Data Modelling and Databases

Fabian Bösiger

Contents

Relational Model 1
- Database Schema 1
- Relation Schema 1
- Fields/Attributes 2
- Relational Algebra 2
- Relational Calculus 3

SQL 3
- Create Table 3
- Integrity Constraints 4
- Recursion 4
- Functional Dependencies 4
- Normal Form 4

Entity Relationship Model 4
- Entity Relationship to Relational Model 4

Database System Overview 4

Transactions and ACID 4
- Isolation and Locking 4
- Recovery 5

Relational Model

Database Schema

A set of relation schemas.

Relation Schema

A name and a set of fields/attributes.

For a relation $R(f_1 : D_1, \ldots, f_n : D_n)$, an Instance I_R is a set of tuples $I_R \subseteq D_1 \times \cdots \times D_n$.

Fields/Attributes

A name and a domain (e.g. Integer, String).

Relational Algebra

Relational algebra is an imperative way of obtaining the data. It is step-by-step instructions on how to obtain the result.

Basic Operators

- Union: $\cup$
- Difference: $-$
- Selection: σ
- Projection: Π
- Cartesion Product: $\times$
- Renaming: ρ

Joins

Natural Join $R_1(A, B) \bowtie R_2(B, C) = \Pi_{A,B,C}(\sigma_{R_1.B=R_2.B}(R_1 \times R_2))$

If R_1 and R_2 have no shared attributes, $R_1 \bowtie R_2 = R_1 \times R_2$.

If R_1 and R_2 share all attributes, $R_1 \bowtie R_2 = R_1 \cap R_2$.

Theta Join $R_1 \bowtie_\theta R_2 = \sigma_\theta(R_1 \times R_2)$

Equi-Join $R_1 \bowtie_{A=B} R_2 = \sigma_{A=B}(R_1 \times R_2)$

Semi-Join $R_1(A_1, \ldots, A_n) \ltimes_C R_2(B_1, \ldots, B_m) = \Pi_{A_1,\ldots,A_n}(R_1 \bowtie R_2)$

Relational Division $R \div S = T$, where T is the largest relation such that $S \times T \subseteq R$.

$R \div S = \Pi_{R-S}R - \Pi_{R-S}((\Pi_{R-S}R) \times S - R)$

Relational Calculus

Relational Calculus queries data in a declarative way. It tells the system what we want instead of how to get it.

Example: Get all students who take the database course.

$\Pi_{\text{PersNr}}\sigma_{\text{Title="Database"}}((\text{Student} \bowtie \text{Attends}) \bowtie \text{Lecture})$

Is equivalent to:

$\{p : \exists n, s.\text{Student}(p, n, s) \wedge \exists l, t, c, r(\text{Attends}(p, l) \wedge \text{Lecture}(l, t, c, r) \wedge t = \text{"Database"})\}$

Safety We say a relational calculus query Q is safe, if $Q(I)$ is finite for all instances I. An unsafe example would be $\{x \mid \neg R(x)\}$. The safety problem is undecidable.

SQL

Create Table

```
CREATE TABLE professor (
    pers_nr INTEGER,
    name    VARCHAR(32),
    level   CHARACTER(2) DEFAULT "AP",
    PRIMARY KEY (pers_nr)
)
```

TODO

Integrity Constraints

Recursion

Functional Dependencies

Normal Form

Entity Relationship Model

Entity Relationship to Relational Model

Database System Overview

Transactions and ACID

Isolation and Locking

Every transaction starts with `BEGIN` and ends with either `COMMIT` or `ABORT`. If `COMMIT`, all changes are saved, if `ABORT`, all changes are undone.

Serializability Classes

Serializable Schedules that lead to the same answer as some serial schedules.

```
                                10, 10
Read(A, t)
t := t + 100
Write(A, t)                     110, 10
                Read(A, s)
                s := s * 2
                Write(A, s)     220, 10
Read(B, t)
t := t + 100
Write(B, t)                     220, 110
                Read(B, s)
                s := s * 2
                Write(B, s)     220, 220
```

Conflict Serializable We can translate a schedule into a serial shedule with a sequence of nonconflicting swaps of adjacent actions.

Types of Conflicts:

- Read-Write Conflict
- Write-Read Conflict
- Write-Write Conflict

Locking

Two-Phase Locking Growing Phase:

- Each transaction requests the locks that it needs
- Locks cannot be released in this phase

Shrinking Phase:

- The transaction is only allowed to release locks that it previously required
- The transaction cannot acquire new locks

Cascading Abort

Serial Transactions are serial.

```
                                                10, 10
Read(A, t)
t := t + 100
Write(A, t)                                     110, 10
Read(B, t)
t := t + 100
Write(B, t)                                     110, 110
                        Read(A, s)
                        s := s * 2
                        Write(A, s)             220, 110
                        Read(B, s)
                        s := s * 2
                        Write(B, s)             220, 220
```

Recovery

Recoverability Classes

Recoverable If T_i reads from T_j and commits, then $c_j < c_i$.

Avoids Cascading Aborts If T_i reads X from T_j, then $c_j < r_i[X]$.

Strict If T_i reads from or overwrites X written by T_j, then $(c_j < r_i[X] \wedge c_j < w_i[X]) \vee (a_j < r_i[X] \wedge a_j < w_i[X])$.