' \bowtie \mathbf{T}' \bowtie \mathbf{S})$

b. $\pi_B((\mathbf{R}' \bowtie \mathbf{T}') \cap (\mathbf{S} \times \pi_B(\mathbf{T})))$

c. $\pi_B(\mathbf{R}' \bowtie \mathbf{S} \bowtie \mathbf{S} \bowtie \mathbf{T}')$

d. $\pi_B(\mathbf{R}' \bowtie \mathbf{S}) \cap \pi_B(\mathbf{S} \bowtie \mathbf{T}')$

e. $\pi_B(\mathbf{R}' \bowtie \mathbf{S} \bowtie \mathbf{T}')$

Well, $\mathbf{S} \bowtie \mathbf{S}$ *is just* $\mathbf{S} \cap \mathbf{S}$ *which is, of course, just* $\mathbf{S}$*. So now, queries a., c., and e. are all clearly identical.*
What is happening in query b.? Can we make that '$\cap$*' between "*$\mathbf{S}$*" and "*$\mathbf{R}' \bowtie \mathbf{T}'$*" a '*$\bowtie$*' instead? Well,* $\mathbf{S}$*'s attributes are* A *and* D*, but the attributes of* $\mathbf{R}' \bowtie \mathbf{T}'$ *are* A*,* B*, and* D*. Clearly though, if, for any tuple from* $\mathbf{R}' \bowtie \mathbf{T}'$*, there is a tuple from* $\mathbf{S}$ *that matches it on* A *and* D*, there must be a tuple from* $\mathbf{S} \times \pi_B(\mathbf{T})$ *that matches it exactly. Why? the matching* $\mathbf{S}$ *tuple has been crossed with* all *the possible* B*'s, so one of them must match with the* B *of the* $\mathbf{R}' \bowtie \mathbf{T}'$ *tuple. So, in this case, we can replace* $(\mathbf{R}' \bowtie \mathbf{T}') \cap (\mathbf{S} \times \pi_B(\mathbf{T}))$ *with* $\mathbf{R}' \bowtie \mathbf{T}' \bowtie \mathbf{S}$*! Thus, query b. is the same as the others too.*
Why is query d. different? The '$\cap$*' would be equivalent to a '*$\bowtie$*' if* all *the attributes were involved. But we project to just* B *on both sides first. So more tuples can survive the final cut than with the other four queries.*
Consider the following example tables:

R

A	B	C
1	2	5
5	6	5

T

D	B	E
3	2	7
8	6	7

S

A	D
1	8
5	3

For queries a., b., c., and e., the answer table is empty. For query d., though, it has rows $\langle 2\rangle$ *and* $\langle 6\rangle$*.*

14. **Relational Algebra Queries.** *Parlez-vous l'algébra rélational?*

Customer	
cust#	PK
cname	
fav_colour	
phone#	

Item	
item#	PK
prod#	FK to **Product**
cust#	FK to **Customer**
colour	
date_sold	

Product	
prod#	PK
pname	
cost	
maker	FK to **Company**

Avail_Colours	
prod#	PK, FK to **Product**
colour	PK

Consider the relational schema above. PK stands for *primary key*, and FK for *foreign key*. Not all the schema is shown, but you do not need the parts not seen. Make natural assumptions about what the table attributes mean.

Write a relational algebra query for each of the following.

a. Show, for each customer (reporting the customer's name), the products by name that come in the customer's favourite colour.

$$\pi_{cname,pname}((\textbf{Customer} \bowtie_{fav\_colour=colour} \textbf{Avail\_Colour}) \bowtie \textbf{Product})$$

b. Show, for each customer (reporting the customer's name), the products by name that *do not* come in the customer's favourite colour.

$$\begin{aligned}&\pi_{cname,pname}(\\ &\qquad \pi_{cust\#,cname,prod\#,pname}(\textbf{Customer} \times \textbf{Product}) -\\ &\qquad \pi_{cust\#,cname,prod\#,pname}((\textbf{Customer} \bowtie_{fav\_colour=colour} \textbf{Avail\_Colour}) \bowtie \textbf{Product})\\ &)\end{aligned}$$

c. List pairs of customers (columns: first_cust#, first_cname, second_cust#, second_cname) such that the two customers own at least two products in common.

Write your query so that a pair is not listed twice. For instance, if $\langle 5, bruce, 7, parke\rangle$ is listed, then $\langle 7, parke, 5, bruce\rangle$ should not be.

$$\begin{aligned}&\rho(\textbf{One}, \sigma_{c1\#<c2\#}(\\ &\qquad \pi_{c1\#,cn1,prod\#}(\rho((cust\# \rightarrow c1\#, cname \rightarrow cn1), \textbf{Customer} \bowtie \textbf{Item})) \bowtie\\ &\qquad \pi_{c2\#,cn2,prod\#}(\rho((cust\# \rightarrow c2\#, cname \rightarrow cn2), \textbf{Customer} \bowtie \textbf{Item}))))\\ &\pi_{c1\#,cn1,c2\#,cn2}(\sigma_{p1\#\neq p2\#}(\rho((prod\# \rightarrow p1\#), \textbf{One}) \bowtie \rho((prod\# \rightarrow p2\#), \textbf{One})))\end{aligned}$$

d. List customers who own items in all the available colours. That is, for every available colour, the customer owns some item in that colour.

$$\begin{array}{l}\pi_{\mathit{cust\#},\mathit{cname}}(\textbf{Customer}) \bowtie \\ (\pi_{\mathit{cust\#}}(\textbf{Customer}) - \\ \qquad \pi_{\mathit{cust\#}}((\pi_{\mathit{cust\#}}(\textbf{Customer}) \times \pi_{\mathit{colour}}(\textbf{Avail\_Colour})) - \pi_{\mathit{cust\#},\mathit{colour}}(\textbf{Item})))\end{array}$$

Or

$$\pi_{\mathit{cust\#},\mathit{cname},\mathit{colour}}(\textbf{Customer} \bowtie \textbf{Item}) \div \pi_{\mathit{colour}}(\textbf{Avail\_Colour})$$

e. List each customer by name, paired with the product(s) by name that he or she has bought that was the most expensive (cost) of all the products he or she has bought.

Note that there actually may be ties. For instance, $\langle bruce, ferrari \rangle$ and $\langle bruce, porsche \rangle$ would both qualify if both were \$80,000, and for everything else he has bought, each item was less expensive than \$80,000.

$$\rho(\textbf{Costs},\ \pi_{\mathit{cust\#},\mathit{cname},\mathit{prod\#},\mathit{pname},\mathit{cost}}((\textbf{Customer} \bowtie \textbf{Item}) \bowtie \textbf{Product}))$$

$$\begin{array}{l}\pi_{\mathit{cust\#},\mathit{cname},\mathit{prod\#},\mathit{pname}}(\\ \qquad \textbf{Costs} - \pi_{\mathit{cust\#},\mathit{cname},\mathit{prod\#},\mathit{pname},\mathit{cost}}(\sigma_{\mathit{cost}<\mathit{cost2}}(\\ \qquad\qquad\qquad \textbf{Costs} \bowtie \pi_{\mathit{cust\#},\mathit{cname},\mathit{cost2}}(\rho((\mathit{cost} \rightarrow \mathit{cost2}), \textbf{Costs})))))\end{array}$$

15. **Domain Relational Calculus.** *Déjà vu to you too.*

Write the same queries as in Question 14, but as domain relational calculus queries instead.

a. Show, for each customer (reporting the customer's name), the products by name that come in the customer's favourite colour.

$$\begin{array}{l}
\{\langle C\#, Cn, P\#, Pn\rangle \mid \\
\quad \exists Col(\\
\quad\quad \exists Ph\#(\langle C\#, Cn, Col, Ph\#\rangle \in \mathit{Customer}) \wedge \\
\quad\quad \langle P\#, Col\rangle \in \mathit{Avail\_Colours} \wedge \\
\quad\quad \exists Ct, M(\langle P\#, Pn, Ct, M\rangle \in \mathit{Product}))\}
\end{array}$$

b. Show, for each customer (reporting the customer's name), the products by name that *do not* come in the customer's favourite colour.

$$\begin{array}{l}
\{\langle C\#, Cn, P\#, Pn\rangle \mid \\
\quad \exists FC(\\
\quad\quad \exists Ph\#(\langle C\#, Cn, FC, Ph\#\rangle \in \mathit{Customer}) \wedge \\
\quad\quad \exists Ct, M(\langle P\#, Pn, Ct, M\rangle \in \mathit{Product}) \wedge \\
\quad\quad \forall Col(\langle P\#, Col\rangle \in \mathit{Avail\_Colours} \rightarrow \\
\quad\quad\quad FC \neq Col))\}
\end{array}$$

c. List pairs of customers (columns: first_cust#, first_cname, second_cust#, second_cname) such that the two customers own at least two products in common.

Write your query so that a pair is not listed twice. For instance, if $\langle 5, bruce, 7, parke\rangle$ is listed, then $\langle 7, parke, 5, bruce\rangle$ should not be.

$$\begin{array}{l}
\{\langle C1\#, Cn1, C2\#, Cn2\rangle \mid \\
\quad \exists FC, Ph\#(\langle C1\#, Cn1, FC, Ph\#\rangle \in \mathit{Customer}) \wedge \\
\quad \exists FC, Ph\#(\langle C2\#, Cn2, FC, Ph\#\rangle \in \mathit{Customer}) \wedge \\
\quad \exists P1\#, P2\#(\\
\quad\quad \exists I\#, Col, DS(\langle I\#, P1\#, C1\#, Col, DS\rangle \in \mathit{Item}) \wedge \\
\quad\quad \exists I\#, Col, DS(\langle I\#, P1\#, C2\#, Col, DS\rangle \in \mathit{Item}) \wedge \\
\quad\quad \exists I\#, Col, DS(\langle I\#, P2\#, C1\#, Col, DS\rangle \in \mathit{Item}) \wedge \\
\quad\quad \exists I\#, Col, DS(\langle I\#, P2\#, C2\#, Col, DS\rangle \in \mathit{Item}) \wedge \\
\quad\quad P1\# \neq P2\#) \wedge \\
\quad C1\# < C2\#\}
\end{array}$$

d. List customers who own items in all the available colours. That is, for every available colour, the customer owns some item in that colour.

$$
\begin{array}{l}
\{\langle C\#, Cn\rangle \mid \\
\quad \exists FC, Ph\#(\langle C\#, Cn, FC, Ph\#\rangle \in \mathit{Customer}) \land \\
\quad \forall Col, P\#(\langle Col, P\#\rangle \in \mathit{Avail\_Colours} \rightarrow \\
\qquad (\exists I\#, P\#, DS(\langle I\#, P\#, C\#, Col, DS\rangle \in \mathit{Item})))\}
\end{array}
$$

e. List each customer by name, paired with the product(s) by name that he or she has bought that was the most expensive (cost) of all the products he or she has bought.

Note that there actually may be ties. For instance, $\langle bruce, ferrari\rangle$ and $\langle bruce, porsche\rangle$ would both qualify if both were \$80,000, and for everything else he has bought, each item was less expensive than \$80,000.

$$
\begin{array}{l}
\{\langle Cn, Pn\rangle \mid \\
\quad \exists C\#, FC, Ph\#(\langle C\#, Cn, FC, Ph\#\rangle \in \mathit{Customer} \land \\
\qquad \exists I\#, P\#, Col, DS(\langle I\#, P\#, C\#, Col, DS\rangle \in \mathit{Item} \land \\
\qquad\quad \exists Cost, M(\langle P\#, Pn, Cost, M\rangle \in \mathit{Product} \land \\
\qquad\qquad \forall I2\#, P2\#, Col2, DS2(\langle I2\#, P2\#, C\#, Col2, DS2\rangle \in \mathit{Item} \rightarrow \\
\qquad\qquad\quad \forall Pn2, Cost2, M2(\langle P2\#, Pn2, Cost2, M2\rangle \in \mathit{Product} \rightarrow \\
\qquad\qquad\qquad \mathit{Cost} \geq \mathit{Cost2})))))\}
\end{array}
$$

or

$$
\begin{array}{l}
\{\langle Cn, Pn\rangle \mid \\
\quad \exists C\#, FC, Ph\#(\langle C\#, Cn, FC, Ph\#\rangle \in \mathit{Customer} \land \\
\qquad \exists I\#, P\#, Col, DS(\langle I\#, P\#, C\#, Col, DS\rangle \in \mathit{Item} \land \\
\qquad\quad \exists Cost, M(\langle P\#, Pn, Cost, M\rangle \in \mathit{Product} \land \\
\qquad\qquad \forall I2\#, P2\#, Col2, DS2(\langle I2\#, P2\#, C\#, Col2, DS2\rangle \in \mathit{Item} \rightarrow \\
\qquad\qquad\quad \exists Pn2, Cost2, M2(\langle P2\#, Pn2, Cost2, M2\rangle \in \mathit{Product} \land \\
\qquad\qquad\qquad \mathit{Cost} \geq \mathit{Cost2})))))\}
\end{array}
$$

16. **Decomposition.** *Totally lossless!*

Consider the relation schema ABCDEF with the following set of functional dependencies:

$$\begin{array}{rr} \mathsf{A} \mapsto \mathsf{B} & \mathsf{AE} \mapsto \mathsf{F} \\ \mathsf{CD} \mapsto \mathsf{A} & \mathsf{CE} \mapsto \mathsf{F} \\ \mathsf{BC} \mapsto \mathsf{D} & \end{array}$$

Provide a lossless-join decomposition that is in BCNF for ABCDEF above with respect to the set of functional dependencies.

We start with ABCDEF. The FD A ↦ B breaks BCNF because {A} is not a key nor superkey. Let us decompose using this to produce tables AB and ACDEF. This is a lossless decompostion by definition, as the intersection of the two tables, A, is key for one of them, AB.
Are we done? No, ACDEF is not in BCNF. CD ↦ A breaks BCNF for it. We recheck and ACE and CDE are the candidate keys for ACDEF. So since CD is not a key nor superkey (with respect to table ACDEF and all the FDs, implied too), then ACDEF is not in BCNF. So we can decompose ACDEF into ACD and CDEF. Again, this is a lossless decomposition. (I'm limiting myself to chosing lossless steps.)
Are we done? No, CDEF is not yet in BCNF. CE ↦ F breaks BCNF for it. We recheck CDEF's candidate keys: only CDE works. So CE is not a key nor superkey (with respect to table CDEF and all the FDs, implied too). We can decompose CDEF then using CE ↦ F: CDE and CEF. (Lossless, by definition, again.)
Are we done? We can find no FDs that pertain to CDE. Of course CDE has candidate key; itself. We can find no FDs that pertain to CEF. It has a candidate key of CE. We find no others. The only FD (implied or otherwise) that applies to CEF is CE ↦ F. So both CDE and CEF are in BCNF.
All together, we have decomposed ABCDEF into AB, ACD, CDE, and CEF. This is a losseless decomposition, as shown by the steps above. All four resulting tables are in BCNF with respect to the FDs, so this is a BCNF decomposition.

17. **Decomposition.** *Lose something?*

Consider again the relation schema ABCDE, and the same set of functional dependencies, from Question 16. Dr. Dogfurry, the world famous database consultant, came up with the following decomposition: AEF, AB, and ACD.

Is this a lossless join decomposition with respect to ABCDEF and the set of functional dependencies?

No, this is not a lossless join decomposition. There are two ways to demonstrate this. One is to devise example tables that obey the FDs and show that (natural) joining the decomposed tables together result in more *tuples than the original table. This is only possible when the decomposition is lossy. The other way is to show that no lossless-join decomposition tree exists that results in these three (and* only *these three) tables.*
Let us pursue the latter approach. The join of AEF and ACD is lossy because A is not a candidate key for either table. The joins of AB and AEF and of AB and ACD are lossless, however, because A is a candidate key for AB.
Consider the first of these lossless joins, resulting in ABEF. Is the join of ABEF and the remaining table ACD lossless? No, because A is not a candidate key for ABEF nor for ACD. Consider the second of these lossless joins, resulting in ABCD. Is the join of ABCD and the remaining table AEF lossless? No again, because A is not a candidate key for ABCD nor for AEF.
We have run out of possibilities for how to join the three tables losslessly to regain the original. So there is no lossless-join decomposition tree that works. Therefore, this decomposition is lossy.

18. **Minimal Cover.** *Got you covered.*

Given the set $\mathcal{F}$ of functional dependencies $\{\mathsf{AB} \mapsto \mathsf{C}, \mathsf{A} \mapsto \mathsf{D}, \mathsf{BD} \mapsto \mathsf{C}\}$, find a minimal cover of $\mathcal{F}$.

To find a minimal cover, we first repeatedly eliminate an FD from the collection as long as the remaining FDs imply the one we are eliminating. Second, we "generalize" the remaining FDs by dropping as many of the attributes on the left-hand side of an FD as possible while implying the same set of FDs.
Let us do that here. Can we drop $\mathsf{BD} \mapsto \mathsf{C}$*? No. The remaining two do not imply it. Can we drop* $\mathsf{A} \mapsto \mathsf{D}$*? No. The remaining two do not imply it. Can we drop* $\mathsf{AB} \mapsto \mathsf{C}$*? Yes! The remaining two* do *imply it. We know* $\mathsf{BD} \mapsto \mathsf{C}$*. Since* $\mathsf{A} \mapsto \mathsf{D}$*, by transitivity* $\mathsf{AB} \mapsto \mathsf{C}$*.*
So we know have $\{\mathsf{A} \mapsto \mathsf{D}, \mathsf{BD} \mapsto \mathsf{C}\}$*. Can we reduce in the second step by eliminating any attributes on the left-hand sides?* BD *is the only possibility. Does* $\mathsf{D} \mapsto \mathsf{C}$ *follow from the remaining? No. Does* $\mathsf{B} \mapsto \mathsf{C}$ *follow from the remaining? No. So we could not "generalize" any.*
The final answer: $\{\mathsf{A} \mapsto \mathsf{D}, \mathsf{BD} \mapsto \mathsf{C}\}$*.*
Half of folks answered this question incorrectly with the answer $\{\mathsf{AB} \mapsto \mathsf{C}, \mathsf{A} \mapsto \mathsf{D}\}$*. It is easy to think that since* A *implies* D*, then we can replace the* A *in* $\mathsf{AB} \mapsto \mathsf{C}$ *with* D*. But that is applying FD transitivity in the wrong direction.*
For example, let A *be a person's* ssn*,* D *be the person's birthday,* B *be hour-of-day, and* C *hourly wage. Then* $\mathsf{AB} \mapsto \mathsf{C}$ *says that* ssn *and* hour-of-day *together determines* hourly-wage*. (So, for example, people may earn more on off-shift hours.) However,* birthday *and* hour-of-day *would not determine* hourly-wage*. Be careful in reasoning with FDs. It is easy to mess up.*