Institut für Informationssysteme
Technische Universität Braunschweig

Relational Database Systems 1

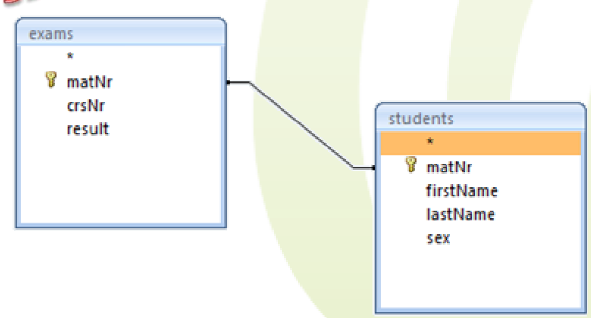
Wolf-Tilo Balke
Simon Barthel

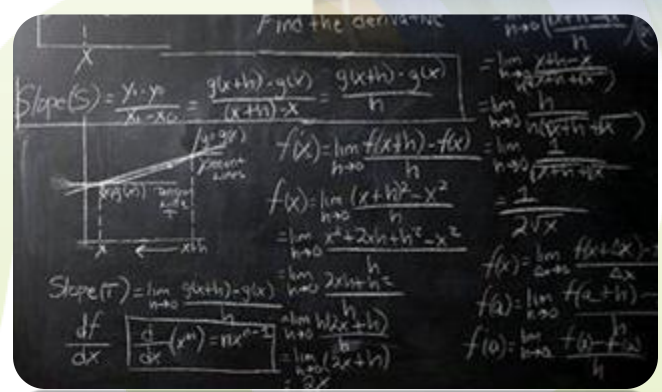
Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de

Overview

- Relational tuple calculus
  - SQUARE, SEQUEL
- Domain tuple calculus
  - Query-by-example (QBE)

$$F_2(t_1) \equiv \exists t_3\, (SC(t_3) \land t_3.matNr = t_1.matNr$$
$$\land\ t_3.crsNr = t_2.crsNr)$$



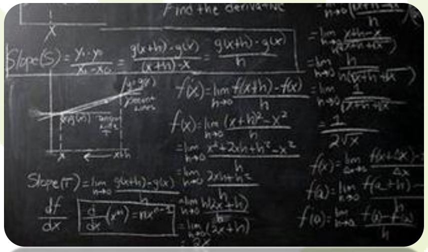


7.1 Summary of Last Week

- **Basic relational algebra**
  - **Selection σ**
  - **Projection π**
  - **Renaming ρ**
  - **Union ∪, intersection ∩,** and **set difference **
  - **Cartesian product ×**
- **Extended relational algebra**
  - **Theta-join** $\bowtie_{(\theta\text{-cond})}$, **Equi-join** $\bowtie_{(=\text{-cond})}$, **Natural join ⋈**
  - **Left semi-join ⋉** and **right semi-join ⋊**
  - **Division ÷**
- **Advanced relational algebra**
  - **Left outer join ⟕, right outer join ⟖, full outer join ⟗**
  - **Aggregation** $\mathfrak{F}$

7.1 Introduction

- Beside the **relational algebra,** there are two other major **query** paradigms within the relational model
  - **Tuple relational calculus** (TRC)
  - **Domain relational calculus** (DRC)
- All three provide the **theoretical** foundation of the relational database model
- They are mandatory for certain DB features:
  - Relational algebra → **Query optimization**
  - TRC → **SQL** query language
  - DRC → **Query-by-example** paradigm



7.1 Introduction

- Relational algebra has some **procedural** aspects
  - You specify an **order of operations** describing how to retrieve data
- Relational calculi (TRC, DRC) are **declarative**
  - You just **specify** how the desired **tuples look like**
  - The query contains no information about how to create the result set
  - Provides an alternative approach to querying

7.1 Introduction

- Both calculi are special cases of the **first-order predicate calculus**
  - **TRC** = logical expressions on **tuples**
  - **DRC** = logical expressions on **attribute domains**

7.2 Tuple Relational Calculus

- **TRC**
  - Describe the **properties** of the desired **tuples**
  - "Get all students *s* for that there is an exam report *r* such that *s*' student number is the same as the student number mentioned in *r*, and the result mentioned in *r* is better than 2.7"

7.2 Tuple Relational Calculus

- **Queries in TRC:**
  - { *t* | CONDITION(*t*) }
  - *t* is a **tuple variable**
    - *t* usually **ranges over** all tuples of a relation
    - *t* may take the value of **any tuple**
  - CONDITION(*t*) is a **logical statement** involving *t*
    - All those tuples *t* are retrieved that satisfy CONDITION(*t*)
  - Reads as:
    "Retrieve all tuples *t* for that CONDITION(*t*) is true"

7.2 Tuple Relational Calculus

- **Example:** Select all female students

$\{ t \mid Student(t) \land t.sex = \text{'f'} \}$

Range = relation "Student"

Condition for result tuples

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m
6676	Erik	Magnus	m
8024	Jeanne	Gray	f
9876	Logan		m

This type of expression resembles relational algebra's selection!

7.2 Tuple Relational Calculus

- It is possible to retrieve only a subset of attributes
  - The **request attributes**
- **Example:** Select the names of all female student

  { *t*.firstName, t.lastName | Student(*t*) and *t*.sex = 'f' }

  Result attributes

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m
6676	Erik	Magnus	m
8024	Jeanne	Gray	f
9876	Logan		m

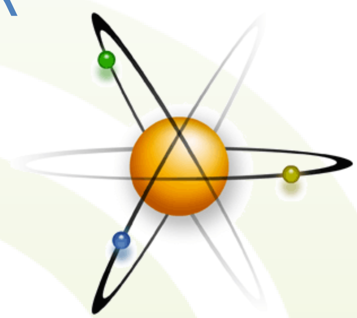
This type of expression resembles relational algebra's projection!

7.2 Tuple Relational Calculus

- **Full query syntax:**
  - $\{ t_1.A_1, t_2.A_2, \ldots, t_n.A_n \mid CONDITION(t_1, t_2, \ldots, t_n) \}$
  - $t_1, t_2, \ldots, t_n$ are **tuple variables**
  - $A_1, A_2, \ldots, A_n$ are **attributes,** where A_i is an attribute of tuple t_i
  - *CONDITION* specifies a condition on tuple variables
    - More precise (to be defined in detail later): *CONDITION* is a **formula** with free variables $t_1, t_2, \ldots, t_n$
  - The **result** is the set of all tuples $(t_1.A_1, t_2.A_2, \ldots, t_n.A_n)$ fulfilling the formula $CONDITION(t_1, t_2, \ldots, t_n)$

7.2 Tuple Relational Calculus

- What is a **formula?**
  - A formula is a logical expression made up of **atoms**
- Atom types
  - **Range atom** $R(t)$
    - Evaluates if a tuple is an element of the relation R
      - "Binds R to the tuple variable t_i as **range relation**"
    - **Example:** Student(t)
  - **Comparison atom** $(s.A \ \theta \ t.B)$
    - Provides a simple condition based on comparisons
    - s and t are tuple variables, A and B are attributes
    - θ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$
    - **Example:** t_1.id = t_2.id

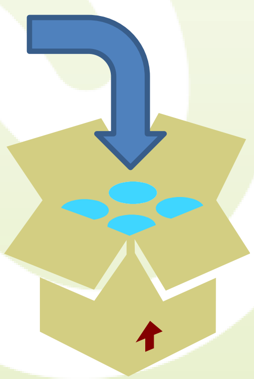


7.2 Tuple Relational Calculus

– **Constant comparison atom**
  $(t.A \;\; \theta \;\; c)$ or $(c \;\; \theta \;\; t.A)$
  - A simple condition comparing an attribute value to some constant
  - t is a tuple variable, A is an attribute, c is a constant
  - θ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$
  - **Example:** t_1.name = 'Peter Parker'

7.2 Tuple Relational Calculus

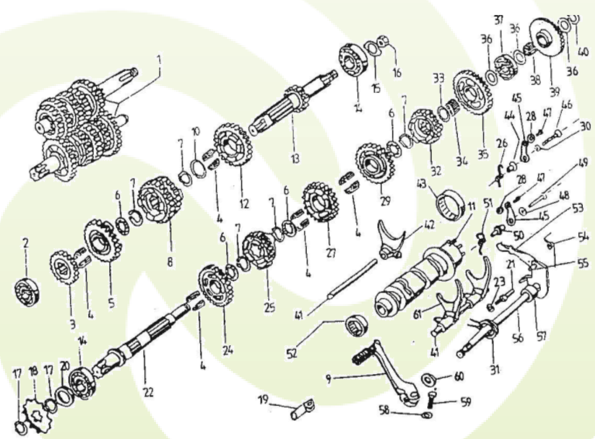
- Tuple variables have to be **substituted** by tuples
- For each substitution, atoms evaluate either to **true** or **false**
  - **Range atoms** are true, iff a tuple variable's value is an element of the **range relation**
  - **Comparison atoms** are either true or false for the currently substituted tuple variable values



7.2 Tuple Relational Calculus

- **Formulas** are defined recursively by four rules
  1. Every **atom** is a formula
  2. If F_1 and F_2 are formulas, then also the following are formulas:
     - $(F_1 \land F_2)$: true iff both F_1 and F_2 are true
     - $(F_1 \lor F_2)$: false iff both F_1 and F_2 are false
     - $\neg F_1$: false iff F_1 is true

Rules 3 and 4 on later slides …



7.2 Tuple Relational Calculus

- **Evaluating formulas:**
  - TRC relies on the so-called **open world** assumption
    - That is, every substitution for variables is possible
- Evaluating $\{ t_1, \ldots, t_n \mid F(t_1, \ldots, t_n) \}$
  - **Substitute** all tuple variables in *F* by all combinations of all possible tuples
    - Open world: Really, all!
    - Also all really stupid ones!
    - **ALL!**
  - Put all those tuple combinations for that *F* is true into the **result set**

7.2 Tuple Relational Calculus

- Example: { t | Student(t) ∧ firstName = 'Clark' }
  - Substitute t, one after another, with all possible tuples
    - <>, <1>, <2>, … , <1005, Clark, Kent, m>, …, <Hurz!, Blub, 42, Balke, Spiderman>, …
    - Open world!
  - Of course, the formula will only be true for those tuples in the students relation
    - Great way of saving work: Bind t one after another to all tuples which are contained in the students relation
    - Only those tuples (in students) whose firstName value is "Clark" will be returned

7.2 Tuple Relational Calculus

- **Example**: All male students with student number greater than 6000
  - $\{ t \mid \text{Student}(t) \land t.\text{matNr} > 6000 \land t.\text{sex} = \text{'m'} \}$
  - Evaluate formula for every tuple in students

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m
6676	Erik	Magnus	m
8024	Jeanne	Gray	f
9876	Logan		m

true ∧ false ∧ true = false

true ∧ false ∧ false = false

Result tuples

true ∧ true ∧ true = true

true ∧ true ∧ false = false

7.2 TRC: Examples

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m
6676	Erik	Magnus	m
8024	Jeanne	Gray	f
9876	Logan		m

Course

crsNr	title
100	Intro. to being a Superhero
101	Secret Identities 2
102	How to take over the world

exam

student	course	result
9876	100	3.7
2832	102	2.0
1005	101	4.0
1005	100	1.3
6676	102	4.3
5119	101	1.7

7.2 TRC: Examples

- Selection

"Select all female students"

$\sigma_{sex = 'f'}$ Student

{ t | Student(t) ∧ t.sex='f' }

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m
6676	Erik	Magnus	m
8024	Jeanne	Gray	f
9876	Logan		m

matNr	firstName	lastName	sex
2832	Louise	Lane	f
8024	Jeanne	Gray	f

7.2 TRC: Examples

"Retrieve first name and last name of all female students"

$\pi_{\text{firstName, lastName}}\ \sigma_{\text{sex='f'}}\ \text{Student}$

$\{t.\text{firstName}, t.\text{lastName} \mid \text{Student}(t) \land t.\text{sex}=\text{'f'}\}$

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m
6676	Erik	Magnus	m
8024	Jeanne	Gray	f
9876	Logan		m

$\{ t \mid \text{Student}(t) \land t.\text{sex}=\text{'f'} \}$

matNr	firstName	lastName	sex
2832	Louise	Lane	f
8024	Jeanne	Gray	f

$\{ t.\text{firstName}, t.\text{lastName} \mid \text{Student}(t) \land t.\text{sex}=\text{'f'} \}$

firstName	lastName
Loise	Lane
Jeanne	Gray

7.2 TRC: Examples

"Compute the union of all courses with id 100 and 102"

$\sigma_{crsNr = 100}\, Course \;\cup\; \sigma_{crsNr = 102}\, Course$

$\{\, t \mid Course(t) \land (t.crsNr = 100 \;\lor\; t.crsNr = 102) \,\}$

"Get all courses with an id greater than 100, excluding those with an id of 102"

$\sigma_{crsNr > 100}\, Course \setminus \sigma_{crsNr = 102}\, Course$

$\{\, t \mid Course(t) \land (t.crsNr > 100 \;\land\; \neg\, t.crsNr = 102) \,\}$

$\sigma_{crsNr=100}\, Course$

crsNr	title
100	Intro. to being a Superhero

$\sigma_{crsNr=102}\, Course$

crsNr	title
102	How to take over the world

$\sigma_{crsNr=100}\, Course \cup \sigma_{crsNr=102}\, Course$

crsNr	title
100	Intro. to being a Superhero
102	How to take over the world

7.2 TRC: Examples

"Compute the cross product of students and exams"

$$\text{Student} \times \text{exam}$$

$$\{\, t_1, t_2 \mid \text{Student}(t_1) \land \text{exam}(t_2) \,\}$$

So, TRC can obviously do the same as relational algebra and vice versa…

"Compute a join of students and exams"

$$\text{Student} \bowtie_{\text{matNr}=\text{student}} \text{exam}$$

$$\{\, t_1, t_2 \mid \text{Student}(t_1) \land \text{exam}(t_2) \land t_1.\text{matNr} = t_2.\text{student} \,\}$$

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m

exam

student	course	result
9876	100	3.7
2832	102	2.0
1005	101	4.0
1005	100	1.3

7.2 Tuple Relational Calculus

- Additionally, in TRC there can be formulas considering all tuples
  - **Universal quantifier ∀**
    - Can be used with a formula that evaluates to true if the **formula is true for all tuples**
    - "All students have passed the exam"
  - **Existential quantifier ∃**
    - Can be used with a formula that evaluates to true if the **formula is true for at least one tuple**
    - "There are students who passed the exam"

7.2 Tuple Relational Calculus

- With respect to quantifiers, **tuple variables** can be either **free (unbound)** or **bound**
  - If F is an **atom** (and thus also a formula), each tuple variable occurring in F is **free** within F
    - Example:
      - $F = (t_1.\mathrm{crsNr} = t_2.\mathrm{crsNr})$
      - Both t_1 and t_2 are free in F
  - If t is a **free tuple** variable in F, then it can be **bound** in formula F' either by
    - $F' = \forall t\ (F)$, or
    - $F' = \exists t\ (F)$
      - t is free in F and bound in F'

7.2 Tuple Relational Calculus

- If F_1 and F_2 are formulas combined by
  $$F' = (F_1 \land F_2) \text{ or } F' = (F_1 \lor F_2)$$
  and t is a **tuple variable** occurring in F_1 and/or F_2, then
  - t in is **free in F'** if it is **free in both F_1** and **F_2**
  - t in is **free in F'** if it is **free in one** of **F_1** and **F_2** but does **not occur** in the other
  - If t is bound in both **F_1** and **F_2**, t is also **bound in F'**
  - If t is bound in one of **F_1** and **F_2** but free in the other, one says that t is **bound and unbound in F'**
- The last two cases are a little complicated and should be avoided altogether by renaming the variables (see next slides)

7.2 Tuple Relational Calculus

- If a formula contains no free variables, it is called **closed.** Otherwise, it is called **open.**
  - Open formulas should denote all **free variables as parameters**
    - The truth value of open formulas depends on the value of free variables
    - Closed formulas do not depend on specific variable values, and are thus constant

  - Example:
    - $F_1(t_1, t_2)$ is open and has t_1 and t_2 as free variables
    - $F_2()$ is closed and has no free variables

7.2 Tuple Relational Calculus

- Examples:
  - $F_1(t_1) = (t_1.\text{name} = \text{'Clark Kent'})$
    - t_1 is free, F_1 is open
  - $F_2(t_1, t_2) = (t_1.\text{matNr} = t_2.\text{matNr})$
    - t_1 and t_2 are free, F_2 is open
  - $F_3(t_1) = \exists t_2(F_2(t_1,t_2)) = \exists t_2(t_1.\text{matNr} = t_2.\text{matNr})$
    - t_1 is free, t_2 is bound, F_3 is open
  - $F_4() = \exists t_1(t_1.\text{sex} = \text{'female'})$
    - t_1 is bound, F_4 is closed

7.2 Tuple Relational Calculus

- Examples:
  - $F_1(t_1) = (t_1.\text{name} = \text{'Clark Kent'})$
  - $F_3(t_1) = \exists t_2(F_2(t_1, t_2)) = \exists t_2(t_1.\text{matNr} = t_2.\text{matNr})$
  - $F_5(t_1) = F_1(t_1) \wedge F_3(t_1)$

    $= \big(t_1.\text{name} = \text{'Clark Kent'}$

    $\wedge \exists t_2(t_1.\text{matNr} = t_2.\text{matNr})\big)$

    - t_1 is free, t_2 is bound, F_5 is open

7.2 Tuple Relational Calculus

- Examples:
  - $F_1(t_1) = (t_1.\text{name} = \text{'Clark Kent'})$
  - $F_4() = \exists t_1 (t_1.\text{sex} = \text{'female'})$
  - $F_6(t_1) = F_1(t_1) \land F_4()$
    $= \big(t_1.\text{name} = \text{'Clark Kent'} \land \exists t_1 (t_1.\text{sex} = \text{'female'})\big)$
    - t_1 is free, t_1 is also bound, F_6 is open
- In F_6, t_1 is **bound and unbound** at the same time
  - Actually, the t_1 in F_4 is **different** from the t_1 in F_1 because F_4 is **closed**
    - The t_1 of F_4 is only valid in F_4, thus it could (and should!) **renamed** without affecting F_1

- Convention:
  Avoid conflicting variable names!
  - **Rename** all conflicting **bound** tuple variables when they are combined with another formula

- **Examples**:
  - $F_1(t_1) = (t_1.\text{name} = \text{‘Clark Kent’})$
  - $F_4() = \exists t_1(t_1.\text{sex} = \text{‘female’}) \equiv \exists \boldsymbol{t_2}(\boldsymbol{t_2}.\text{sex} = \text{‘female’})$
  - $F_7(t_1) = F_1(t_1) \land F_4()$
    $\equiv \big(t_1.\text{name} = \text{‘Clark Kent’} \land \exists t_2(t_2.\text{sex} = \text{‘female’})\big)$
    - t_1 is free, t_2 is bound, F_7 is open

7.2 Tuple Relational Calculus

- What are **formulas?**
  1. Every atom is a formula
  2. If F_1 and F_2 are formulas,
     then also their logical combination are formulas
  3. If F is an open formula with the free variable t,
     then $F' = \exists t(F)$ is a formula
     - F' is true, if there is **at least one** tuple such that F is true
  4. If F is an open formula with the free variable t,
     then $F' = \forall t(F)$ is a formula
     - F' is true, if F is true **for all** tuples

7.2 Tuple Relational Calculus

- Thoughts on quantifiers:
  - Any formula with an **existential quantifier** can be **transformed** into one with an **universal quantifier** and vice versa
  - Quick rule: Replace $\vee$ by $\wedge$ and negate everything
    - $\forall t\ (F(t)) \equiv \neg\exists t\ (\neg F(t))$
    - $\exists t\ (F(t)) \equiv \neg\forall t\ (\neg F(t))$
    - $\forall t\ (F_1(t) \wedge F_2(t)) \equiv \neg\exists t\ (\neg F_1(t) \vee \neg F_2(t))$
    - $\forall t\ (F_1(t) \vee F_2(t)) \equiv \neg\exists t\ (\neg F_1(t) \wedge \neg F_2(t))$
    - $\exists t\ (F_1(t) \wedge F_2(t)) \equiv \neg\forall t\ (\neg F_1(t) \vee \neg F_2(t))$
    - $\exists t\ (F_1(t) \vee F_2(t)) \equiv \neg\forall t\ (\neg F_1(t) \wedge \neg F_2(t))$

7.2 Tuple Relational Calculus

- More considerations on **evaluating** TRC:
  What happens to **quantifiers** and **negation**?
  - Again: Open world!
- Consider relation students

matNr	name	sex
1776	Leni Zauber	f
8024	Jeanne Gray	f

  - **∃*t* (*t*.sex = 'm')** ≡ true
    - *t* can represent any tuple, and there can be a tuple for that the condition holds, e.g. <0012, Scott Summers, m> or <-1, &cjndks, m>
  - **∃*t* (Student(*t*) ∧ *t*.sex = 'm')** ≡ false
    - There is no male tuple in Student
  - **∀*t* (*t*.sex = 'f')** ≡ false
  - **∀*t* (¬Student(*t*) ∨ *t*.sex = 'f')** ≡ true
    - All tuples are either female or they are not in Student
    - "All tuples in the relation are girls"

7.2 TRC: Examples

"List the names of all students that took some exam"

$$\pi_{\text{firstName}}\,(\text{Student} \ltimes_{\text{matNr=student}} \text{exam})$$

$$\{\ t_1.\text{firstName} \mid \text{Student}(t_1) \land \exists t_2(\text{exam}(t_2) \land t_2.\text{matNr} = t_1.\text{student})\ \}$$

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m

exam

student	course	result
9876	100	3.7
2832	102	2.0
1005	101	4.0
1005	100	1.3

7.2 TRC: Examples

"List students having at least those exams Clark Kent had"

$$SC \div (\pi_{crsNr}\, \sigma_{name = \text{'Clark Kent'}}\, SC)$$

$$\{\, t_1.\text{matNr}, t_1.\text{name} \mid SC(t_1) \land F_1(t_1) \,\}$$

$$F_1(t_1) = \forall t_2\, (\neg SC(t_2) \lor \neg t_2.\text{name} = \text{'Clark Kent'} \lor F_2(t_1))$$

$$F_2(t_1) = \exists t_3\, (SC(t_3) \land t_3.\text{matNr} = t_1.\text{matNr} \land t_3.\text{crsNr} = t_2.\text{crsNr})$$

For all tuples of Clark Kent, F_2 is true

There is a tuple of the same student originally selected who has the same course than the currently selected tuple of Clark Kent in F_2

SC (= students and courses)

matNr	name	crsNr
1000	Clark Kent	100
1000	Clark Kent	102
1001	Louise Lane	100
1002	Lex Luther	102
1002	Lex Luther	100
1002	Lex Luther	101
1003	Charles Xavier	103
1003	Charles Xavier	100

Result

matNr	name
1000	Clark Kent
1002	Lex Luther

7.2 Tuple Relational Calculus

- Consider the TRC query **{ *t* | ¬Student(*t*) }**
  - This query returns **all** tuples which are not in the students relation …
  - The number of such tuples is **infinite!**
  - All queries that eventually return an infinite number of tuples are called **unsafe**
- **Unsafe** queries have to be **avoided** and cannot be evaluated (reasonably)!
  - One reliable way of avoiding unsafe expressions is the **closed world assumption**

7.2 Tuple Relational Calculus

- The **closed world** assumption states that only those **tuples** may be **substitutes** for tuple variables that **are actually present** in the current **relations**
  - Assumption usually not applied to TRC
  - However, is part of most applications of TRC like **SEQUEL** or **SQL**
  - Removes the need of explicitly dealing with unknown tuples when **quantifiers** are used
  - However, it's a restriction of expressiveness

7.2 Tuple Relational Calculus

- **Open world** vs. **closed world**

matNr	name	sex
1776	Leni Zauber	f
8024	Jeanne Gray	f

Expression	Open World	Closed World
$\exists t\ (t.\text{sex} = \text{'m'})$	true	false
$\exists t\ (\text{Student}(t) \wedge t.\text{sex} = \text{'m'})$	false	false
$\forall t\ (t.\text{sex} = \text{'f'})$	false	true
$\forall t\ (\neg\text{Student}(t) \vee t.\text{sex} = \text{'f'})$	true	true

7.2 Tuple Relational Calculus

- "Why did we do this weird calculus?"
  - Because it is the foundation of **SQL**, the standard language for database querying!

The obvious may escape many …

7.3 SQUARE & SEQUEL

Detour

- The design of relational query languages
  - Donald D. **Chamberlin** and Raymond F. **Boyce** worked on this task
  - Both of **IBM Research** in San Jose, California
  - Main concern: "**Querying** relational databases is **too difficult** with current paradigms"

7.3 SQUARE & SEQUEL

Detour

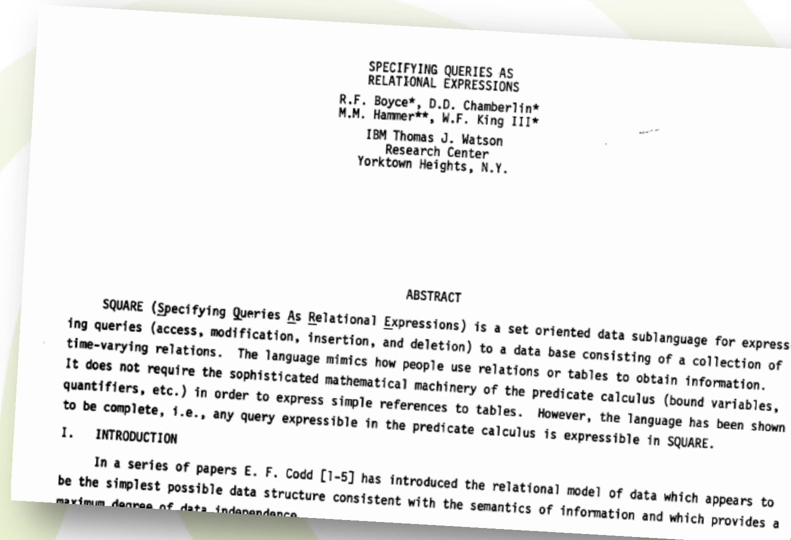
- "Current paradigms" at the time:
  - **Relational algebra**
    - Requires users to define **how** and in **which order** data should be retrieved
    - The specific choice of a sequence of operations has an enormous influence on the system's performance
  - **Relational calculi (tuple, domain)**
    - Provide **declarative** access to data, which is good
    - Just state **what you want** and not how to get it
    - Relational calculi are **quite complex:** many variables and quantifiers

7.3 SQUARE & SEQUEL

Detour

- Chamberlin and Boyce's first result was a query language called **SQUARE**
  - "**S**pecifying **qu**eries **a**s **r**elational **e**xpressions"
  - Based directly on **tuple relational calculus**
  - Main observations:
    - Most database **queries are rather simple**
    - Complex queries are rarely needed
    - Quantification confuses people
    - Under the **closed-world assumption,** any **TRC expression with quantifiers** can be replaced by **a join of quantifier-free expressions**

- SQUARE is a notation for (or interface to) TRC
  - **No quantifiers,** implicit notation of variables
  - Adds **additional functionality** needed in practice (grouping, aggregating, among others)
  - Solves **safety problem** by introducing the **closed world assumption**



SPECIFYING QUERIES AS
RELATIONAL EXPRESSIONS

R.F. Boyce*, D.D. Chamberlin*
M.M. Hammer**, W.F. King III*

IBM Thomas J. Watson
Research Center
Yorktown Heights, N.Y.

ABSTRACT

SQUARE (Specifying Queries As Relational Expressions) is a set oriented data sublanguage for express-ing queries (access, modification, insertion, and deletion) to a data base consisting of a collection of time-varying relations. The language mimics how people use relations or tables to obtain information. It does not require the sophisticated mathematical machinery of the predicate calculus (bound variables, quantifiers, etc.) in order to express simple references to tables. However, the language has been shown to be complete, i.e., any query expressible in the predicate calculus is expressible in SQUARE.

I. INTRODUCTION

In a series of papers E. F. Codd [1-5] has introduced the relational model of data which appears to be the simplest possible data structure consistent with the semantics of information and which provides a maximum degree of data independence.

7.3 SQUARE & SEQUEL

Detour

- Retrieve the names of all female students
  - TRC: { *t*.name | Student(*t*) ∧ *t*.sex = 'f' }
  - SQUARE: ${}_{\text{name}}\text{Student}_{\text{sex}}$ ('f')

Conditions

What part of the result tuples should be returned?

The range relation of the result tuples

Attributes with conditions

- Get all exam results better than 2.0 in course 101
  - TRC:

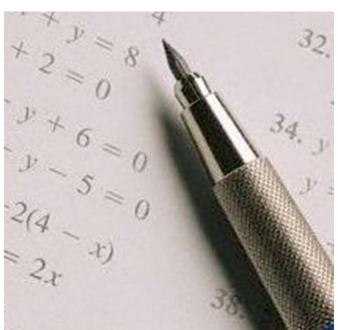
    { *t*.result | exam(*t*) ∧ *t*.course = 101 ∧ *t*.result < 2.0}
  - SQUARE: ${}_{\text{result}}\text{exam}_{\text{course, result}}$ (101, <2.0)

- Get a list of all exam results better than 2.0 along with the according student name
  - TRC:

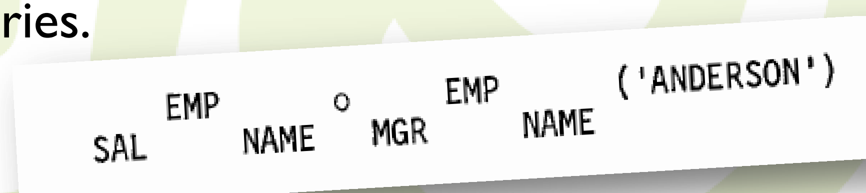
    $\{\ t_1.\text{name}, t_2.\text{result} \mid \text{Student}(t_1) \wedge \text{exam}(t_2) \wedge t_1.\text{matNr} = t_2.\text{student} \wedge t_2.\text{result} < 2.0\ \}$
  - SQUARE:

    ${}_{\text{name result}}\,\text{Student}_{\text{matNr}} \circ {}_{\text{student}}\text{exam}_{\text{result}}\,(<2.0)$

Join of two SQUARE queries



Also, ∪, ∩, and \ can be used to combine SQUARE queries.



7.3 SQUARE & SEQUEL

Detour

- Also, SQUARE is **relationally complete**
  - You do not need explicit quantifiers
  - Everything you need can be done using conditions and query combining
- However, SQUARE was not well received
  - Syntax was **difficult to read and parse,** especially when using text console devices:
    - `name result Student matNr ∘ student exams crsNr result (102, <2.0)`
  - SQUARE's syntax is too mathematical and artificial

7.3 SQUARE & SEQUEL

Detour

- In 1974, Chamberlin & Boyce proposed **SEQUEL**
  - **S**tructured **E**nglish **Que**ry **L**anguage
  - Based on SQUARE
- Guiding principle:
  - Use natural **English keywords** to structure queries
  - Supports **“fluent”** vocalization and notation

> A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

7.3 SQUARE & SEQUEL

Detour

- Fundamental keywords
  - **SELECT:** What attributes should be retrieved?
  - **FROM:** What relations are involved?
  - **WHERE:** What conditions should hold?

> SEQUEL presents the user with a consistent template for expression of simple queries. The user must specify the columns he wishes to SELECT, the table FROM which the query columns are to be chosen, and the conditions WHERE the rows are to be returned. The SELECT-FROM-WHERE block is the basic component of the language. In an interactive system this template might be presented to the user, who then fills in the blanks.

7.3 SQUARE & SEQUEL

Detour

- Get all exam results better than 2.0 for course 101
  - SQUARE:

$$_{\text{result}}\text{exam}_{\text{course result}}\ (101, < 2.0)$$

  - SEQUEL:

SELECT result **FROM** exam
WHERE course = 101 **AND** result < 2.0

7.3 SQUARE & SEQUEL

Detour

- Get a list of all exam results better than 2.0, along with the according student names
  - SQUARE:

    $${}_{\text{name result}}\ \text{Student}\ {}_{\text{matNr}} \circ {}_{\text{student result}}\ \text{exam}_{\text{result}}\ (< 2.0)$$

  - SEQUEL:

    SELECT name, result
    FROM Student, exam
    WHERE Student.matNr = exam.student
    AND result < 2.0

7.3 SQUARE & SEQUEL

Detour

- IBM integrated SEQUEL into **System R**
- It proved to be a **huge success**
  - Unfortunately, the name SEQUEL already has been registered as a **trademark** by the Hawker Siddeley aircraft company
  - Name has been changed to **SQL** (spoken: Sequel)
    - **S**tructured **q**uery **l**anguage
  - Patented in 1985 by IBM

HAWKER SIDDELEY

7.3 SQUARE & SEQUEL

Detour

- Since then, SQL has been adopted by all(?) relational database management systems
- This created a need for **standardization:**
  - 1986: SQL-86 (ANSI standard, ISO standard)
  - SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008
  - The official pronunciation is “es queue el”
- However, most database vendors treat the standard as some kind of “recommendation”
  - More on this later (next lecture)

7.4 Domain Relational Calculus

- The **domain relational calculus** is also a calculus like TRC, but
  - **Variables** are different
  - **TRC: Tuple variables** ranging over all tuples
  - **DRC: Domain variables** ranging over the values of the domains of individual attributes
- **Query form**
  - $\{ x_1, \ldots, x_n \mid CONDITION(x_1, \ldots, x_n) \}$
  - $x_1, \ldots, x_n$ are **domain variables**
  - *CONDITION* is a **formula** over the domain variables, where $x_1, \ldots, x_n$ are *CONDITION*'s free variables

7.4 Domain Relational Calculus

- DRC also defines formula **atoms**
  - **Relation atoms:** $R(x_1, x_2, \ldots, x_n)$
    - Also written without commas as $R(x_1x_2\ldots x_n)$
    - R is a n-ary relation
    - $x_1, \ldots, x_n$ are (all) domain variables of R
    - Atom evaluates to true iff, for a list of attribute values, an according tuple is in the relation R
  - **Comparison atoms:** $(x\ \theta\ y)$
    - x_i and x_j are domain variables
    - θ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$
  - **Comparison atoms:** $(x\ \theta\ c)$ or $(c\ \theta\ x)$
    - x is a domain variable, c is a constant value
    - θ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$

7.4 Domain Relational Calculus

- The **recursive construction** of DRC **formulas** is analogous to TRC
  1. Every atom is a formula
  2. If F_1 and F_2 are formulas, then also their logical combinations are formulas
  3. If F is a open formula with the free variable x, then $\exists x(F)$ is a formula
  4. If F is a open formula with the free variable x, then $\forall x(F)$ is a formula
- Other aspects of DRC are similar to TRC

7.4 DRC: Examples

"Retrieve first name and last name of all female students"

Algebra: $\pi_{\text{firstName, lastName}}\ \sigma_{\text{sex = 'f'}}$ Student

TRC: { *t*.firstName, *t*.lastName | Student(*t*) ∧ *t*.sex = 'f' }

$\{\ fn, ln \mid \exists mat, s\ (\text{Student}(mat, fn, ln, s) \land s = \text{'f'})\ \}$

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m
6676	Erik	Magnus	m
8024	Jeanne	Gray	f
9876	Logan		m

{ *mat*, *fn*, *ln*, *s* | Student(*mat*, *fn*, *ln*, *s*) ∧ s='f'}

matNr	firstName	lastName	sex
2832	Louise	Lane	f
8024	Jeanne	Gray	f

{ *fn*, *ln* | ∃*mat*, *s* (Student(*mat*, *fn*, *ln*, s) ∧ s='f') }

firstName	lastName
Louise	Lane
Jeanne	Gray

7.4 DRC: Examples

"List the first names of all students that took at least one exam"

Algebra: $\pi_{\text{firstName}}$ (Student $\ltimes_{\text{matNr=student}}$ exam)

TRC: $\{\ t_1.\text{firstName} \mid \text{Student}(t_1) \land \exists t_2(\text{exam}(t_2) \land t_2.\text{student} = t_1.\text{matNr})\ \}$

DRC: $\{\ fn \mid \exists mat, ln, s\ (\text{Student}(mat, fn, ln, s) \land \exists st, co, r\ (\text{exam}(st, co, r) \land st{=}mat))\ \}$

Student

matNr	firstName	lastName	sex
1005	Clark	Kent	m
2832	Louise	Lane	f
4512	Lex	Luther	m
5119	Charles	Xavier	m

exam

student	course	result
9876	100	3.7
2832	102	2.0
1005	101	4.0
1005	100	1.3

7.4 Domain Relational Calculus

- In DRC, a lot of **existential** quantification is used in conjunction with **equality** comparisons
  - $\{fn, ln \mid \exists\ \boldsymbol{mat, s}\ (\text{Student}(mat, fn, ln, s) \land \boldsymbol{s} = \text{'}\mathbf{f}\text{'})\}$
  - $\{fn \mid \exists\ \boldsymbol{mat, ln, r}\ (\text{Student}(mat, fn, ln, r) \land \exists\ \boldsymbol{st, co, r}\ (\text{exam}(st, co, r) \land \boldsymbol{st} = \boldsymbol{mat}))\}$
- As a shorthand, a **formally inaccurate** notation is sometimes used
  - Pull equality comparisons into domain atoms
  - Use implicit existential quantifiers
  - $\{fn, ln \mid \text{Student}(mat, fn, ln, \text{'}\mathbf{f}\text{'})\}$
  - $\{fn \mid \text{Student}(\boldsymbol{mat}, fn, ln, s) \land \exists \text{co, r}(\text{exam}(\boldsymbol{mat}, co, r))\}$

7.5 QBE

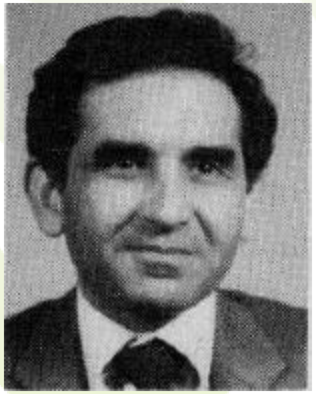
Detour

- The first version of SQL, **SEQUEL**, was developed in early 1970 by D. **Chamberlin** and R. **Boyce** at **IBM** Research in **San Jose,** California
  - Based on the **tuple relational calculus**
- At the same time, another query language, **QBE,** was developed independently by M. **Zloof** at IBM Research in **Yorktown Heighs,** New York
  - Based on the **domain relational calculus (DRC)**

7.5 QBE

Detour

- **Query by Example** (QBE) is an alternative database query language for relational databases
- First **graphical query language**
  - It used visual tables where the user would enter commands, example elements and conditions
  - Based on the domain relational calculus
- Devised by Moshé M. Zloof at IBM Research during the mid-1970s



7.5 QBE

Detour

- QBE has a **two dimensional syntax**
  - Queries look like tables
- QBE queries are expressed **“by example”**
  - Instead of formally describing the desired answer, the user gives an example of what is desired
- This was of course much **easier for users** than specifying difficult logical formulae
  - “The age of the nonprogrammer user of computing systems is at hand, bringing with it the special need of persons who are professionals in their own right to have easy ways to use a computing system.”
    - M. Zloof: Office-by-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail. IBM Systems Journal, Vol. 21(3), 1982

7.5 QBE

Detour

- **Skeleton tables** show the relational schema of the database
  - Users **select the tables** needed for the query and fill the table with example rows
    - Example rows consist of **constants** and **example elements** (i.e. domain variables)
    - Domain variables are denoted beginning with an underscore
  - **Conditions** can be written in a special **condition box**
  - **Arithmetic comparisons,** including **negation,** can be written directly into the rows
  - To **project** any attribute 'P.' is written before the domain variable

7.5 QBE

Detour

$\pi_{\text{matNr}}\sigma_{\text{lastName = 'Parker'}}$ Student

{ mat | ∃ fn, ln, s (Student(mat, fn, 'Parker', s)) }

Student	matNr	firstName	lastName	sex
	P.		Parker	

QBE

{ st | ∃co, r (exam(st, co, r) ∧ r > 2.0) }

exam	student	course	result
	P.		> 2.0

{ st | ∃ co, r (exam(st, co, r) ∧ co ≠ 102 }

exam	student	course	result
	P.	≠ 102	

7.5 QBE

Detour

- Add a row if you need to connect conditions
  - Get the matNr of students who took exams in courses 100 **and** 102
  - $\{st \mid \exists r\,(exam(st, 100, r)) \wedge \exists r(exam(st, 102, r))\}$

exam	student	course	result
	P._12345	100	
	_12345	102	

_12345 is the "example" in "query-by-example"!

  - Get the matNr of students who took exams in course 100 **or** 102

exam	student	course	result
	P.	100	
	P.	102	

7.5 QBE

Detour

- Get the matNr of students who took the same course as the student with matNr 1005

exam	student	course	result
	P._12345	_54321	
	1005	_54321	

– Also grouping (G.) and aggregate functions (in an additional column) are supported

- Get the average results of each student

exam	student	course	result	
	G.P._12345		_2.0	P.AVG._2.0

7.5 QBE

Detour

- This can of course also be applied between tables
  - Analogous to joins in relational algebra
  - Example: What are the lastNames of all females who got a very good grade in some exam?

Student	matNr	firstName	lastName	sex
	_12345		P.	f

exam	student	course	result
	_12345		< 1.3

7.5 QBE

Detour

- Besides the DML aspect for querying also the DDL aspect is covered
  - Single tuple insertion

Student	matNr	firstName	lastName	sex
I.	1005	Clark	Kent	m

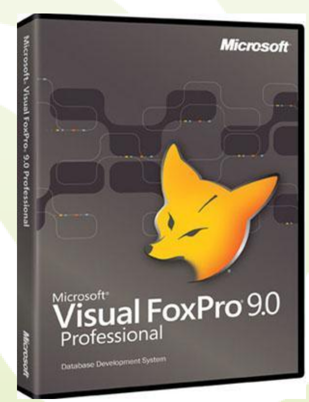
  - Or from other tables by connecting them with domain variables
  - Insert (I.), delete (D.), or update (U.)
    - Update even in columns

exam	student	course	result
	2832	102	U._2.0 +1.0

7.5 QBE

Detour

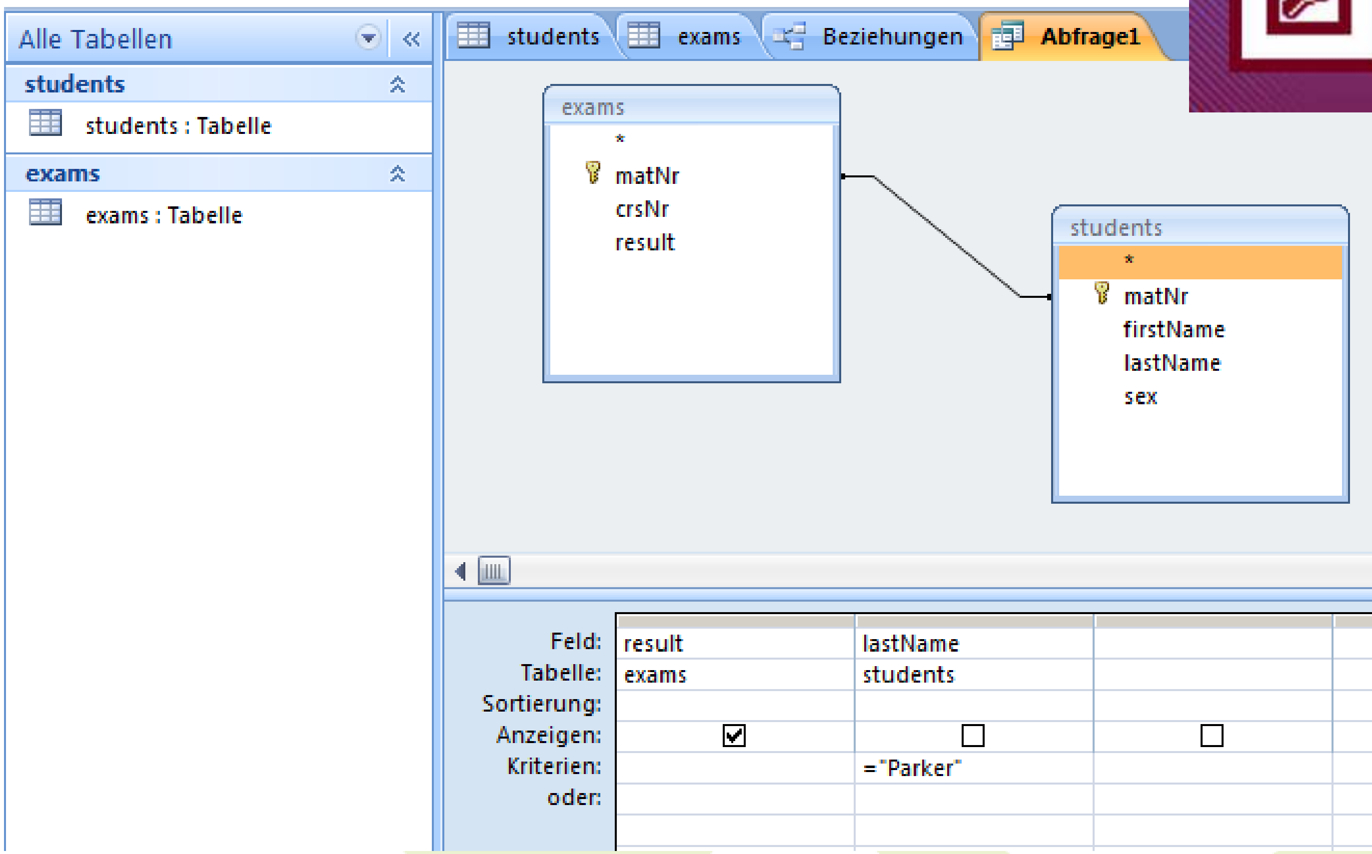
- The graphical query paradigm was transported into databases for end users
  - "Desktop databases" like Microsoft Access, Fox Pro, Corel Paradox, etc.
  - Tables are shown on a query design grid
  - Lines can be drawn between attributes of two tables instead of a shared variable to specify a join condition



7.5 QBE

Detour

- Example: What results did the student with last name Parker get in the exams?



7.5 QBE

Detour

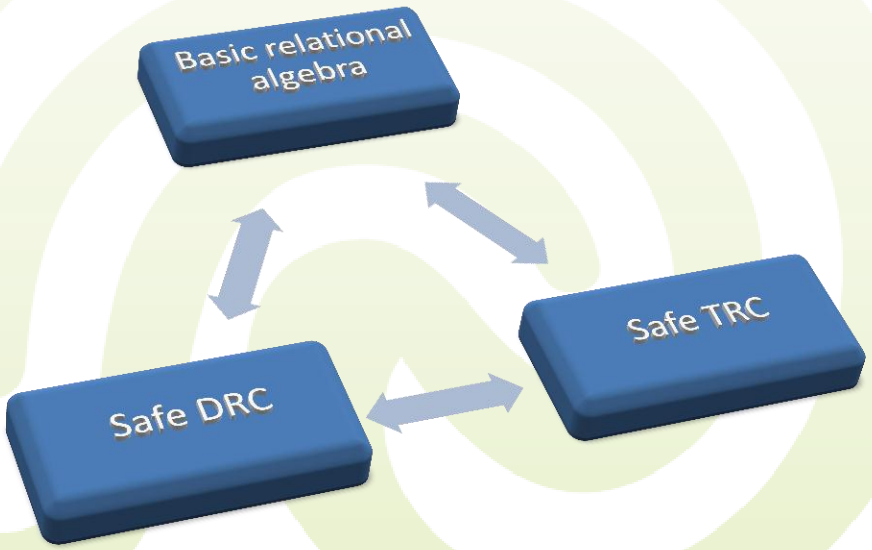
- The current state of QBE
  - Popular to query object relational databases
  - Not very widespread anywhere else ...
  - When used to query a relational database, QBE usually is implemented on top of SQL (wrapper)

7.6 Relational Completeness

- Up to now,
  we have studied **three query paradigms**
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- However, these paradigms have the **same expressiveness**
  - Any query can be written in either one of them and can easily be transformed
  - Every query language that can be mapped to one of those three is called **relational complete**

7.6 Relational Completeness

- Which parts are relational complete?
  - **Basic** relational algebra
    - Just five basic operations
    - Selection σ, Projection π, Renaming ρ, Union ∪, Set Difference \, Cartesian Product ×
  - **Safe** TRC queries
  - **Safe** DRC queries



7.6 Relational Completeness

- Also, **extended basic relational algebra** is relationally complete
  - Intersection $\cap$, theta join $\bowtie_{(\theta\text{-cond})}$, equi-join $\bowtie_{(=\text{-cond})}$, natural join $\bowtie$, left semi-join $\ltimes$, right semi-join $\rtimes$, division $\div$
  - New operations can be **composed** of the basic operations
  - New operations are just for **convenience**
- **Advanced relational algebra** is more expressive
  - Left outer join ⟕, right outer join ⟖, full outer join ⟗, aggregation $\mathfrak{F}$
  - These operations **cannot be expressed** with either DRC, TRC, nor with basic relational algebra

Next Lecture

- SQL
  - Queries
    - SELECT
  - Data manipulation language (next lecture)
    - INSERT
    - UPDATE
    - DELETE
  - Data definition language (next lecture)
    - CREATE TABLE
    - ALTER TABLE
    - DROP TABLE