A Strategy

- start with FROM – identify a table that contains all the rows and columns needed for the end result (extra rows and columns are fine, but missing ones are not)
  - if the desired columns are not all in the same table, use JOIN or NATURAL JOIN (or, more rarely, CROSS PRODUCT) to combine tables
    - prefer JOIN/NATURAL JOIN when combining rows based on matching columns
    - prefer CROSS PRODUCT (+ WHERE) for all pairs or combinations not based on matching columns
  - if the desired columns don't exist directly and must be computed, use a subquery to do the computation and then JOIN, NATURAL JOIN, or CROSS PRODUCT as appropriate/needed to combine with other tables
  - if the desired rows aren't all in one table, use UNION (either with subqueries in FROM, or with the top-level queries)

A Strategy

- if the FROM table has extra rows, use WHERE to pick out just the desired rows
  - if determining whether the current row should be part of the result requires information that is not in that row (e.g. it involves other rows or it involves columns in other tables), use a subquery to get that information
    - this is most likely a correlated subquery – it references something from the current row in the outer query
  - cannot use aggregation functions directly in WHERE – use a subquery or GROUP BY/HAVING instead (depending on the goal)

Strategies

- use GROUP BY if you need an aggregation function (COUNT, SUM, AVG, MIN, MAX, etc) – identify which column's value to use to group rows together
  - but only if it is possible to have more than one group (if not, the aggregation function just applies to all the rows that pass the WHERE) and if groups can have more than one row (if not, nothing is achieved by grouping)
  - for COUNT, consider whether to use COUNT(*), COUNT(*x*), COUNT(DISTINCT *x*)
- use HAVING if you only want some of the groups
  - can use aggregation functions in HAVING – applied to the current group

Strategies

Finishing steps –

- finish the SELECT clause
- ORDER BY
- LIMIT

Throughout –

- when you've written a query or subquery, put it back into language in the vein of the original task to check the logic
- use stepping stones (incremental development)
  - for complicated tasks, start with queries that solve part of the task and work towards the full thing