Data Modelling and Databases - Week 2 (Lectures)

- Author: Ruben Schenk
- Date: 27.04.2021
- Contact: ruben.schenk@inf.ethz.ch

Query Language 2: Relational Algebra

We now show a query language that queries data in a `declarative way` - it tells the system *what* we want, instad of *how* to get it. Example:

$$\{(pid,\ cid) \mid \exists n, p(Product(pid, n, p)) \wedge \exists cn, c(Customer(cid, cn, n)) \\ \wedge \exists s(Purchase(pid, cid, s))\}$$

It is easy to see that this query gets the same result as the following relational algebra query:

$$\Pi_{pid,cid}((Customer \bowtie Purchase) \bowtie Product)$$

Formal Definition for Relational Calculus

We introduce the following formal definitions:

- Database Schema: $S = (R_1, \dots, R_m)$ where each R_i is a Relation
- Relation Schema: $R(A_1 : D_1, \dots, A_n : D_n)$
- Domain: $dom = \cup_i D_i$

We can then define the syntax as follows:

- Let ϕ be a first-order logic formula with free variables $x_1, \dots, x_k$, then $Q_\phi = \{(x_1, \dots, x_k) \mid \phi\}$ is a `domain relational calculus` query.

And we define the semantic as follows:

- Each *relation* R corresponds to a predicate R in ϕ
- Each *instance* I corresponds to a first-order interpretation I
- An *assignment* is a mapping $\alpha : var \to dom$

Therefore the `answer of` Q `over` I is:

$$Q(I) = \{(\alpha(x_1), \dots, \alpha(x_k)) \mid I, \alpha \models \phi\}$$

Safe and Unsafe Queries

Let Q_ϕ be a relational calculus query. Then we say Q_ϕ is `safe`, if $Q_\phi(I)$ is finite for all instances I.

SQL (Structured Query Language)

`SQL` is a familiy of standards:

- Data definition language (DDL)
- Data manipualtion language (DML)
- Query language

SQL: Data Definition Language

DDL provides statements to define the schema. In SQL, you need to provide a name, a set of columns, and their types. Example:

```
CREATE TABLE Professor(
    PersNR integer,
    Name varchar(30),
    Level character(2) default "AP",
    PRIMARY KEY (PersNR)
);
```

We `delete` a relation with the `DROP` keyword:

```
    DROP TABLE Professor;
```

We `modify` a table with the `ALTER` keyword:

```
    -- add a column
    ALTER TABLE Professor ADD COLUMN (age integer);

    -- delete a column
    ALTER TABLE Professor DROP COLUMN age;
```

SQL: Data Manipulation Language

Example:

```sql
-- insert values
INSERT INTO Student (PersNr, Name)
VALUES (28121, 'Frey');

-- delete values
DELETE Student
WHERE Semester < 13;

-- update values
UPDATE Student
SET Semester = Semester + 1;
```

However it is to note, that populating a real DB cannot be done manually tuple by tuple (too cumbersome, error prone, etc.).

SQL: Query Language

Nearly all queries follow the form `SELECT ... FROM ... WHERE ...`. We can put this into relational algebra the following way:

SQL

```sql
SELECT PersNr, Name
FROM Professor
WHERE Level = 'FP';
```

Relational Algebra

$$\Pi_{PersNr,\, Name}(\sigma_{Level=''FP''} Professor)$$

Another example:

SQL

```sql
SELECT Name
FROM Professor P, Lecture L
WHERE P.PersNr = L.ProfNr
AND L.Title = 'Database';
```

Relational Algebra

$$\Pi_{Name}(\sigma_{PersNr=ProfNr \land Title="Database"}(Professor \times Lecture))$$

It is important to note, that *every RA expression can be written in SQL subset:*

- Union $\cup$: $R_1 \cup R_2$ = `(SQL1) UNION (SQL2)`
- Difference $-$: $R_1 - R_2$ = `(SQL1) EXCEPT (SQL2)`
- Selection σ : $\sigma_c(R)$ = `SELECT * FROM (SQL1) WHERE c;`
- Projection Π : $\Pi_{A_1,\dots,A_n} R$ = `SELECT A1,..., An FROM (SQL1)`
- Cross Product $\times$: $R_1 \times R_2$ = `SELECT * FROM (SQL1), (SQL2);`
- Rename ρ : $\rho_{a,b,c} R$ = `SELECT A as a,..., C as c FROM (SQL1);`

Sorting

```sql
SELECT PersNr, Name, Level
FROM Professor
ORDER BY Level DESC, Name DESC;
```

Grouping

```sql
SELECT Level, COUNT(*)
FROM Professor
GROUP BY Level;
```

Known Unknowns and Incomplete Information

NULL and Its Semantics

One way to model incomplete information is to place the values that we don't know with a special state `NULL`. It is important to note, that NULL represents a *state*, not a value.

Operations Over NULL

Arithmetic:

- (NULL + 1) -> NULL
- (NULL * 0) -> NULL

Comparisons:

- (NULL = NULL) -> Unknown
- (NULL < 13) -> Unknown
- (NULL > NULL) -> Unknown
- NULL IS NULL -> True