Databases: Exercises

Martin Kleppmann (martin.kleppmann@cst.cam.ac.uk)

Exercise 1. What are some advantages and disadvantages of a declarative query language compared to a general-purpose programming language such as Python?

Exercise 2. Instead of having a separate `ratings` table, another way of making a one-to-one relation would be to add `rating` and `votes` as columns of the `movies` table. What are the pros and cons of these approaches? Think about how the tables might be used by an application, how you might enforce constraints, and how future-proof each approach is.

Exercise 3. Give the SQL statements that you would use to change the example database to add the concept of a series of movies. You can add a column to an existing table using the `ALTER TABLE` statement; see the DuckDB documentation for details. How would you record a movie's position within its series (e.g. being part 1, 2, or 3 of a trilogy)? Can you prevent two movies from being given the same position within their series?

Exercise 4. Every type of relationship that a person may have with a movie – acting, directing, producing, writing, composing – is recorded in one table, `has_position`, with a `position` column saying which type each row is. An alternative design would be to create a separate table for each type of relationship: `directs`, `produces`, `composes`, and so on. How would you decide which of these approaches to use?

Exercise 5. Write each of the following as a relational algebra expression over the example database, and then as a SQL query.

1. The titles of the movies released in 1995.
2. The titles of the movies that are at least three hours long, together with their running times.

Exercise 6. Both of the following are valid SQL. Which one corresponds to $\sigma_{\text{year}=1995}(\text{movies})$?

```sql
SELECT * FROM movies WHERE year = 1995;
```

```sql
SELECT year = 1995 FROM movies;
```

What does the other query return, and why?

Exercise 7. Two different movies may have the same title – for example, a remake. Write a query that returns the titles that are shared by two or more movies in the example database.

Exercise 8. Write a relational algebra expression, and then a SQL query, that returns the title of every movie paired with the name of every genre it belongs to. That query will return several rows for the same movie if it belongs to multiple genres. Next, modify the query so that it returns one row per movie, with the movie's genres gathered into a single comma-separated string. (Hint: use `GROUP BY` together with DuckDB's `string_agg()` aggregate function.)

Exercise 9. Write a SQL query that returns the ID and title of all the movies in the example database that have no actors, using `LEFT JOIN`. (Hint: to test whether the value in some column `x` is null you have to use the operator `x IS NULL`, not `x = NULL` as you might expect. The query should return 2 rows on the example database.)

Exercise 10. The following query returns the IDs of the movies that belong to both the Crime genre and the Drama genre:

```sql
-- Return the IDs of all movies that are both in the
-- Crime (genre_id = 6) and Drama (genre_id = 8) genres
SELECT movie_id FROM has_genre WHERE genre_id = 6
INTERSECT
SELECT movie_id FROM has_genre WHERE genre_id = 8;
```

Write a query that returns the same result using a join instead of `INTERSECT`.

Exercise 11. Write a SQL query that returns the ID and title of all the movies in the example database that have no actors, like in Exercise 9, but this time using `EXCEPT` instead of `LEFT JOIN`. Why would it be a mistake to write this query returning the title alone?

Exercise 12. In boolean logic, the expression `not (a and b)` is equivalent to `(not a) or (not b)`, and the expression `not (a or b)` is equivalent to `(not a) and (not b)`. Is the same true in 3-valued logic?

Is the expression `a or (not a)` always true in 3-valued logic? If not, is there a similar expression that is always true regardless of the value of `a`?

Exercise 13. The following pairs of queries are over two tables, `r` with columns `x` and `y`, and `s` with columns `y` and `z`. For each pair, say whether the two queries always return the same result. If they do not, give an example database on which the results differ.

1. ```sql
   SELECT r.x FROM r WHERE r.y IN (SELECT s.y FROM s);
   ```

   ```sql
   SELECT r.x FROM r JOIN s ON r.y = s.y;
   ```

2. ```sql
   SELECT r.x FROM r JOIN s ON r.y = s.y AND s.z = 99;
   ```

   ```sql
   SELECT r.x FROM r JOIN s ON r.y = s.y WHERE s.z = 99;
   ```

3. ```sql
   SELECT r.x FROM r LEFT JOIN s ON r.y = s.y AND s.z = 99;
   ```

   ```sql
   SELECT r.x FROM r LEFT JOIN s ON r.y = s.y WHERE s.z = 99;
   ```

4. ```sql
   SELECT r.x FROM r LEFT JOIN s ON r.y = s.y WHERE s.y IS NULL;
   ```

   ```sql
   SELECT r.x FROM r WHERE r.y NOT IN (SELECT s.y FROM s);
   ```

Exercise 14. Write a query that returns the names of people who are credited as a composer on at least one movie, like on the slide, but using an inner join instead of the `EXISTS` or `IN` operator. Be careful that it still gives the right answer if two different people in the database happen to have the same name.

Exercise 15. An index, which we discussed in the introduction, is also a form of redundancy: the values of the indexed column are stored both in the table and in the index. In what ways is an index similar to denormalisation, and in what ways is it different? Consider who is responsible for keeping the copies consistent, what happens on reads and on writes, and whether the choice affects how queries are written and what they return.

Exercise 16. Write a SQL query over the relational version of the example database that produces one JSON document per movie, containing the movie's ID, title, and year, its genres as an array of strings, and its actors as an array of objects, each with the actor's `person_id` and `name`. You will need DuckDB's `json_object` function, which takes alternating keys and values and returns a JSON object, and the `list` aggregate function, which collects the values in a group into a list. How does your output differ from the documents in the JSON version of the database?

Exercise 17. For each of the following applications, would you store the data in a relational schema or as documents? Justify your answers, and if you choose documents, say what each document would contain.

1. A blogging platform, where each post has a title, a body text, some tags, and comments from readers.
2. A bank, which stores customers, their accounts, and the transfers between accounts.
3. An event log, which stores every time various functions were invoked, and what arguments were passed to the function invocation.
4. An online shop's product catalogue, in which different types of product have different attributes.
5. A system for storing the settings of a mobile app, such as the user's preferred language, notification preferences, and so on.

Exercise 18. Which of the following properties of the people documents in the example database can be enforced by a JSON Schema, and which cannot? Explain why.

1. Every person has a name.
2. A person's `deathyear`, if present, is not earlier than their `birthyear`.
3. No two documents have the same `person_id`.
4. Every `movie_id` in a person's `acted_in` array is the ID of a movie document.
5. The `title` in each element of `acted_in` is the same as the title in the corresponding movie document.

Exercise 19. Write SQL queries over the JSON version of the example database to answer the following questions.

1. What are the IDs of all documents that needed to be updated when Lana Wachowski changed her name? Assume you know her `person_id` is `nm0905154`. (Hint: the JSONPath expression `$.*[*]` matches every element of every array at the top level of a document.)

2. Who are the five people who have directed the most movies? (Hint: there is a function `json_array_length`.)
3. In which movies did one of the directors also act? (Hint: DuckDB's `list_has_any(l1, l2)` function returns true if the two lists have at least one element in common.)
4. What is the average rating of the movies in each genre, for the five genres with the highest average rating?

Exercise 20. Write Cypher queries over the graph version of the example database to answer the following questions.

1. Which movies did Charles Chaplin both direct and act in?
2. Who are the five people who composed the music for the most movies?
3. On which movies did both Lana Wachowski and Lilly Wachowski work, in any position?

Exercise 21. Write a SQL query over the relational version of the example database that returns the same result as the Cypher query that finds the pairs of actors who have appeared together most often. Compare the two queries.

Exercise 22. In the graph version of the example database, each movie's genres are a list of strings in its `genres` property. An alternative design would be to make each genre a vertex with the label `Genre`, and to connect each movie to its genres with `HAS_GENRE` edges. For each design, write a Cypher query that counts the movies that are both romantic and comedies (that is, have both the Romance and the Comedy genres). What are the advantages and disadvantages of each design?

Exercise 23. In the recursive SQL query that finds the movies filmed in Cambridgeshire, what would happen if `UNION` were replaced by `UNION ALL`? What would happen to each version of the query if, due to an error in the data, `located_in` contained a cycle (for example, if Cambridgeshire were recorded as being located within Cambridge)? How does the Datalog version behave in the same situations?

Exercise 24. Write a Datalog query that finds the names of all places that lie within more than one continent. You can recognise a continent as a place that is not located in any other place. Then write the same query in Cypher, assuming that the graph has a vertex with the label `Place` for each place, and an edge with the label `LOCATED_IN` for each `located_in` fact.

Exercise 25. Write a Datalog query that counts, for each country, the number of movies that were filmed somewhere in that country. You can recognise a country as a place that is located directly in a continent (see Exercise 24). Then write the same query in SQL, assuming that the relational database has tables `places`, `located_in`, and `filmed_at` with the same columns as the Datalog relations. Did you need to take care of anything in SQL that you did not need to worry about in Datalog?

Exercise 26. Write a SQL query using `WITH RECURSIVE` that computes the same result as the Datalog query for Bacon numbers. Does your query also need a bound on the distance? Why, or why not?