Relačný model a SQL

Ján Mazák

FMFI UK Bratislava

Relačný model

Databáza pozostáva z tabuliek (relácií).

Stĺpce — atribúty; každý má doménu (množinu povolených hodnôt, čiže dátový typ prípadne zúžený dodatočnými obmedzeniami). Určené pomenovaním alebo pozíciou.

Riadky — záznamy (records, rows, tuples, n-tice).

Relačný model

```
CREATE TABLE employees (
    id INTEGER PRIMARY KEY,
    lastName TEXT NOT NULL,
    firstName VARCHAR(255),
    age INTEGER CHECK (age >= 18)
);
```

Relačný model

Reláciu možno vnímať ako predikát alebo ako (multi)množinu záznamov.

V bežných DBMS sa relácia chápe ako multimnožina:

- na poradí riadkov nezáleží;
- riadky v tabuľke sa môžu opakovať.

Odstránenie duplikátnych záznamov z výsledku dotazu:

```
SELECT DISTINCT x, y FROM ...
```

NULL

- Špeciálna hodnota NULL zodpovedá neznámej hodnote.
- Trojhodnotová logika, napr. `NULL OR FALSE` je NULL.
- Test, či je hodnota NULL: `x IS NULL` / `x IS NOT NULL`
- Pri vytváraní tabuľky možno NULL zakázať. Inak treba starostlivo zvažovať, ako ovplyvní operátory a agregačné funkcie (napr. priemer hodnôt v stĺpci). Nehádajte, použite dokumentáciu.

Dotazy v SQL

Základná štruktúra:

```
SELECT attribute1 AS a1, attribute2 AS a2
FROM table AS t
WHERE t.attribute2 > 10
ORDER BY a1, a2
```

(Všimnite si, že aliasy a1, a2 nemožno použiť nikde vnútri dotazu. Kľúčové slovo AS je takmer všade nepovinné.)

Dotazy v SQL

- Pri kľúčových slovách jazyka SQL sa nerozlišujú malé a veľké písmená, ale pre dáta uložené v db áno (ak to nezmeníme napr. použitím ILIKE v podmienke za WHERE).
- Case-sensitivity názvov tabuliek a atribútov závisí od DBMS a operačného systému.
- Úvodzovky pre stĺpce: `"atribút s medzerou v názve"`
- Apostrofy pre konštantné reťazce: `'reťazec'`

Bežné konvencie:

- názvy tabuliek aj atribútov lower case
- kľúčové slová SQL upper case

Dotazy v SQL

Vo výsledku nemusia byť len pôvodné hodnoty atribútov, ale aj čosi z nich vyrátané (napr. aritmetické výrazy zložené z konštánt, funkcií implementovaných v db a hodnôt atribútov daného riadka).

```
SELECT
    concat(e.firstname,' ',e.lastname) AS ename,
    0.8 * e.salary AS salaryAfterTax
    (CASE
        WHEN e.bonus IS NULL THEN e.salary
        ELSE e.bonus + e.salary
    ) AS total_salary,
FROM employee AS e
WHERE dept_id >= 20 AND lower(e.firstname) = 'john'
ORDER BY 0.8 * e.salary
```

Zoradenie záznamov

Zoznam zamestnancov usporiadaný od najvyššieho platu po najnižší, pri rovnakom plate abecedne:

```
SELECT name, salary, department
FROM employee
ORDER BY salary DESC, name ASC
```

Join

Join — spojenie záznamov z dvoch tabuliek.

Je to podmnožina karteziánskeho súčinu tabuliek (každý riadok s každým, dvojicu záznamov spojíme do jedného dlhšieho) špecifikovaná dodatočnými podmienkami na prepájanie.

Join — karteziánsky súčin

Name	Deptno
John	10
Thomas	20
Joe	40

X

Deptno	Dept. name
10	Accounting
20	PR
30	Development

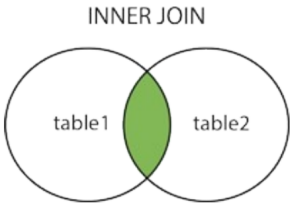
=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
John	10	20	PR
John	10	30	Development
Thomas	20	10	Accounting
Thomas	20	20	PR
Thomas	20	30	Development
Joe	40	10	Accounting
Joe	40	20	PR
Joe	40	30	Development

SELECT * FROM emp, dept

Join — INNER JOIN

INNER JOIN = JOIN:



Name	Deptno
John	10
Thomas	20
Joe	40

JOIN

Deptno	Dept. name
10	Accounting
20	PR
30	Development

=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
~~John~~	~~10~~	~~20~~	~~PR~~
~~John~~	~~10~~	~~30~~	~~Development~~
~~Thomas~~	~~20~~	~~10~~	~~Accounting~~
Thomas	20	20	PR
~~Thomas~~	~~20~~	~~30~~	~~Development~~
~~Joe~~	~~40~~	~~10~~	~~Accounting~~
~~Joe~~	~~40~~	~~20~~	~~PR~~
~~Joe~~	~~40~~	~~30~~	~~Development~~

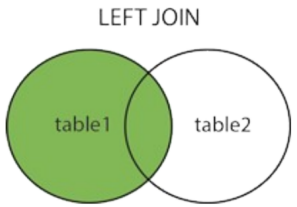
```
SELECT * FROM emp e, dept d
WHERE e.deptno = d.deptno
```

```
SELECT * FROM emp e
        JOIN dept d ON e.deptno = d.deptno
```

```
SELECT * FROM emp e NATURAL JOIN dept d
```

Join — LEFT JOIN

LEFT [OUTER] JOIN:



Name	Deptno
John	10
Thomas	20
Joe	40

LEFT JOIN

Deptno	Dept. name
10	Accounting
20	PR
30	Development
10	Human res.

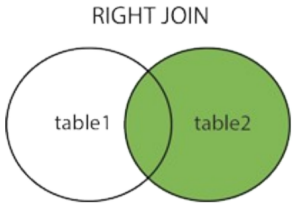
=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
John	10	10	Human res.
Thomas	20	20	PR
Joe	40	null	null

```
SELECT *
FROM emp as e
        LEFT JOIN dept as d
                ON e.deptno = d.deptno
```

Join — RIGHT JOIN

RIGHT [OUTER] JOIN:



Deptno	Dept. name
10	Accounting
20	PR
30	Development
10	Human res.

RIGHT JOIN

Name	Deptno
John	10
Thomas	20
Joe	40

=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
John	10	10	Human res.
Thomas	20	20	PR
Joe	40	null	null

To isté ako LEFT JOIN, akurát v obrátenom poradí

```
SELECT *
FROM dept AS d
        RIGHT JOIN emp AS e ON e.deptno = d.deptno
```

Join — FULL OUTER JOIN

Name	Deptno
John	10
Joe	40

X

Deptno	Dept. name
10	Accounting
30	Development

=

Name	Deptno	Deptno	Dept. name
John	10	10	Accounting
Joe	40	null	null
null	null	30	Development

```
SELECT *
FROM emp e FULL JOIN dept d ON e.deptno = d.deptno
```

Negácia

Negáciu možno vyjadriť pomocou NOT EXISTS:

```
/* zamestnanci, ktorí nemajú podriadených */
SELECT emp.name FROM employee emp
WHERE NOT EXISTS (
    SELECT 1 FROM employee emp2
    WHERE emp2.superior_id = emp.employee_id
)
```

(Nezáleží na tom, ktoré stĺpce sú vymenované za SELECT vo vnútornom dotaze, pretože EXISTS len testuje, či sa tam nachádza aspoň jeden riadok.)

Negácia

Samotný operátor EXISTS bez NOT je ekvivalentný joinu.

```
/* zamestnanci, ktorí majú podriadených */
SELECT DISTINCT emp.name
FROM employee emp
WHERE EXISTS (
    SELECT 1 FROM employee emp2
    WHERE emp2.superior_id = emp.employee_id
)

SELECT DISTINCT emp.name
FROM employee emp, employee emp2
WHERE emp2.superior_id = emp.employee_id
```

Všeobecný kvantifikátor

Jazyk SQL nemá prostriedky na priame vyjadrenie všeobecného kvantifikátora. Postupovať možno v duchu

$$\forall x\ P(x) \qquad \Leftrightarrow \qquad \neg\ \exists x\ \neg\ P(x).$$

Zamestnanci, ktorí majú najvyšší plat spomedzi všetkých $\Leftrightarrow$ zamestnanci, ku ktorým neexistuje zamestnanec s vyšším platom.

```
SELECT e.name
FROM employee e
WHERE NOT EXISTS (SELECT 1
                  FROM employee e2
                  WHERE e2.salary > e.salary)
```

Vnorené dotazy (subqueries)

Okrem operátora EXISTS sa možno pomocou operátora IN (resp. NOT IN) pýtať na prítomnosť v množine.

```
SELECT name
FROM employee e
WHERE e.dept_id IN (SELECT dept_id
                    FROM department d
                    WHERE d.location = 'New York')
```

IN sa optimalizuje horšie ako join a antijoin (NOT EXISTS), dá sa však v prípade potreby prepísať inak. IN má tiež nevýhodu, ak sa môže vyskytnúť NULL.

Vnorené dotazy (subqueries)

Ak je výsledkom vnoreného dotazu tabuľka 1x1, možno ju použiť ako skalár.

```
SELECT name
FROM employee e
WHERE e.dept_id = (SELECT dept_id
                   FROM department d
                   WHERE d.location = 'New York')
```

Ak by vo vnorenom dotaze vyšlo viac riadkov (alebo žiaden), dôjde k chybe.
(Takýmto chybám treba čo najviac predchádzať, napr. použitím UNIQUE už pri vytváraní tabuľky.)

Množinové operácie

Bez zachovania násobnosti riadkov (matematické množiny):

- UNION — zjednotenie
- INTERSECT — prienik
- EXCEPT — rozdiel množín

So zachovaním násobnosti riadkov (multimnožiny):

- UNION ALL — zjednotenie
- INTERSECT ALL — prienik
- EXCEPT ALL — rozdiel množín

Množinové operácie

```
SELECT nazov
FROM kry

UNION

SELECT nazov
FROM stromy
```

```
SELECT nazov
FROM dreviny

EXCEPT

SELECT nazov
FROM stromy
```

Mačky nežerúce myši — bežná chyba

Mačky možno rozdeliť do 4 disjunktných skupín:

A. nežerie nič
B. žerie niečo, ale nie myši
C. žerie myši, ale nič iné
D. žerie myši aj niečo iné

mačky nežerúce myši $= A \cup B$
mačky žerúce iné ako myši $= B \cup D$

Ak je dotaz o „mačkách nežerúcich myši“ a máme za WHERE „potrava <> myši“, asi je to nesprávne. Pre určenie toho, či mačka žerie myši, je úplne irelevantná existencia záznamov o tom, že žerie niečo iné ako myši.
(Dve množiny popísané vyššie však budú totožné, ak každá mačka žerie práve jednu potravu.)

CREATE TEMPORARY TABLE

Používateľ si vie sám vytvoriť dočasnú tabuľku, ktorá zanikne po odpojení od db alebo na konci transakcie. (Taká tabuľka nemusí fyzicky existovať, DBMS ju môže pri každom použití nanovo vypočítať.)

```
CREATE TEMPORARY TABLE employee_details AS (
    SELECT e.emp_id, e.name, e.dept_id,
           d.name, d.location
    FROM employee e, department d
    WHERE e.dept_id = d.dept_id
);
SELECT * FROM employee_details WHERE ...;
```

Hodí sa to napr. ak chceme opakovane využívať ten istý komplikovaný join.

VIEW

Prístup k dátam možno uľahčiť vytvorením pohľadu (VIEW).

```
CREATE VIEW employee_details AS (
    SELECT e.emp_id, e.name, e.dept_id,
           d.name, d.location
    FROM employee e, department d
    WHERE e.dept_id = d.dept_id
);

SELECT * FROM employee_details WHERE ...;
```

VIEW je permanentný objekt v db. Jeho riadky však nemusia byť materializované. Možno cezeň aj vkladať dáta, ale nerobte to (veľa komplikácií s NULL).

Funkcie pre jednotlivé dátové typy

Pri praktickej práci v SQL využívame rôzne funkcie pre špecifické dátové typy.

- dátum a čas
- reťazce
- numerické výpočty
- pretypovanie
- ...
- často špecifické pre konkrétny DBMS
- nebudeme s nimi strácať čas, ľahko sa rieši za pochodu (dokumentácia / LLM)

How to write SQL

Zapisujte dotazy čitateľne a zrozumiteľne.
Vyhýbajte sa zápisom vedúcim k neefektívnemu výpočtu.

- `https://github.com/jarulraj/sqlcheck/blob/master/README.md#query-anti-patterns`
- `https://www.stratascratch.com/blog/best-practices-to-write-sql-queries-how-to-structure-your-code/`
- `https://www.sqlstyle.guide/`

Prečítajte si tieto konvencie *teraz* a nezaťažujte budúcich kolegov nevhodnými zápismi. (Berte ich však ako odporúčania; rôzne zdroje v niektorých veciach nesúhlasia. Zvyčajne sa pri existujúcom kóde riadime miestnymi konvenciami.)

Literatúra

- https://cs186berkeley.net/notes/note1/
- https://cs186berkeley.net/notes/note2/
- https://www.postgresqltutorial.com/ (Sections 1, 2, 3)
- https://www.w3schools.com/sql/sql_intro.asp
- https://sqlbolt.com/lesson/select_queries_introduction
- https://www.learnsqlonline.org/
- https://www.postgresqltutorial.com/postgresql-views/managing-postgresql-views/
- https://drive.google.com/file/d/1HCq2KMZO5UvtXGe1nTqNmkwhc3X-NLI3/view

Na zamyslenie

- Akú časovú zložitosť má výpočet DISTINCT (v závislosti od počtu riadkov za predpokladu, že sa všetky riadky zmestia do RAM)?
- Join tabuliek sa dá počítať len z dvoch tabuliek naraz. Ak ich máme viac, join rátame postupne (predstavte si operátorový strom pre binárnu operáciu, v listoch sú pôvodné tabuľky a v koreni výsledok joinu). Koľko je možných postupov na výpočet joinu 4 tabuliek? Odhadnite počet postupov pre n tabuliek.
- Vymyslite postup výpočtu INTERSECT. Akú má časovú zložitosť v závislosti od počtu riadkov relácií na vstupe? (Chcete lepšiu ako kvadratickú zložitosť.)

Úlohy: SQL

Databáza: *osoba*(meno), *pozna*(kto, koho)

- osoby, ktoré poznajú sysľa
- osoby, ktoré poznajú aspoň dve entity (nemusia to byť osoby)
- osoby, ktoré nepoznajú nič a nikoho
- osoby, ktoré nepoznajú žiadne iné osoby
- osoby, ktoré poznajú iba Jožka
- osoby, ktoré pozná presne jedna osoba
- osoby, ktoré poznajú všetkých známych svojich známych