Databázové systémy

Agregace, sumarizace, vnořené dotazy

Vilém Vychodil

KMI/DATA1, Přednáška 7

Databázové systémy

Přednáška 7: Přehled

1. Agregace:
   - obecné agregační funkce,
   - agregace hodnot relací v Tutorial D.
2. Sumarizace:
   - sumarizace v Tutorial D a SQL,
   - sumarizace jako odvozená operace,
   - agregace vs. sumarizace.
3. Vnořené dotazy a kvantifikace:
   - syntax vnořených dotazů,
   - typy vnořených dotazů: skalární, řádkové, tabulkové,
   - korelované vnořené dotazy a otázky efektivity.
4. Okrajová témata:
   - nerelační modifikace výpisu dotazu.

Agregační operace

motivace:

Ze všech hodnot konkrétního atributu nějaké relace chceme vypočíst jedinou (agregovanou) hodnotu, typicky skalárního typu.

agregační funkce, angl.: *aggregation function*

Za agragační funkci považujeme každou funkci, která na základě dané relace (a jejího atributu) vrací hodnotu, která závisí pouze na hodnotách daného atributu v relaci.

Tutorial D:

- primitivní agregační operátory: `AND`, `AVG` (průměrná hodnota), `COUNT` (vyžaduje právě jeden argument – relaci), `D_UNION`, `EQUIV` (sudý počet hodnot je `FALSE`), `EXACTLY` (vyžaduje dodatečný argument n typu `INT`, uvedený jako první; výsledek agregace je `TRUE` pokud má atribut právě n hodnot `TRUE`), `INTERSECT`, `MAX`, `MIN`, `OR`, `SUM`, `UNION`, `XOR` (lichý počet hodnot je `TRUE`), `XUNION`

Příklad (Tutorial D: Vstupní data pro příklady)

```
VAR scores BASE
  RELATION {name CHAR, course CHAR, date CHAR, score INT}
  KEY {name, course, date};
```

předpokládaná hodnota proměnné:

NAME	COURSE	DATE	SCORE
Abbe	KMI/DATA1	13/01/05	56
Abbe	KMI/DATA1	13/01/08	83
Abbe	KMI/PAPR1	12/04/13	65
Blangis	KMI/PAPR1	12/04/14	34
Blangis	KMI/DATA2	13/01/08	13
Curval	KMI/PAPR1	12/04/14	75
Durcet	KMI/PAPR1	12/04/14	75
Durcet	KMI/DATA1	13/02/23	38
Durcet	KMI/DATA1	13/02/26	89

Příklad (Tutorial D: Agregační operace produkující číselnou hodnotu)

```
/* mean and extremal values */
AVG (scores, score) ⟹ 58.666
MIN (scores, score) ⟹ 13
MAX (scores, score) ⟹ 89

/* computing sums */
SUM (scores, score)          ⟹ 528
SUM (scores {score}, score) ⟹ 453

/* counts of tuples in relations */
COUNT (scores)          ⟹ 9
COUNT (scores {score}) ⟹ 8

/* sum with explicit type conversion */
CAST_AS_RATIONAL (SUM (scores, score)) ⟹ 528.0
```

Příklad (Tutorial D: Agregační operace pro pravdivostní hodnoty)

```
/* auxiliary virtual variables */
VAR high VIRTUAL (EXTEND scores: {foo := score > 50});
VAR low VIRTUAL (EXTEND scores: {foo := score <= 50});

/* aggregation by logical equivalence */
EQUIV (high, foo) ⟹ FALSE

/* aggregation by logical non-equivalence */
XOR (high, foo) ⟹ FALSE

/* aggregation by logical conjunction and disjuntion */
AND (high, foo) ⟹ FALSE
OR (high, foo)  ⟹ TRUE

/* exact number of values is true */
EXACTLY (6, high, foo) ⟹ TRUE
EXACTLY (3, low, foo)  ⟹ TRUE
```

Příklad (Tutorial D: Agregační operace produkující relační hodnotu)

```
/* auxiliary virtual variable */
VAR course_results
  VIRTUAL (scores GROUP {name, date, score} AS result);
/* aggregations of relational values */
UNION (course_results, result)     ⟹ ···
D_UNION (course_results, result)   ⟹ ···
XUNION (course_results, result)    ⟹ ···
INTERSECT (course_results, result) ⟹ ···

/* auxiliary virtual variable */
VAR date_exams
  VIRTUAL (scores {ALL BUT score}
             GROUP {name, course} AS registered);
D_UNION (date_exams, registered)   /* error: not disjoint */
```

Příklad (Tutorial D: Aplikace agregačních operací, vyjádření UNGROUP)

```
/* new virtual variable */
VAR goo VIRTUAL (scores GROUP {date, score} AS result)
  KEY {name, course};

/* ungrouping of the result attribute */
goo UNGROUP result

/* can equivalently be expressed by */
((EXTEND UNION (goo, result): {
    res := RELATION {TUPLE {score score, date date}}
  }) TIMES goo WHERE res <= result) {ALL BUT res, result}

/* alternatively, using WRAP, UNWRAP, and IN */
((UNION (goo, result) WRAP ({score, date} AS res)) TIMES goo
 WHERE res IN result) {ALL BUT result} UNWRAP (res)
```

Sumarizace

agregace × sumarizace:

- **agregace** – jediná hodnota stanovená ze všech hodnot atributu dané relace
- **sumarizace** – výsledkem je relace obsahující hodnoty vypočtené z (částí) hodnot (některých) atributů v dané relaci

Tutorial D:

- obecné výrazy `SUMMARIZE` (viz dále)

SQL:

```
SELECT DISTINCT ⟨výraz_1⟩ AS ⟨nový-atribut_1⟩, ...
  FROM ⟨jméno⟩
  WHERE ⟨podmínka⟩
  GROUP BY ⟨atribut_1⟩, ..., ⟨atribut_n⟩
  HAVING ⟨agregační-podmínka⟩
```

Sumarizace v Tutorial D: Syntax

možné tvary použití:

SUMMARIZE ⟨*relační-výraz*⟩ : {⟨*nový-atribut*$_1$⟩ := ⟨*výraz*$_1$⟩, . . .}

SUMMARIZE ⟨*relační-výraz*⟩ BY {⟨*atribut*$_1$⟩, . . .} : {⟨*nový-atr*$_1$⟩ := ⟨*výraz*$_1$⟩, . . .}

SUMMARIZE ⟨*relační-výraz*⟩ PER (⟨*univerzum*⟩) : {⟨*nový-atr*$_1$⟩ := ⟨*výraz*$_1$⟩, . . .}

role argumentů:

- hodnotou ⟨*relační-výraz*⟩ je relace, kterou chceme sumarizovat
- ⟨*nový-atribut*$_i$⟩ jsou atributy nevyskytující se ve schématu relačního výrazu
- ⟨*výraz*$_i$⟩ jsou výrazy obsahující *sumarizační funkce*: AND, AVG, COUNT, D_UNION, EQUIV, EXACTLY, INTERSECT, MAX, MIN, OR, SUM, UNION, XOR, XUNION
- schéma relačního výrazu ⟨*univerzum*⟩ je podmnožinou schémtu ⟨*relační-výraz*⟩
- v druhém případě je možné použít i notaci BY {ALL BUT ⟨*atribut*$_1$⟩, . . .}}

Sumarizace v Tutorial D: Sémantika

interpretace sumarizace ve tvaru:

SUMMARIZE $\langle relační\text{-}výraz \rangle$ PER $(\langle univerzum \rangle)$: {$\langle nový\text{-}atr_1 \rangle$:= $\langle výraz_1 \rangle$, ...}

je vyjádřena v následujících krocích:

1. nechť $\langle relační\text{-}výraz \rangle$ má hodnotu $\mathcal{D}$ (na schématu R)
   a $\langle univerzum \rangle$ má hodnotu $\mathcal{D}_U$ (na schématu $S \subseteq R$);
2. pro každou $s \in \mathcal{D}_U$ vypočti $\{r \in \mathcal{D} \mid s \subseteq r\}$;
3. vyhodnoť všechny $\langle výraz_i \rangle$ za předpokladů, že proměnné odpovídající atributům
   - $y \in S$ jsou navázány na hodnoty $r(y)$;
   - $y \in R \setminus S$ jsou navázány na relace $\{s\} \circ \mathcal{D}$
     a jejich hondoty musí být *sumarizovány* pomocí *sumarizačních funkcí*;

   n-tici výsledků označ t;
4. vlož $s \cup t$ do výsledku

Sumarizace v Tutorial D: Poznámky

ekvivalentní vyjádření v SUMMARIZE:

BY {⟨$atribut_1$⟩,...} ≡ PER (⟨*relační-výraz*⟩ {⟨$atribut_1$⟩,...})

vynechání PER i BY je ekvivalentní uvedení PER (TABLE_DEE)

Příklad (Tutorial D: Mezní případy sumarizace)

```
/* summarization over DUM and DEE */
SUMMARIZE scores PER (TABLE_DUM): {x := COUNT()}
  ⟹ RELATION {x INTEGER} {}
(SUMMARIZE scores PER (TABLE_DEE): {x := COUNT()})
  ⟹ RELATION {TUPLE {x 9}}
/* summarization over the same relation */
SUMMARIZE scores PER (scores): {x := COUNT()}
  ≡ EXTEND scores: {x := 1}
```

Příklad (Tutorial D: Sumarizace)

```
/* aggregation vs. summarization */
AVG (scores, score) ⟹ 58.666

SUMMARIZE scores: {average := AVG (score)}
  ⟹ RELATION {TUPLE {average 58.666}}

SUMMARIZE scores BY {name}: {m := MAX (score), cnt := COUNT ()}

/* allows to have zero counts */
SUMMARIZE scores PER (people {name}): {cnt := COUNT ()}

/* summarization of relational attributes */
SUMMARIZE (scores GROUP {name, date, score} AS result) {result}: {
  x := UNION (result),
  y := INTERSECT (result)
}
```

Agregační vs. sumarizační funkce

Poznámka o významu sumarizačních funkcí

Sumarizační funkce nejsou *funkce* (procedury) v pravém slova smyslu: jejich použití je možné pouze uvnitř těla výrazu SUMMARIZE a narozdíl od agregačních funkcí se jim nepředávají jako argumenty relace, ve kterých se hodnoty agregují (sumarizují).

Příklad (Tutorial D: Sumarizace vs. nekorektní použití agregace)

```
/* correct use of summarization function AVG */
SUMMARIZE scores: {average := AVG (score)}
  ⟹ RELATION {TUPLE {average 58.666}}

/* error: makes no sense outside of SUMMARIZE */
AVG (score)
```

Příklad (Tutorial D: Sumarizace jako odvozená operace)

```
/* summarization */
SUMMARIZE foo PER (baz {qux}): {
  total := COUNT (),
  conj := AND (b)
}

/* can be expressed as extension */
EXTEND baz {qux}: {
  total := COUNT (foo COMPOSE RELATION {TUPLE {qux qux}}),
  conj := AND (foo COMPOSE RELATION {TUPLE {qux qux}}, b)
}
```

foo:

QUX	B
10	FALSE
10	TRUE
20	FALSE

baz:

QUX	NAME
10	Abbe
20	Blangis
30	Curval

result:

QUX	TOTAL	CONJ
10	2	FALSE
20	1	FALSE
30	0	TRUE

Příklad (SQL: Sumarizace)

```
SELECT count (*) FROM scores;
SELECT count (name) FROM scores;

SELECT avg (score), min (score), max (score) FROM scores;
SELECT avg (score) AS average FROM scores;
SELECT name, avg (score) AS average FROM scores GROUP BY name;

SELECT name, avg (score) AS average FROM scores
  WHERE date LIKE '13%'
  GROUP BY name;

SELECT name, avg (score) AS average FROM scores
  WHERE date LIKE '13%'
  GROUP BY name
  HAVING min (score) < 50;
```

Vnořené dotazy v SQL

Definice (Vnořené dotazy v SQL, angl.: *subqueries*)

Uzávorkovaný výraz ve tvaru příkazu `SELECT`, případně uzávorkovaný výraz typu `VALUES` se nazývá **vnořený dotaz** (je to výraz, na konci není znak „;“).

Příklad (SQL: Vnoření dotazy)

```
(SELECT * FROM foo)
(SELECT x, y FROM foo)
(SELECT x, y FROM foo, bar)
(SELECT x, y FROM foo, bar WHERE foo.x = bar.y)

(VALUES (10))
(VALUES ('Abbe', 10))
(VALUES ('Abbe', 10), ('Blangis', 20))
```

Typy vnořených dotazů v SQL

Vnořené dotazy v SQL mají *jinou sémantiku* podle jejich *místa výskytu* v dotazu. (**!!**)

Vnořené dotazy v SQL dělíme na:

- **skalární** – v dotazu se vyskytují v místech, kde je očekávaná *skalární hodnota:* v tom případě se vnořený dotaz musí vyhodnotit na relaci obsahující jedinou n-tici a jediný atribut a tato relace je následně *přetypována na skalární hodnotu;*
- **řádkové** – v dotazu se vyskytují v místech, kde je očekávaná hodnota typu *řádek:* vnořený výraz se musí vyhodnotit na relaci obsahující jedinou n-tici a tato relace je následně *přetypována na hodnotu typu „řádek“;*
- **tabulkové** – všechny ostatní.

poznámka:

- *koerce* (implicitní přetypování) – dominantní (a dost nebezpečný) rys SQL

Příklad (SQL: Skalární vnořené dotazy)

```
/* students with scores better than the average */
SELECT name, course, date FROM scores WHERE
  score > (SELECT avg (score) FROM scores);

/* students and their best scores (notice the alias for scores) */
SELECT DISTINCT name,
  (SELECT max (score) FROM scores WHERE name = x.name) AS best
  FROM scores AS x;

/* scores of Abbe */
SELECT * FROM scores WHERE name = (VALUES ('Abbe'));

/* extension by a new constrant attribute */
SELECT *, (VALUES (10)) AS ten FROM scores;

SELECT *, (SELECT (VALUES (10))) AS ten FROM scores;
```

Příklad (SQL: Řádkové vnořené dotazy)

```
/* results of Abbe in KMI/DATA1 */
SELECT * FROM scores
  WHERE (name, course) = (VALUES ('Abbe', 'KMI/DATA1'));

/* worst score of Abbe */
SELECT * FROM scores
  WHERE (name, score) =
    (SELECT name, min (score) FROM scores
       WHERE name = 'Abbe' GROUP BY name);

/* the same using explicit ROW */
SELECT * FROM scores
  WHERE ROW (name, score) =
    (SELECT name, min (score) FROM scores
       WHERE name = 'Abbe' GROUP BY name);
```

Příklad (SQL: Tabulkové vnořené dotazy)

```
/* scores in courses where the average score is less than 65 */
SELECT * FROM scores WHERE course IN
  (SELECT course FROM scores
     GROUP BY course HAVING avg (score) < 65);

/* courses with difference of best/worst scores less than 20 */
SELECT * FROM
  (SELECT course, max (score) AS best, min (score) AS worst
     FROM scores GROUP BY course) AS foo
  WHERE best - worst < 20;

/* table subqueries using VALUES */
SELECT * FROM scores WHERE (name, course) IN
  (VALUES ('Abbe', 'KMI/DATA1'), ('Curval', 'KMI/DATA1'));

SELECT * FROM (VALUES (10, 20), (30, 40)) AS foo (x, y);
```

Korelované vnořené dotazy v SQL

Definice (korelovaný vnořený dotaz, angl.: *correlated query*)

Vnořený dotaz se nazývá **korelovaný**, pokud výraz, ze kterého se vnořený dotaz skládá, obsahuje nějaký odkaz na atributy tabulky ve vnější části celého dotazu.

typická forma dotazu:

SELECT ⋯ FROM ⟨*tabulka*⟩ AS ⟨*jméno*⟩ WHERE
(SELECT ⋯ FROM ⋯ WHERE ⋯ ⟨*jméno*⟩.⟨*atribut*⟩ ⋯);

podmínka obsahující ⟨*atribut*⟩ z tabulky ⟨*jméno*⟩

poznámky:

- klasický *indikátor neefektivity* – snažíme se je eliminovat
- korelovaný vnořený dotaz se opakovaně vyhodnocuje

Příklad (SQL: Korelované vnořené dotazy)

```
/* non-correlated table subquery */
SELECT * FROM scores WHERE course IN
  (SELECT course FROM scores
     GROUP BY course HAVING avg (score) < 65);

/* scalar correlated variant of the previous query */
SELECT * FROM scores AS foo WHERE
  (SELECT avg (score) FROM scores
     WHERE course = foo.course) < 65;

/* non-correlated variant using natural join */
SELECT foo.* FROM scores AS foo
  NATURAL JOIN
     (SELECT course, avg (score) AS average
        FROM scores GROUP BY course) AS bar
  WHERE bar.average < 65;
```

Příklad (SQL: Srovnání efektivity)

```
/* non-correlated table subquery */
EXPLAIN SELECT * FROM scores WHERE course IN
  (SELECT course FROM scores
     GROUP BY course HAVING avg (score) < 65);
```

použitý plán:

```
Hash Semi Join  (cost=25.15..48.25 rows=255 width=128)
  Hash Cond: ((scores.course)::text = (scores_1.course)::text)
  ->  Seq Scan on scores  (cost=0.00..15.10 rows=510 width=128)
  ->  Hash  (cost=22.65..22.65 rows=200 width=32)
        ->  HashAggregate  (cost=17.65..20.65 rows=200 width=64)
              Filter: (avg(scores_1.score) < 65::numeric)
              ->  Seq Scan on scores scores_1
                    (cost=0.00..15.10 rows=510 width=64)
```

Příklad (SQL: Srovnání efektivity)

```
/* scalar correlated variant of the previous query */
EXPLAIN SELECT * FROM scores AS foo WHERE
  (SELECT avg (score) FROM scores
     WHERE course = foo.course) < 65;
```

použitý plán:

```
Seq Scan on scores foo  (cost=0.00..8377.83 rows=170 width=128)
  Filter: ((SubPlan 1) < 65::numeric)
  SubPlan 1
    ->  Aggregate  (cost=16.39..16.40 rows=1 width=32)
          ->  Seq Scan on scores  (cost=0.00..16.38 rows=3 width=32)
                Filter: ((course)::text = (foo.course)::text)
```

Postranní vnořené dotazy v SQL

motto:

"Following on from the previous point, it's not clear why 'lateral' subqueries are required in any case." — C. J. Date

Definice (postranní vnořený dotaz, angl.: *lateral query*)

Vnořený dotaz vyskytující se v části `FROM` vnějšího dotazu se nazývá **postranní**, pokud je uvozen klíčovým slovem `LATERAL` a je povoleno, aby obsahoval odkazy na sloupce definované tabulkami nebo vnořenými dotazy, které se *vyskytují na levo od něj* v části `FROM`.

sémantika:

- pro každou n-tici, na jejíž atributy se odkazuje daný postranní dotaz, je daný postranní dotaz vyhodnocen a výsledné n-tice se spojí

Příklad (SQL: Postranní vnořené dotazy)

```
/* scores with additional column of course average scores */
SELECT * FROM
  scores AS foo,
  LATERAL (SELECT avg (score) AS avg_score
             FROM scores
             WHERE course = foo.course) AS bar;

/* equivalently (and more efficiently) without LATERAL */
SELECT * FROM scores
  NATURAL JOIN
(SELECT course, avg (score) AS avg_score
  FROM scores GROUP BY course) AS bar;
```

poznámka:

- v prvním případě by vynechání `LATERAL` způsobilo chybu

Souhrn vybraných vnořených dotazů v SQL

vnořený dotaz ($\langle dotaz \rangle$ musí vracet relaci s jedním atributem):

```
SELECT * FROM ⟨jméno⟩ WHERE ⟨atribut⟩ IN (⟨dotaz⟩)
```

vnořený dotaz ($\langle dotaz \rangle$ musí vracet singleton):

```
SELECT * FROM ⟨jméno⟩ WHERE ⟨atribut⟩ = (⟨dotaz⟩)
SELECT * FROM ⟨jméno⟩ WHERE ⟨atribut⟩ <= (⟨dotaz⟩)
```

existenční a univerzální kvantifikace podmínky:

```
SELECT * FROM ⟨jméno⟩ WHERE ⟨atribut⟩ <= SOME (⟨dotaz⟩)
SELECT * FROM ⟨jméno⟩ WHERE ⟨atribut⟩ <= ALL (⟨dotaz⟩)
```

vnořený dotaz na úrovni atributu:

```
SELECT *, (⟨dotaz⟩) AS ⟨nový-atribut⟩ FROM ⟨jméno⟩
```

vnořený dotaz na úrovni jména tabulky:

```
SELECT * FROM (⟨dotaz⟩) AS ⟨jméno-vnořeného-dotazu⟩
```

Příklad (Tutorial D: Analogie „vnořených dotazů“ ze SQL)

```
/* analogy of generally quantified SQL statement */
foo WHERE IS_EMPTY (bar RENAME {x AS nx} WHERE NOT (x > nx))

/* analogy of existentially quantified SQL statement */
foo WHERE NOT IS_EMPTY (bar RENAME {x AS nx} WHERE (x > nx))

/* analogy of IN-clause in SQL statement */
foo WHERE NOT IS_EMPTY (bar RENAME {x AS nx} WHERE (x = nx))

/* analogy of the nested restrict-before-join query in SQL */
(bar WHERE x > 30) JOIN baz

/* analogy of nested query in place of an attribute */
EXTEND foo: {
  /* add new attribute of relational type */
  y := bar RENAME {x AS nx} WHERE (x = nx)
}
```

Nerelační modifikace výpisu dotazu: Třídění

nerelační operace třídění:

- výsledek reprezentuje pořád stejnou relaci
- stojí mimo RM (přidáním do modelu bychom neudrželi 1NF)
- „pořadí“ n-tic je pouze věcí výpisu, tj. externí prezentace relace

Tutorial D:

$\langle$*relační-výraz*$\rangle$ `ORDER (DESC` $\langle atribut_1 \rangle$`, ASC` $\langle atribut_2 \rangle$`, ...)`

SQL:

`SELECT * FROM` $\langle$*jméno*$\rangle$ `ORDER BY` $\langle výraz_1 \rangle$ `DESC,` $\langle výraz_1 \rangle$ `ASC, ...`

poznámky:

- implicitní je `ASC` (vzestupně), volitelné `DESC` (sestupně)

Nerelační modifikace výpisu dotazu: Ořezání výsledků

nerelační operace třídění:

- výsledek reprezentuje pořád stejnou relaci
- stojí mimo RM (přidáním do modelu bychom neudrželi 1NF)
- „pořadí“ n-tic je pouze věcí výpisu, tj. externí prezentace relace

další klauzule v SQL: `LIMIT`, `OFFSET` (viz cvičení)

Přednáška 7: Závěr

pojmy k zapamatování:

- agregační funkce, agregace hodnot v relacích
- sumarizační funkce, sumarizace
- vnořené dotazy, typy vnořených dotazů, korelované dotazy
- nerelační modifikace dotazu

použité zdroje:


Date C. J.: *Database in Depth: Relational Theory for Practitioners*
O'Reilly Media 2005, ISBN 978–0596100124


Date C. J., Darwen H.: *Databases, Types and the Relational Model*
Addison Wesley 2006, ISBN 978–0321399427


Date C. J.: *SQL and Relational Theory: How to Write Accurate SQL Code*
O'Reilly Media 2011, ISBN 978–1449316402