Suppose we have the following database, storing students, courses, and courses taken by students. There are two foreign keys in courses_taken: sid, referencing students(sid), and cid, referencing courses(cid).

students

| sid | name |
|---|---|
| 1 | alice |
| 2 | bob |
| 3 | charlie |

courses

| cid | cname |
|---|---|
| 1 | databases |
| 2 | calculus |

courses_taken

| sid | cid |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |
| 3 | 2 |

**1. Print the sids of students who took every course**

```
SELECT S.sid
FROM students S
WHERE NOT EXISTS (SELECT *
                  FROM courses C
                  WHERE NOT EXISTS (SELECT *
                                    FROM courses_taken T
                                    WHERE C.cid=T.cid AND S.sid=T.sid))
```

Now recall the supply chain database from the midterm:
Suppliers(sid, sname, address) Parts(pid, pname, colour) Catalog(sid, pid, price)

**2. Print the sids of suppliers who supply at least one red part and at least one green part**

Two correct answers:

```
SELECT c.sid
FROM Catalog c NATURAL JOIN Parts p
WHERE p.colour='red' AND c.sid IN (
   SELECT c2.sid
   FROM Catalog c2 NATURAL JOIN Parts p2
   WHERE p2.colour='green')
```

```
SELECT c.sid
FROM Catalog c NATURAL JOIN Parts p
WHERE p.colour='red' AND EXISTS (
   SELECT *
   FROM Catalog c2 NATURAL JOIN Parts p2
   WHERE c2.sid=c.sid and p2.colour='green')
```

Subqueries in EXIST and NOT EXIST must be "correlated" with the outer queries. In the above example, the EXISTS query is correlated to the outer query via c2.sid=c.sid. Let's try to remove the correlation. We get the following query:

```
SELECT c.sid
FROM Catalog c NATURAL JOIN Parts p
WHERE p.colour='red' AND EXISTS (
    SELECT *
    FROM Catalog c2 NATURAL JOIN Parts p2
    WHERE p2.colour='green')
```

This query is meaningless. It returns an empty answer if no supplier supplies a green part; otherwise, it prints all sids who supply a red part.

**3. For each part, print its cheapest price and the supplier that supplies it at the cheapest price.**

```
SELECT c.sid, c.pid, c.price
FROM Catalog c
WHERE c.price <= ALL
  (SELECT c2.price
  FROM Catalog c2
  WHERE c2.pid=c.pid)
```

Note: If we remove the correlating condition, what does the following query compute? (Answer: the row(s) from the catalog table with the lowest price)

```
SELECT c.sid, c.pid, c.price
FROM catalog c
WHERE c.price <= ALL
  (SELECT c2.price
  FROM catalog c2)
```

**4. Print all part IDs supplied by at least one supplier, and such that all suppliers that supply them charge <$10 for them.**

```
SELECT c.pid
FROM catalog c
WHERE 10 > ALL
  (SELECT c2.price
  FROM catalog c2
  WHERE c.pid=c2.pid)
```

**5. What is the sid of the cheapest supplier of part 3?**

```
SELECT c.sid
FROM Catalog c
WHERE c.pid=3 AND c.price =
        (SELECT MIN(c2.price)
        FROM Catalog c2
        WHERE c2.pid=3)
```

**6. Print the minimum average price across all parts**

```
SELECT MIN(temp.average_cost) as minimum_average_price
FROM (
        SELECT c.pid as pid, AVG(c.price) as average_cost
        FROM Catalog c
        GROUP BY c.pid
) AS temp
```

here is another way to write this query:

```
WITH temp AS
        (SELECT c.pid as pid, AVG(c.price) as average_cost
        FROM Catalog c
        GROUP BY c.pid )
SELECT MIN(temp.average_cost) as minimum_average_price
FROM temp
```

Summary of nested query templates:

```
SELECT FROM WHERE something IN (SELECT FROM WHERE)
SELECT FROM WHERE something NOT IN (SELECT FROM WHERE)
SELECT FROM WHERE EXISTS (SELECT FROM WHERE)
SELECT FROM WHERE NOT EXISTS (SELECT FROM WHERE)
SELECT FROM WHERE something/constant ARITHIMETIC_OPERATOR (SELECT FROM WHERE)
SELECT FROM WHERE something/constant ARITHIMETIC_OPERATOR ALL (SELECT FROM WHERE)
SELECT FROM (SELECT FROM WHERE)
WITH something AS (SELECT FROM WHERE) SELECT FROM something WHERE
```

See MySQL manual for more information: https://dev.mysql.com/doc/refman/8.0/en/