Mini SQL Engine

Goal

Simulate a tiny in-memory database over a **predefined table** with a **fixed subset of SQL**. All SQL statements strictly follow the templates given below.

Table (Predefined)

```
TABLE people (
  id   INT,
  name TEXT,
  age  INT,
  city TEXT
)
```

Columns are exactly: `id`, `name`, `age`, `city`.

Statements

INSERT Insert exactly one row:

```
INSERT INTO people VALUES (INT, 'TEXT', INT, 'TEXT');
```

UPDATE Set a single column to a literal; optional single-condition WHERE:

```
UPDATE people SET col = LIT [WHERE col = LIT];
```

DELETE Optional single-condition WHERE:

```
DELETE FROM people [WHERE col = LIT];
```

SELECT Select rows from the table in descending order; optional single-condition WHERE. Exactly one column is selected and exactly one column is used in `ORDER BY`:

```
SELECT col FROM people [WHERE col = LIT] ORDER BY col DESC;
```

Literals and Tokens

- `col`: one of `id`, `name`, `age`, `city`
- `INT`: an integer
- `TEXT`, `LIT`: a string of `A-Z`, `a-z`, `0-9`

Execution Rules

- Start with an empty database.
- Execute statements in order.
- Print the result of **every** `SELECT`.

Input Format

- Line 1: integer `Q`, the number of statements.
- Next `Q` lines: SQL statements, each ending with a semicolon `;` and strictly following the templates.

Output Format

For each `SELECT`:

1. Print a line with the number of result rows `R` (after filtering).
2. Then print `R` lines: the single selected column per row. Print integers as decimal, and text without quotes.

Sample

Input

```
10
INSERT INTO people VALUES (1, 'ALICE', 20, 'NY');
INSERT INTO people VALUES (2, 'BOB', 21, 'SF');
INSERT INTO people VALUES (3, 'CAROL', 20, 'NY');
UPDATE people SET name = 'HUANCHEN' WHERE id = 2;
SELECT name FROM people ORDER BY id DESC;
SELECT name FROM people WHERE age = 20 ORDER BY name DESC;
DELETE FROM people WHERE city = 'NY';
SELECT id FROM people ORDER BY age DESC;
INSERT INTO people VALUES (4, 'DAVE', 22, 'LA');
SELECT city FROM people WHERE id = 4 ORDER BY city DESC;
```

Output

```
3
CAROL
HUANCHEN
ALICE
2
CAROL
ALICE
1
2
1
LA
```

Step-by-Step State (after each mutating statement)

We show the entire `people` table state after each statement that changes data (`INSERT`, `UPDATE`, `DELETE`). Columns are `id`, `name`, `age`, `city`.

1. Insert `(1, 'ALICE', 20, 'NY')`

id	name	age	city
1	ALICE	20	NY

2. Insert `(2, 'BOB', 21, 'SF')`

id	name	age	city
1	ALICE	20	NY
2	BOB	21	SF

3. Insert `(3, 'CAROL', 20, 'NY')`

id	name	age	city
1	ALICE	20	NY
2	BOB	21	SF
3	CAROL	20	NY

4. Update `name = 'HUANCHEN'` where `id = 2`

id	name	age	city
1	ALICE	20	NY
2	**HUANCHEN**	21	SF
3	CAROL	20	NY

5. `SELECT name FROM people ORDER BY id DESC;` No state change. Sort rows by `id` descending (3, 2, 1), then output the `name` column: `CAROL`, `HUANCHEN`, `ALICE`.

6. `SELECT name FROM people WHERE age = 20 ORDER BY name DESC;` No state change. Filter `age=20` gives rows for `ALICE` and `CAROL`. Order by `name` descending: `CAROL`, `ALICE`.

7. Delete where `city = 'NY'`

id	name	age	city
2	HUANCHEN	21	SF

8. `SELECT id FROM people ORDER BY age DESC;` No state change. Only one row remains; output `id = 2`.

9. Insert `(4, 'DAVE', 22, 'LA')`

id	name	age	city
2	HUANCHEN	21	SF
4	DAVE	22	LA

10. `SELECT city FROM people WHERE id = 4 ORDER BY city DESC;` No state change. Filter `id=4`, then output `city = LA`.