Institut für Informationssysteme
Technische Universität Braunschweig

Relational Database Systems 1

Wolf-Tilo Balke
Niklas Kiehne, Enrique Pinto Dominguez
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de

7.0 Introduction

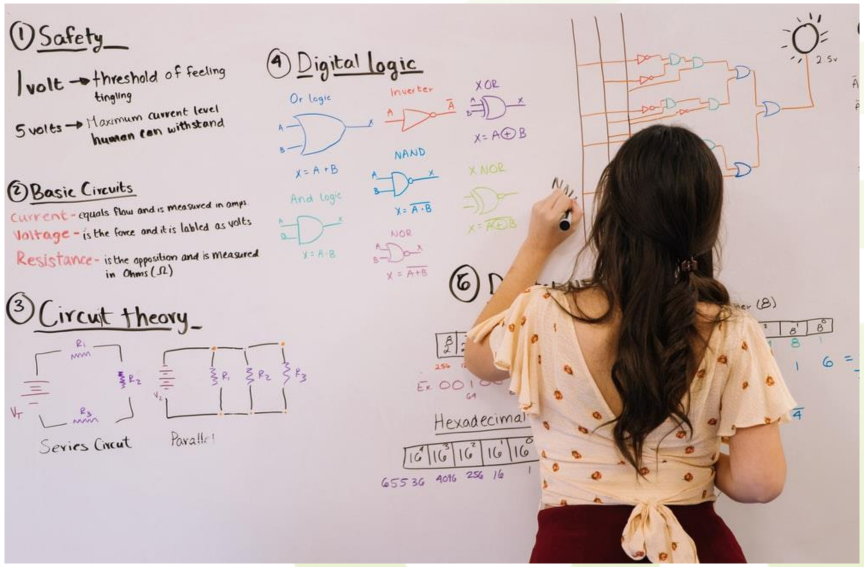
- Besides **relational algebra,** there are two other major **query** paradigms within the relational model
  - **Tuple relational calculus** (TRC)
  - **Domain relational calculus** (DRC)
- All three provide the **theoretical** foundation of the relational database model
- They are mandatory for certain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

7.0 Introduction

- Relational algebra has some **procedural** aspects
  - you specify an **order of operations** describing how to retrieve data
  - Algebra: “the mathematics of operations”
- Relational calculi (TRC, DRC) are **declarative**
  - based on first-order logics (FOL)
  - 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
  - Calculus: “the mathematics of change”

7.0 Introduction

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



① Safety
1 volt → threshold of feeling tingling
5 volts → Maximum current level human can withstand
② Basic Circuits
Current - equals flow and is measured in amps
Voltage - is the force and it is labled as volts
Resistance - is the opposition and is measured in Ohms (Ω)
③ Circuit theory
Series Circuit
Parallel
④ Digital logic
Or logic
X = A + B
And logic
X = A · B
Inverter
NAND
NOR
XOR
X = A ⊕ B
X NOR
Hexadecimal
65536
4096
256
16

7 Relational Calculus

- **Tuple relational calculus**
  - SQUARE, SEQUEL
- Domain relational calculus
  - Query-by-example (QBE)
- Relational Completeness

7.1 Tuple Relational Calculus

- **Query**
  - *Find all students having an exam result better than 2.7.*
- **TRC**
  - describe the **properties** of the desired **tuples**
  - *Get all students **s**,*
    *for which an exam report **r** exists,*
    *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.*

7.1 Tuple Relational Calculus

- Tuple Relational Calculus is an applied and extended **Monadic Predicate Calculus**
  - Fragment of **first-order** predicate logic using **no functions**, and only **monadic predicates** (i.e., with just a single argument)
    - Pure monadic predicate calculi are decidable because of their lack of expressiveness
    - In TRC, monadic predicates will have an interpretation corresponding to database relations
  - Additionally, TRC adds a small set of dyadic predicates
    - Covers the common binary comparison operator

7.1 Tuple Relational Calculus

- **Queries in TRC**
  - { *t* | CONDITION(*t*) }
  - *t* is a **tuple variable**
    - *t* usually **ranges over** all potential tuples of a relation
    - *t* may take the value of **any possible 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) holds.

7.1 Tuple Relational Calculus

- **Example:** *Select all female students.*

$\{\ t \mid \text{Student}(t) \wedge t.\text{sex} = \text{‘f’}\ \}$

Range = Student relation

Condition for result tuples

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

This type of expression resembles relational algebra’s selection!

7.1 Tuple Relational Calculus

- It is possible to retrieve only a subset of attributes
  - the **result attributes**
- **Example:** *Select the names of all female students.*

{ *t*.firstname, t.lastname | Student(*t*) ∧ *t*.sex = ‘f’ }

Result attributes

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

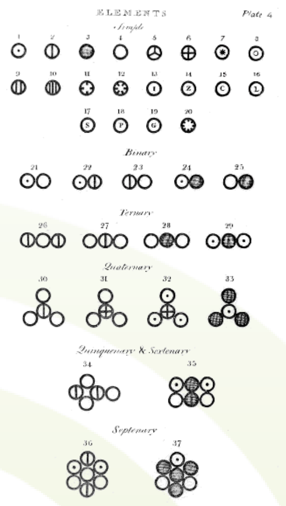
This type of expression resembles relational algebra’s projection!

7.1 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 precisely: *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 $CONDITION(t_1, t_2, \ldots, t_n)$

7.1 Tuple Relational Calculus

- What is a **formula?**
  - a **formula** is a logical expression made up of **atoms**
- Atom types
  - **range atom** $R(t)$
    - true if a tuple is an element of the relation R
      - Binds R to the tuple variable t as **range relation**
    - e.g., 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, $\theta \in \{=, <, \leq, \geq, >, \neq\}$
    - e.g., t_1.id = t_2.id



7.1 Tuple Relational Calculus

– **constant comparison atom**
  $(t.A \ \ \theta \ \ c)$ or $(c \ \ \theta \ \ t.A)$
  - a simple condition comparing an attribute *t.A* value with some constant *c*
  - *t* is a tuple variable, *A* is an attribute, *c* is a constant
  - θ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$
  - e.g., t_1.name = ‘Peter Parker’

7.1 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

7.1 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
     - $(F_1 \lor F_2)$: true iff F_1 or F_2 are true
     - $(F_1 \rightarrow F_2)$: true iff F_1 is false or F_2 is true
     - $\neg F_1$: true iff F_1 is false

Rules 3 and 4 later …

7.1 Tuple Relational Calculus

- **Evaluating formulas**
  - The theory of evaluating formulas is rather complex (see KBS lecture or a course on logics), so we keep it simple…
- TRC relies on the **open world** assumption
  - i.e., 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! Even those which do not exist!**
  - put all those tuple combinations for which *F* is true into the **result set**

7.1 Tuple Relational Calculus

- Example: { *t* | Student(*t*) ∧ t.first_name = 'Johnny' }
  - substitute *t*, one after another, with **all** possible tuples
    - <>, <1>, <2>, … , **<1005, Ronald, Reggae, m>**, …, <Hurz!, Blub, 42, Balke, Beepsheep>, …
    - 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 Student relation
    - only those tuples (in Student) whose first_name value is *Johnny* will be returned
    - Therefore: **Your statement must have a range atom for every tuple variable mentioned in the query!**

7.1 Tuple Relational Calculus

- **Example**: *All male students with matriculation number greater than 6000.*
  - $\{\ t \mid \text{Student}(t) \land t.\text{mat\_no} > 6000 \land t.\text{sex} = \text{‘m’}\ \}$
  - evaluate formula for every tuple in students

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

true ∧ false ∧ true = false

true ∧ false ∧ false = false

Result tuples

true ∧ true ∧ true = true

true ∧ true ∧ false = false

7.1 TRC: Examples

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

Course

crs_no	title
100	Advanced Politics
101	RDB for Dummies
102	Modern Boardgames

exam

student	course	result
9876	100	1.0
2832	102	3.0
1005	101	2.7
1005	100	1.7
6676	102	1.3
5119	101	3.3

7.1 TRC: Examples

- Selection

Select all female students.

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

{ t | Student(t) ∧ t.sex=‘f’ }

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

mat_no	firstname	lastname	sex
2832	Anne	Fedorova	f
8024	Elizabeth	Hargrape	f

7.1 TRC: Examples

Retrieve first name and last name of all female students.

$\pi_{\text{firstname, lastname}}\ \sigma_{\text{sex='f'}}\ \text{Student}$

{t.firstname, t.lastname | Student(t) ∧ t.sex='f'}

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

{ t | Student(t) ∧ t.sex='f' }

mat_no	firstname	lastname	sex
2832	Anne	Fedorova	f
8024	Elizabeth	Hargrape	f

{ t.firstname, t.lastname | Student(t) ∧ t.sex='f' }

firstname	lastname
Anne	Fedorova
Elizabeth	Hargrape

7.1 TRC: Examples

Compute the union of all courses with the numbers 100 and 102.

$\sigma_{crs\_no= 100}$ Course $\cup$ $\sigma_{crs\_no = 102}$ Course

$\{ t \mid Course(t) \land (t.crs\_no = 100 \lor t.crs\_no = 102) \}$

$\sigma_{crs\_no=100}$ Course

crs_no	title
100	Advanced Politics

$\sigma_{crs\_no=102}$ Course

crs_no	title
102	Modern Boardgames

$\sigma_{crs\_no=100}$ Course$\cup$ $\sigma_{crs\_no=102}$ Course

crs_no	title
100	Advanced Politics
102	Modern Boardgames

Get all courses with a number greater than 100, excluding those with a number of 102.

$\sigma_{crs\_no> 100}$ Course \ $\sigma_{crs\_no= 102}$ Course

$\{ t \mid Course(t) \land (t.crs\_no > 100 \land \neg t.crs\_no = 102) \}$

7.1 TRC: Examples

Compute the cross product of students and exams.

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

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

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

Compute a join of students and exams.

$\text{Student} \bowtie_{\text{mat\_no}=\text{student}} \text{exam}$

$\{ t_1, t_2 \mid \text{Student}(t_1) \wedge \text{exam}(t_2) \wedge t_1.\text{mat\_no} = t_2.\text{student} \}$

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m

exam

student	course	result
9876	100	1.0
2832	102	3.0
1005	101	2.7
1005	100	1.7

7.1 TRC: Another Example

- Do we need anything more?
- *Find all students' first names having an exam result better than 2.7.*

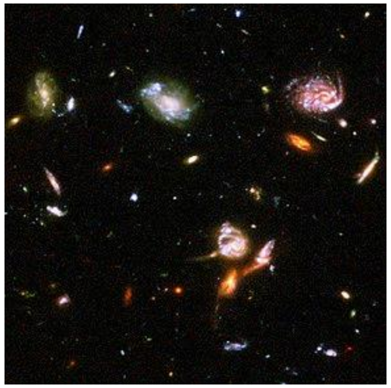
Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

exam

student	course	result
9876	100	1.0
2832	102	3.0
1005	101	2.7
1005	100	1.7
6676	102	1.3
5119	101	3.3

7.1 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.*

7.1 Tuple Relational Calculus

- With respect to quantifiers, **tuple variables** can be either **free** or **bound**
  - if F is an **atom** (and thus also a formula), each tuple variable occurring in F is **free** within F
    - example
      - $F = (t_1.\text{crs\_no} = t_2.\text{crs\_no})$
      - 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'

7.1 Tuple Relational Calculus

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

7.1 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
    - $\boldsymbol{F_1(t_1, t_2)}$ is open and has t_1 and t_2 as free variables
    - $\boldsymbol{F_2()}$ is closed and has no free variables

7.1 Tuple Relational Calculus

- Examples
  - $F_1(t_1) = (t_1.\text{name} = \text{'Ronald Reggae'})$
    - t_1 is free, F_1 is open
  - $F_2(t_1, t_2) = (t_1.\text{mat\_no} = t_2.\text{mat\_no})$
    - t_1 and t_2 are free, F_2 is open
  - $F_3(t_1) = \exists t_2(F_2(t_1,t_2)) = \exists t_2(t_1.\text{mat\_no} = t_2.\text{mat\_no})$
    - 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

7.1 Tuple Relational Calculus

- Examples
  - $F_1(t_1) = (t_1.\text{name} = \text{‘Ronald Reggae’})$
  - $F_3(t_1) = \exists t_2(F_2(t_1, t_2)) = \exists t_2(t_1.\text{mat\_no} = t_2.\text{mat\_no})$
  - $F_5(t_1) = F_1(t_1) \land F_3(t_1)$

    $= \big(t_1.\text{name} = \text{‘Ronald Reggae’} \land \exists t_2(t_1.\text{mat\_no} = t_2.\text{mat\_no})\big)$
    - t_1 is free, t_2 is bound, F_5 is open

7.1 Tuple Relational Calculus

- Examples
  - $F_1(t_1) = (t_1.\text{name} = \text{‘Ronald Reggae’})$
  - $F_4() = \exists t_1(t_1.\text{sex} = \text{‘female’})$
  - $F_6(t_1) = F_1(t_1) \land F_4()$
    $= \big(t_1.\text{name} = \text{‘Ronald Reggae’} \land \exists t_1(t_1.\text{sex} = \text{‘female’})\big)$
    - t_1 is free, t_1 is also bound, F_6 is open
- In F_6, t_1 is **bound and free** 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!) be **renamed** without affecting F_1

7.1 Tuple Relational Calculus

- Convention:
  Avoid conflicting variable names!
  - **rename** all conflicting **bound** tuple variables when they are combined with another formula
- **Examples**
  - $F_1(t_1) = (t_1.\text{name} = \text{‘Erik Reuss’})$
  - $F_4() = \exists t_1(t_1.\text{sex} = \text{‘female’}) \equiv \exists \boldsymbol{t_2}(\boldsymbol{t_2}.\text{sex} = \text{‘female’})$
  - $F_7(t_1) = F_1(t_1) \wedge F_4()$
    $\equiv \big(t_1.\text{name} = \text{‘Erik Reuss’} \wedge \exists t_2(t_2.\text{sex} = \text{‘female’})\big)$
    - t_1 is free, t_2 is bound, F_7 is open

7.1 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 a such that $F(a)$ 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

7.1 TRC: Examples

List the names of all students that took some exam.

$$\pi_{\text{firstname}}\ (\text{Student} \ltimes_{\text{mat\_no=student}} \text{exam})$$

$$\{\ t_1.\text{firstname} \mid \text{Student}(t_1) \land \exists t_2(\text{exam}(t_2) \land t_1.\text{mat\_no} = t_2.\text{student})\ \}$$

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m

exam

student	course	result
9876	100	1.0
2832	102	3.0
1005	101	2.7
1005	100	1.7

7.1 TRC: Examples

"List students having at least those exams Ronald Reggae had"

$SC \div (\pi_{crsNr}\ \sigma_{name = \text{'Ronald Reggae'}}\ SC)$

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

$F_1(t_1) = \forall t_2\ ((SC(t_2) \land t_2.name = \text{'Ronald Reggae'}) \rightarrow F_2(t_1))$

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

SC (= students and courses)

matNr	name	crsNr
1000	Ronald Reggae	C100
1000	Ronald Reggae	C102
1001	Anne Fedorova	C100
1002	Michael Gorby	C102
1002	Michael Gorby	C100
1002	Michael Gorby	C101
1003	Johnny Tolkien	C103
1003	Johnny Tolkien	C100

For all tuples of Ronald Reggae, 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 Ronald Reggae in F_2

Result

matNr	name
1000	Ronald Reggae
1002	Michael Gorby

7.1 TRC: Examples

- **Social Question-Answering**
  - **User** (<u>uid</u>, name)
  - **Question** (<u>qid</u>, author → User, title, text)
  - **Answer** (<u>aid</u>, author → User, question → Question, text)
- Query: Find the ID and title of all questions which have exactly one answer.
  - $\{q.qid, q.title \mid Question(q) \land \exists a_1(Answer(a_1) \land a_1.question = q.qid \land \neg\exists\, a_2(Answer(a_2) \land a_2.question = q.qid \land a_2.aid \neq a_1.aid))\}$

7.1 Tuple Relational Calculus

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

    - $\forall t\ (F_1(t) \wedge F_2(t)) \equiv \neg\exists t\ (\neg F_1(t) \vee \neg F_2(t))$
    - $\forall t\ (F_1(t) \vee F_2(t)) \equiv \neg\exists t\ (\neg F_1(t) \wedge \neg F_2(t))$

    - $\exists t\ (F_1(t) \wedge F_2(t)) \equiv \neg\forall t\ (\neg F_1(t) \vee \neg F_2(t))$
    - $\exists t\ (F_1(t) \vee F_2(t)) \equiv \neg\forall t\ (\neg F_1(t) \wedge \neg F_2(t))$

7.1 Tuple Relational Calculus

- More considerations on **evaluating** TRC:
  What happens to **quantifiers** and **negation**?
  - again: **open world**!

mat_no	name	sex
1776	Ann Hasaway	f
8024	Ina Shepherd	f

- Consider relation Students
  - **$\exists t$ (t.sex = 'm')** ≡ true
    - *t* can represent any tuple, and there can be a tuple for that the condition holds, e.g. <7312, Winnie Phew, m> or <-1, &cjndks, m>
  - **$\exists t$ (Student(t) $\wedge$ t.sex = 'm')** ≡ false
    - there is no male tuple in the Student relation
  - **$\forall t$ (t.sex = 'f')** ≡ false
  - **$\forall t$ ($\neg$Student(t) $\vee$ t.sex = 'f')** ≡ true
    - all tuples are either female or they are not in Student
    - *All tuples in the relation are girls.*

7.1 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 return an infinite number of tuples are called **unsafe**
- **Unsafe** queries should be **avoided** and cannot be evaluated (reasonably)!
  - one reliable way of avoiding unsafe expressions is the **closed world assumption**

7.1 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

7.1 Tuple Relational Calculus

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

Student

mat_no	Name	sex
1776	Ann Hasaway	f
8024	Ina Shepherd	f

Expression	Open World	Closed World
$\exists t$ (t.sex = 'm')	true	false
$\exists t$ (Student(t) $\land$ t.sex = 'm')	false	false
$\forall t$ (t.sex = 'f')	false	true
$\forall t$ ($\neg$Student(t) $\lor$ t.sex = 'f')	true	true

7.1 Tuple Relational Calculus

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

7.1 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.*

7.1 SQUARE & SEQUEL

Detour

- *Current paradigms* at that 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

7.1 SQUARE & SEQUEL

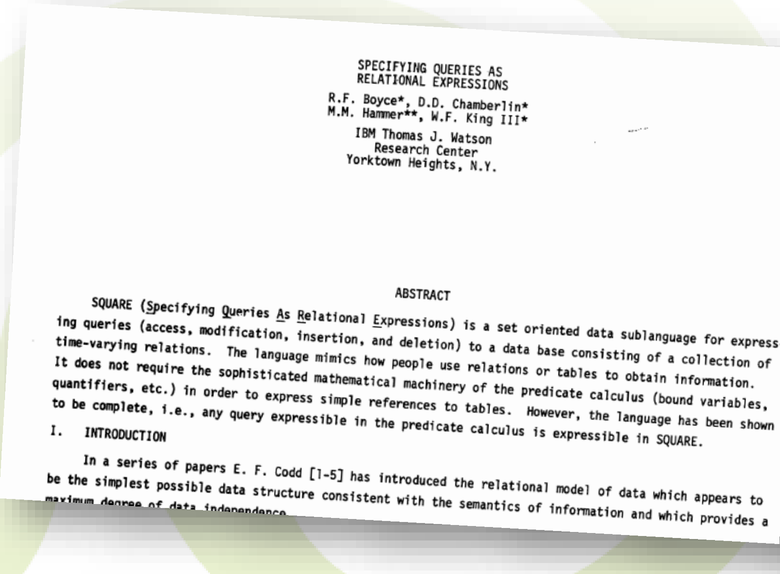
Detour

- 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**

7.1 SQUARE & SEQUEL

Detour

- 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.

7.1 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)

7.1 SQUARE & SEQUEL

Detour

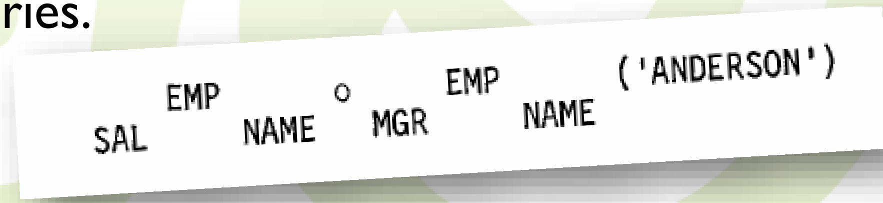
- Get a list of all exam results better than 2.0 along with the according student name
  - TRC:

    $\{\ t_1.\text{name}, t_2.\text{result} \mid \text{Student}(t_1) \land \text{exam}(t_2)$
    $\qquad \land\ t_1.\text{mat\_nr} = t_2.\text{student} \land t_2.\text{result} < 2.0\ \}$
  - SQUARE:

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

Join of two SQUARE queries

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


$${}_{\text{SAL}}\text{EMP}_{\text{NAME}} \circ {}_{\text{MGR}}\text{EMP}_{\text{NAME}}\ (\text{'ANDERSON'})$$

7.1 SQUARE & SEQUEL

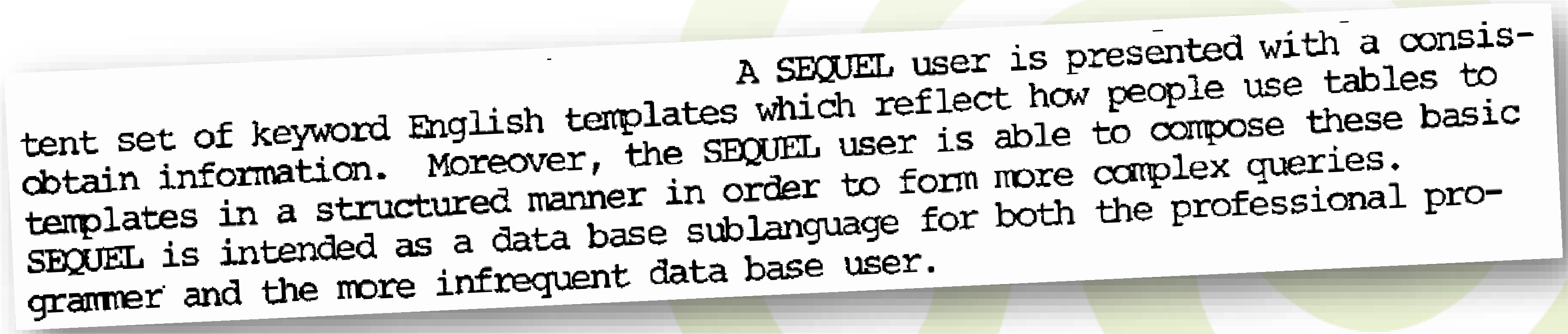
Detour

- 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 mat_nr ∘ student exams crs_nr result (102, <2.0)`
  - SQUARE's syntax is too mathematical and artificial

7.1 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

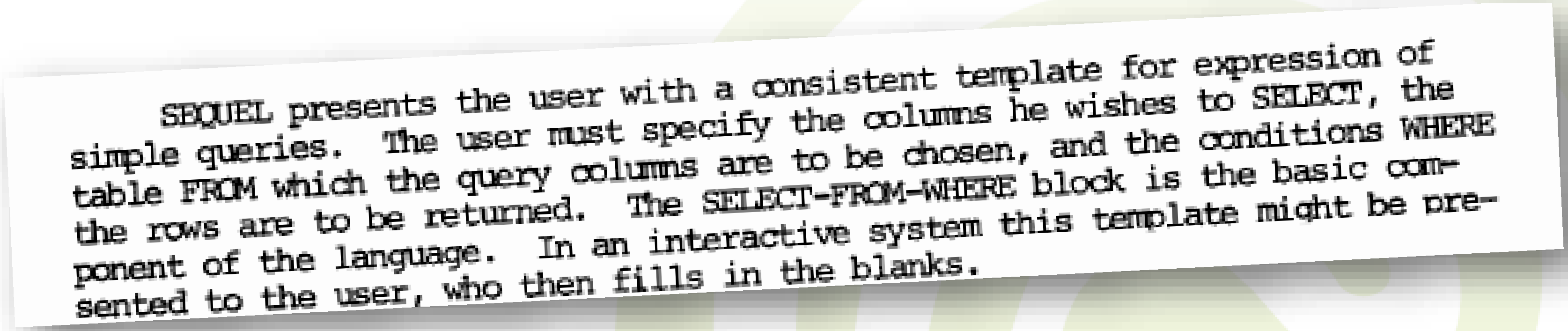


A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage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

7.1 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.

7.1 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

7.1 SQUARE & SEQUEL

Detour

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

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

  - SEQUEL:

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

7.1 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

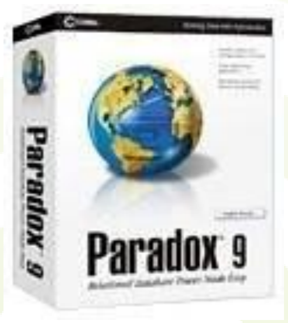
7.1 SQUARE & SEQUEL

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 Relational Calculus

- Tuple relational calculus
  - SQUARE, SEQUEL
- **Domain relational calculus**
  - Query-by-example (QBE)
- Relational Completeness



7.2 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

7.2 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 and y are domain variables
    - θ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$
  - **constant comparison atoms:** $(x\ \theta\ c)$ or $(c\ \theta\ x)$
    - x is a domain variable, c is a constant value
    - θ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$

7.2 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
- Also other aspects of DRC are similar to TRC

7.2 DRC: Examples

Retrieve first name and last name of all female students.

Relational algebra: $\pi_{\text{firstname, lastname}}\ \sigma_{\text{sex = 'f'}}$ Student

TRC: { *t*.firstname, *t*.lastname | Student(*t*) ∧ *t*.sex = ‘f’ }

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

Student

mat_no	firstname	lastname	sex
1005	Ronald	Reggae	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m
6676	Erik	Reuss	m
8024	Elizabeth	Hargrape	f
9876	Plato	NULL	m

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

mat_no	firstname	lastname	sex
2832	Anne	Fedorova	f
8024	Elizabeth	Hargrape	f

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

firstname	lastname
Anne	Fedorova
Elizabeth	Hargrape

7.2 DRC: Examples

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

Relational algebra: $\pi_{\text{firstname}}$ (Student $\ltimes_{\text{mat\_no=student}}$ exam)

TRC: $\{\ t_1.\text{firstname} \mid \text{Student}(t_1) \land \exists t_2(\text{exam}(t_2) \land t_2.\text{student} = t_1.\text{mat\_no})\ \}$

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

mat_no	firstname	lastname	sex
1005	Ronald	Reagan	m
2832	Anne	Fedorova	f
4512	Michael	Gorbachef	m
5119	Johnny	Tolkien	m

exam

student	course	result
9876	100	1.0
2832	102	3.0
1005	101	2.7
1005	100	1.7

7.2 DRC: Examples

- Reconsider last lecture: Algebra Division !!!
  - $R \div S$
    - Read relational algebra division as a "*forall*" statement
  - Given relation R and S:
    - $R(a_1, \ldots, a_n, b_1, \ldots, b_m)$
    - $S(b_1, \ldots, b_m)$
  - $R \div S \ = \{ a_1, \ldots, a_n \mid \forall b_1, \ldots, b_m\ (\neg S(b_1, \ldots, b_m) \vee R(a_1, \ldots, a_n, b_1, \ldots, b_m))\}$

$$= \{ a_1, \ldots, a_n \mid \forall b_1, \ldots, b_m\ (S(b_1, \ldots, b_m) \rightarrow R(a_1, \ldots, a_n, b_1, \ldots, b_m))\}$$

7.2 DRC: Examples

Division:

$SR = \rho_{SR}(\pi_{\text{matnr, lastname, crsnr}}(\text{Student} \bowtie_{\text{matnr=student}} \text{exam}))$

matnr	lastname	crsnr
1000	Reggae	100
1000	Reggae	102
1001	Fedorova	100
1002	Gorby	102
1002	Gorby	100
1002	Gorby	101
1003	Tolkien	103
1003	Tolkien	100

$CRR = \rho_{CRR}(\pi_{crsnr}\sigma_{\text{lastname='Reggae'}} SR)$

crsnr
100
102

$SC \div CCK$

matnr	lastname
1000	Reggae
1002	Gorby

$SR \div CRR = \{ \text{matnr, lastname} \mid \forall \text{crsnr} ($
$\quad CRR(\text{crsnr}) \rightarrow SC(\text{matnr, lastname, crsnr})$
$)\}$

Result contains all those students who took at least the same courses as (Ronald) Reggae.

7.2 Domain Relational Calculus

- There are **formally incorrect shorthands** which can be used
  - When dealing with equality, you can shorten your expressions a bit:

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

  - Which is also useful for joins:

$$\{fn \mid \exists \mathbf{mat}, ln, r\ (\mathrm{Student}(mat, fn, ln, r) \land \exists \mathbf{st}, co, r\ (\mathrm{exam}(st, co, r) \land \mathbf{st = mat}))\}$$
$$= \{fn \mid \exists \mathbf{mat}, ln, r\ (\mathrm{Student}(\mathbf{mat}, fn, ln, r) \land \exists co, r\ (\mathrm{exam}(\mathbf{mat}, co, r)))\}$$

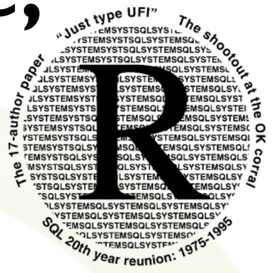
7.2 Domain Relational Calculus

- An even worse shorthand sometimes encountered is the use implicit existential quantifiers
  - You may NOT do this in exercises /exams!
    - {*fn*, *ln* | Student(***mat***, *fn*, *ln*, ’f’)}
    - {*fn* | Student(***mat***, *fn*, *ln*, *s*) ∧ ∃co, r(exam(***mat***, *co*, *r*))}

7.2 QBE

Detour

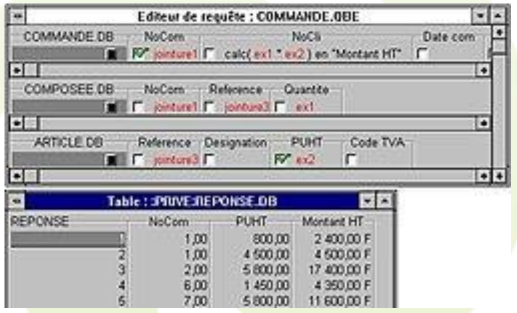
- The first version of SQL, **SEQUEL**, was developed in early 1970 by D. **Chamberlin** and R. **Boyce** at **IBM** Research in **San Jose,** California
  - based on the **tuple relational calculus**
- At the same time, another query language, **QBE,** was developed independently by M. **Zloof** at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


"Just type UFI"
The shootout at the OK corral
The 17-author paper
SQL 20th year reunion: 1975-1995
R

7.2 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



Editeur de requête : COMMANDE.QBE

COMMANDE.DB	NoCom	NoCli	Date com
	✓ jointure1	calc(ex1 * ex2) en "Montant HT"	

COMPOSEE.DB	NoCom	Reference	Quantite
	jointure1	jointure3	ex1

ARTICLE.DB	Reference	Designation	PUHT	Code TVA
	jointure3		✓ ex2	

Table : :PRIVE:REPONSE.DB

REPONSE	NoCom	PUHT	Montant HT
1	1,00	800,00	2 400,00 F
2	1,00	4 500,00	4 500,00 F
3	2,00	5 800,00	17 400,00 F
4	6,00	1 450,00	4 350,00 F
5	7,00	5 800,00	11 600,00 F

7.2 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

7.2 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

7.2 QBE

Detour

$\pi_{mat\_no} \sigma_{lastname = 'Reggae'} \text{Student}$

$\{ mat \mid \exists\, fn, s\ (Student(mat, fn, 'Reggae', s)) \}$

Student	mat_no	firstname	lastname	sex
	P.		Reggae	

← QBE

$\{ st \mid \exists co, r\ (exam(st, co, r) \wedge r > 2.0) \}$

exam	student	course	result
	P.		> 2.0

$\{ st \mid \exists\ co, r\ (exam(st, co, r) \wedge co \neq 102 \}$

exam	student	course	result
	P.	≠ 102	

7.2 QBE

Detour

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

_st is the *example* in *query-by-example*!

exam	student	course	result
	P._st	100	
	_st	102	

- get the mat_no of students who took exams in course 100 **or** 102

exam	student	course	result
	P.	100	
	P.	102	

7.2 QBE

Detour

- *Get the matriculation number of those students who took at least one course that also the student 1005 took.*

exam	student	course	result
	P.	_co	
	1005	_co	

– 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._st		_res	P.AVG._res

7.2 QBE

Detour

- This can of course also be applied between tables
  - analogous to joins in relational algebra
  - e.g. *What are the last names of all female students who got a very good grade in some exam?*

Student	mat_no	firstname	lastname	sex
	_st		P.	f

exam	student	course	result
	_st		< 1.3

7.2 QBE

Detour

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

Student	mat_no	firstname	lastname	sex
I.	1005	Ronald	Reggae	m

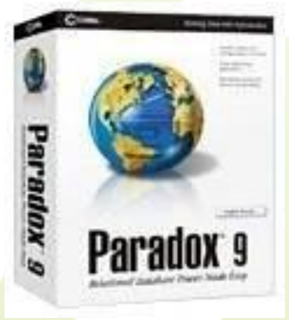
  - or from other tables by connecting them with domain variables
  - insert (I.), delete (D.), or update (U.)
    - update directly in columns

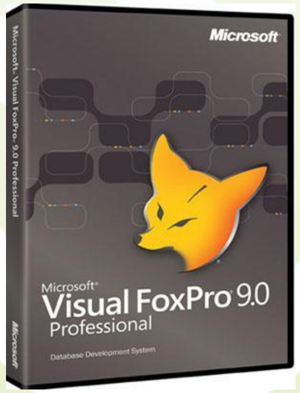
exam	student	course	result
	2832	102	U._res +1.0

7.2 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

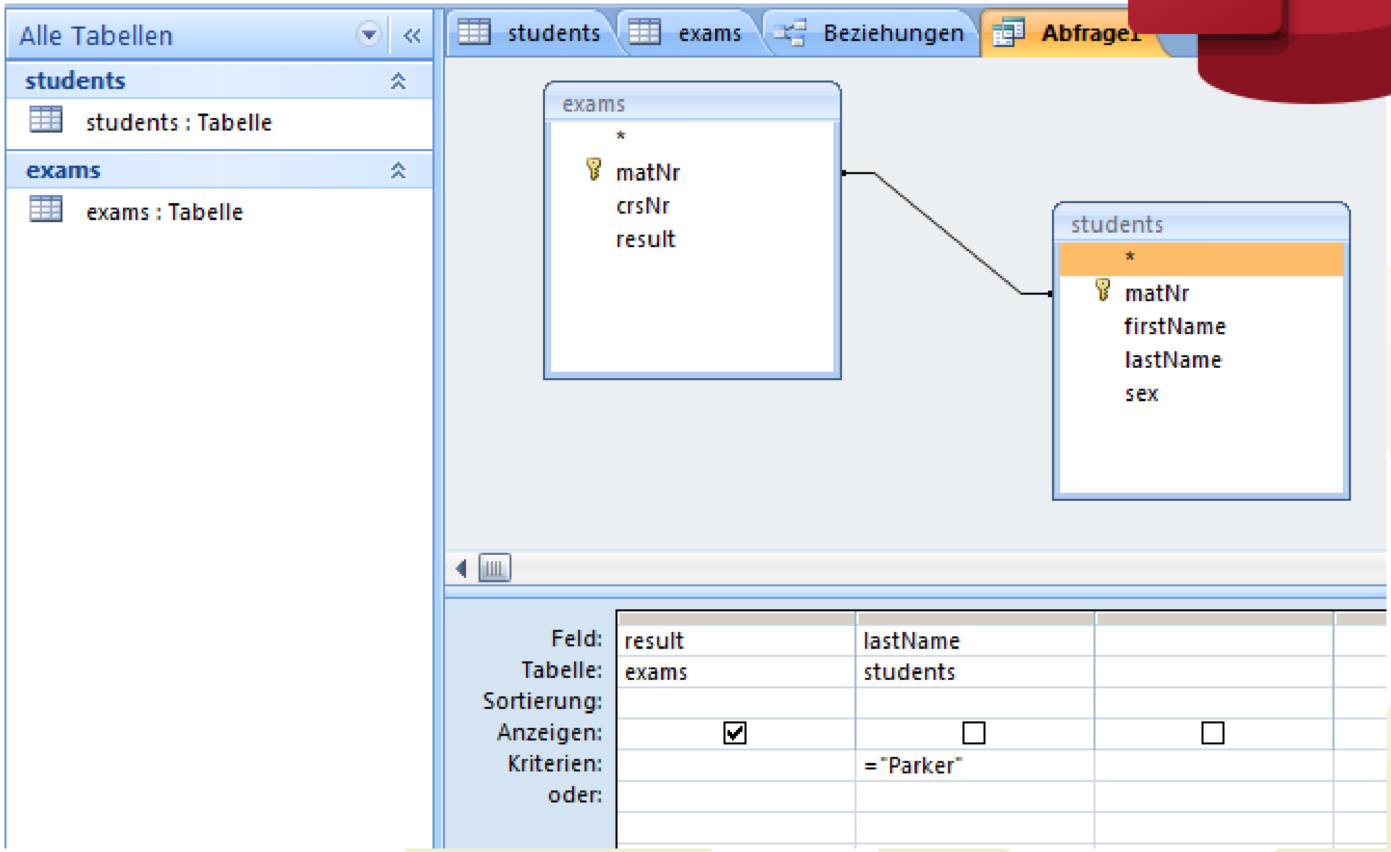



Microsoft
Microsoft Visual FoxPro 9.0 Professional
Database Development System

7.2 QBE

Detour

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



Feld:	result	lastName	
Tabelle:	exams	students	
Sortierung:			
Anzeigen:	☑	☐	☐
Kriterien:		="Parker"	
oder:			

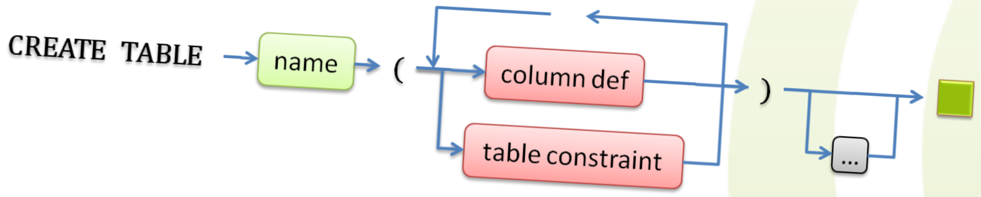
7.2 QBE

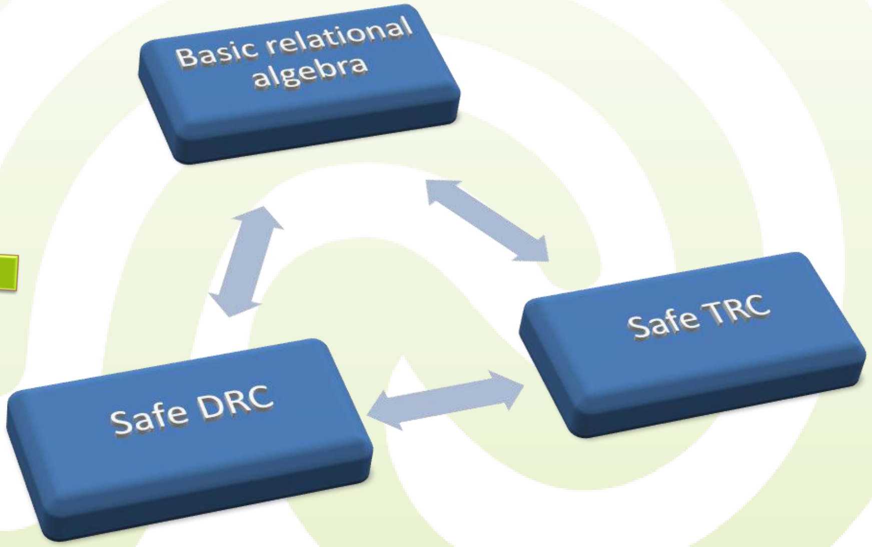
Detour

- 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 Relational Calculus

- Tuple relational calculus
  - SQUARE, SEQUEL
- Domain relational calculus
  - Query-by-example (QBE)
- **Relational Completeness**



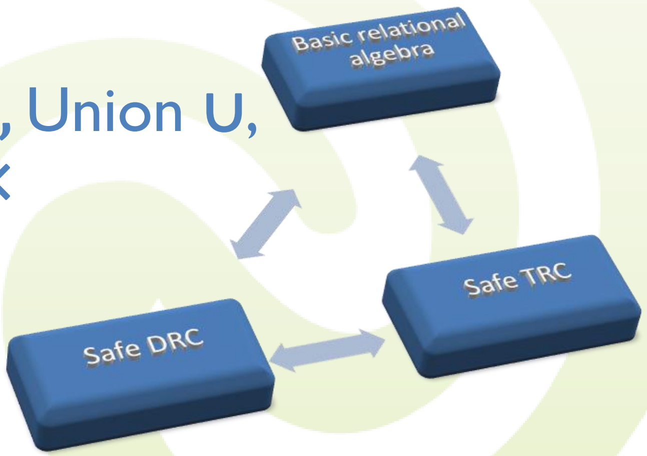


7.3 Relational Completeness

- Up to now, we have studied **three query paradigms**
  - Extended 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 **relationally complete**
    - E.F. Codd, "Relational Completeness of Data Base Sublanguages", 1971

7.3 Relational Completeness

- A language is relationally complete, giving a set of relations, if it permits creating the same relations as safe TRC
  - i.e. if it allows at least creating equivalent queries
- Which parts are relationally 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



7.3 Relational Completeness

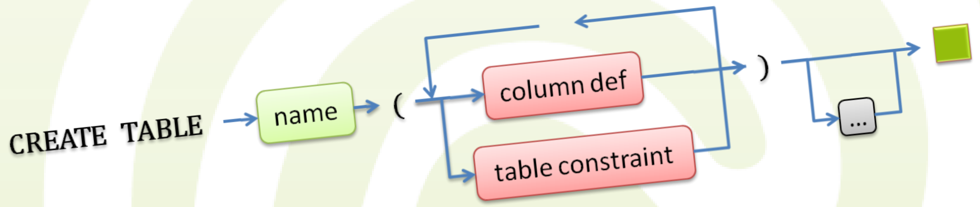
- Also, **extended basic relational algebra** is relationally complete
  - Intersection ∩, Theta Join $\bowtie_{(\theta\text{-cond})}$, Equi-Join $\bowtie_{(=\text{-cond})}$, Natural Join ⋈ , Left Semi-Join ⋉, Right Semi-Join ⋊, Division ÷
  - new operations can be **composed** of the basic operations
  - new operations are just for **convenience**
- **Advanced relational algebra** is also relationally complete but additionally **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

7.3 Relational Completeness

- Relational Completeness does not describe that all useful queries that one would want!
- Missing from relational completeness:
  - **Aggregation and statistics**
    - Advanced Relational Algebra and SQL can do this
  - **Sorting** and **result orders** (i.e. transition from set to lists)
    - SQL can do this (however, beside sorting there are no guarantees)
  - 3-Valued Logic with **NULL values**
    - SQL and Advanced Relational Algebra can do this
  - **Transitive Closures** (i.e. recursive joins)
    - Vanilla SQL cannot do this, but there is an extension called Recursive SQL
  - **Multisets** (i.e. duplicate tuples)
    - SQL can do this

7 Next Lecture

- SQL
  - queries
    - SELECT
  - Data Manipulation Language (in two lecture)
    - INSERT
    - UPDATE
    - DELETE
  - Data Definition Language (in two lecture)
    - CREATE TABLE
    - ALTER TABLE
    - DROP TABLE


CREATE TABLE
name
(
column def
table constraint
)
...