7.0 Summary

• Basic relational algebra
  – Selection $\sigma$
  – Projection $\pi$
  – Renaming $\rho$
  – Union $\cup$, intersection $\cap$, and set difference $\setminus$
  – Cartesian product $\times$

• Extended relational algebra
  – Theta-join $\bowtie_{(\theta\text{-cond})}$, Equi-join $\bowtie_{(=\text{-cond})}$, Natural join $\bowtie$
  – Left semi-join $\bowtie$ and right semi-join $\bowtie$
  – Division $\div$

• Advanced relational algebra
  – Left outer join $\bowtie$, right outer join $\bowtie$, full outer join $\bowtie$
  – Aggregation $\sum$
7.0 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
• 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**
  – 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”
Both calculi are special cases of the first-order predicate calculus

- **TRC** = logical expressions on **tuples**
- **DRC** = logical expressions on **attribute domains**
7 Relational Calculus

• Tuple relational calculus
  – SQUARE, SEQUEL

• Domain relational calculus
  – Query-by-example (QBE)

• Relational Completeness
7.1 Tuple Relational Calculus

• **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

• Queries in TRC

− \{ t \mid \text{CONDITION}(t) \}

− \( t \) is a **tuple variable**
  • \( t \) usually **ranges over** all tuples of a relation
  • \( t \) may take the value of **any tuple**

− \text{CONDITION}(t) is a **logical statement** involving \( t \)
  • all those tuples \( t \) are retrieved that satisfy \text{CONDITION}(t)

− reads as:
  \textit{Retrieve all tuples \( t \) for that \text{CONDITION}(t) is true.}
7.1 Tuple Relational Calculus

**Example:** Select all female students.

\[
\{ t | \text{Student}(t) \land t.\text{sex} = 'f' \}
\]

Range = Student relation

Condition for result tuples

This type of expression resembles relational algebra’s selection!

<table>
<thead>
<tr>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
<tr>
<td>2832</td>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
</tr>
<tr>
<td>4512</td>
<td>Lex</td>
<td>Luther</td>
<td>m</td>
</tr>
<tr>
<td>5119</td>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
</tr>
<tr>
<td>6676</td>
<td>Erik</td>
<td>Magnus</td>
<td>m</td>
</tr>
<tr>
<td>8024</td>
<td>Jeanne</td>
<td>Gray</td>
<td>f</td>
</tr>
<tr>
<td>9876</td>
<td>Logan</td>
<td></td>
<td>m</td>
</tr>
</tbody>
</table>
• It is possible to retrieve only a subset of attributes – the request attributes

• **Example:** Select the names of all female students.

\[
\{ \text{t.firstname, t.lastname} \mid \text{Student(t)} \land \text{t.sex} = 'f' \}
\]

**Result attributes**

<table>
<thead>
<tr>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
<tr>
<td>2832</td>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
</tr>
<tr>
<td>4512</td>
<td>Lex</td>
<td>Luther</td>
<td>m</td>
</tr>
<tr>
<td>5119</td>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
</tr>
<tr>
<td>6676</td>
<td>Erik</td>
<td>Magnus</td>
<td>m</td>
</tr>
<tr>
<td>8024</td>
<td>Jeanne</td>
<td>Gray</td>
<td>f</td>
</tr>
<tr>
<td>9876</td>
<td>Logan</td>
<td></td>
<td>m</td>
</tr>
</tbody>
</table>

This type of expression resembles relational algebra’s projection!
7.1 Tuple Relational Calculus

• Full query syntax:

\[ \{ t_1.A_1, t_2.A_2, \ldots, t_n.A_n \mid \text{CONDITION}(t_1, t_2, \ldots, t_n) \} \]

– \( t_1, t_2, \ldots, t_n \) are tuple variables
– \( A_1, A_2, \ldots, A_n \) are attributes,
  where \( A_i \) is an attribute of tuple \( t_i \)
– \( \text{CONDITION} \) specifies a condition on tuple variables
  • more precise (to be defined in detail later):
    \( \text{CONDITION} \) 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 \( \text{CONDITION}(t_1, t_2, \ldots, t_n) \)
7.1 Tuple Relational Calculus

• What is a **formula**?
  – a **formula** is a logical expression made up of **atoms**

• **Atom types**
  – **range atom** $R(t)$
    • evaluates if a tuple is an element of the relation $R$
      – Binds $R$ to the tuple variable $t_i$ as **range relation**
    • e.g. $\text{Student}(t)$
  – **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
    • $\theta$ is a comparison operator, $\theta \in \{=, <, \leq, \geq, >, \neq\}$
    • e.g. $t_1.id = t_2.id$
– 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
- \(\theta\) is a comparison operator, \(\theta \in \{=, <, \leq, \geq, >, \neq\}\)
- e.g. \(t_1\).name = ‘Peter Parker’
• 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
• **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
   - $(F_1 \rightarrow F_2)$: false iff both $F_1$ is true $F_2$ is false
   - $\neg F_1$: false iff $F_1$ is true

Rules 3 and 4 on later slides …
• **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 so-called **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 that \( F \) is true into the **result set**
7.1 Tuple Relational Calculus

• Example: \{ t | Student(t) \land \text{first\_name} = 'Clark' \}

  – substitute \( t \), one after another, with all possible tuples
    • \(<\rangle, <1>, <2>, \ldots, <1005, \text{Clark, Kent, m}>, \ldots, <\text{Hurz!, Blub, 42, Balke, Spiderman}>, \ldots\)
    • 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 \text{first\_name} value is \text{Clark} will be returned
    • Therefore: \text{Your statement should have a range atom!}
• **Example:** *All male students with matriculation number greater than 6000.*

\[
\{ t \mid \text{Student}(t) \land t.\text{mat\_no} > 6000 \land t.\text{sex} = 'm' \}
\]

– evaluate formula for every tuple in students

<table>
<thead>
<tr>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
<tr>
<td>2832</td>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
</tr>
<tr>
<td>4512</td>
<td>Lex</td>
<td>Luther</td>
<td>m</td>
</tr>
<tr>
<td>5119</td>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
</tr>
<tr>
<td>6676</td>
<td>Erik</td>
<td>Magnus</td>
<td>m</td>
</tr>
<tr>
<td>8024</td>
<td>Jeanne</td>
<td>Gray</td>
<td>f</td>
</tr>
<tr>
<td>9876</td>
<td>Logan</td>
<td></td>
<td>m</td>
</tr>
</tbody>
</table>
# 7.1 TRC: Examples

## Student

<table>
<thead>
<tr>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
<tr>
<td>2832</td>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
</tr>
<tr>
<td>4512</td>
<td>Lex</td>
<td>Luther</td>
<td>m</td>
</tr>
<tr>
<td>5119</td>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
</tr>
<tr>
<td>6676</td>
<td>Erik</td>
<td>Magnus</td>
<td>m</td>
</tr>
<tr>
<td>8024</td>
<td>Jeanne</td>
<td>Gray</td>
<td>f</td>
</tr>
<tr>
<td>9876</td>
<td>Logan</td>
<td></td>
<td>m</td>
</tr>
</tbody>
</table>

## Course

<table>
<thead>
<tr>
<th>crs_no</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Intro. to being a Superhero</td>
</tr>
<tr>
<td>101</td>
<td>Secret Identities 2</td>
</tr>
<tr>
<td>102</td>
<td>How to take over the world</td>
</tr>
</tbody>
</table>

## Exam

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>9876</td>
<td>100</td>
<td>3.7</td>
</tr>
<tr>
<td>2832</td>
<td>102</td>
<td>2.0</td>
</tr>
<tr>
<td>1005</td>
<td>101</td>
<td>4.0</td>
</tr>
<tr>
<td>1005</td>
<td>100</td>
<td>1.3</td>
</tr>
<tr>
<td>6676</td>
<td>102</td>
<td>4.3</td>
</tr>
<tr>
<td>5119</td>
<td>101</td>
<td>1.7</td>
</tr>
</tbody>
</table>
7.1 TRC: Examples

- **Selection**

*Select all female students.*

\[ \sigma_{\text{sex} = 'f'} \text{ Student} \]

\[ \{ t | \text{Student}(t) \land t.\text{sex} = 'f' \} \]
Retrieve first name and last name of all female students.

\[ \pi_{\text{firstname}, \text{lastname}} \sigma_{\text{sex}=\text{f}} \text{Student} \]

\[ \{ \text{t.firstname, t.lastname} \mid \text{Student(t)} \land \text{t.sex}=\text{f} \} \]
7.1 TRC: Examples

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

\[ \sigma_{\text{crs_no}=100} \text{Course} \cup \sigma_{\text{crs_no}=102} \text{Course} \]

\{ t | \text{Course}(t) \land (t.\text{crs_no} = 100 \lor t.\text{crs_no} = 102) \} \]

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

\[ \sigma_{\text{crs_no}>100} \text{Course} \setminus \sigma_{\text{crs_no}=102} \text{Course} \]

\{ t | \text{Course}(t) \land (t.\text{crs_no} > 100 \land \neg t.\text{crs_no} = 102) \} \]

<table>
<thead>
<tr>
<th>crs_no</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Intro. to being a Superhero</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>crs_no</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>How to take over the world</td>
</tr>
</tbody>
</table>
7.1 TRC: Examples

Compute the cross product of students and exams.

Student $\times$ exam

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

Compute a join of students and exams.

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

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

<table>
<thead>
<tr>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
<tr>
<td>2832</td>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
</tr>
<tr>
<td>4512</td>
<td>Lex</td>
<td>Luther</td>
<td>m</td>
</tr>
<tr>
<td>5119</td>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>9876</td>
<td>100</td>
<td>3.7</td>
</tr>
<tr>
<td>2832</td>
<td>102</td>
<td>2.0</td>
</tr>
<tr>
<td>1005</td>
<td>101</td>
<td>4.0</td>
</tr>
<tr>
<td>1005</td>
<td>100</td>
<td>1.3</td>
</tr>
</tbody>
</table>
• Additionally, in TRC there can be formulas considering all tuples

– **universal quantifier** \( \forall \)
  - 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** \( \exists \)
  - 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.*
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
    - \( F = (t_1.crs_no = t_2.crs_no) \)
    - 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 \( F' \)
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

- \( t \) 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 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)
• 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
7.1 Tuple Relational Calculus

• Examples

– $F_1(t_1) = (t_1\text{.name} = \text{‘Clark Kent’})$
  • $t_1$ is free, $F_1$ is open

– $F_2(t_1, t_2) = (t_1\text{.mat\_no} = t_2\text{.mat\_no})$
  • $t_1$ 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

– $F_1(t_1) = (t_1\text{.name} = \text{‘Clark Kent’})$
– $F_3(t_1) = \exists t_2(F_2(t_1, t_2)) = \exists t_2(t_1\text{.mat_no} = t_2\text{.mat_no})$
– $F_5(t_1) = F_1(t_1) \land F_3(t_1)$
  $$= (t_1\text{.name} = \text{‘Clark Kent’})$$
  $$\land \exists t_2(t_1\text{.mat_no} = t_2\text{.mat_no})$$

• $t_1$ is free, $t_2$ is bound, $F_5$ is open
• 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!) 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
  – \( F_1(t_1) = (t_1.name = \text{‘Clark Kent’}) \)
  – \( F_4() = \exists t_1(t_1.sex = \text{‘female’}) \equiv \exists t_2(t_2.sex = \text{‘female’}) \)
  – \( F_7(t_1) = F_1(t_1) \land F_4() \)
    \[ \equiv (t_1.name = \text{‘Clark Kent’} \land \exists t_2(t_2.sex = \text{‘female’})) \]
  
  • \( t_1 \) is free, \( t_2 \) is bound, \( F_7 \) is open
• 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
• Thoughts on quantifiers
  – any formula with an **existential quantifier** can be **transformed** into one with an **universal quantifier** and vice versa
  – quick rule: replace $\lor$ by $\land$ and negate everything
    - $\forall t (F(t)) \equiv \neg \exists t (\neg F(t))$
    - $\exists t (F(t)) \equiv \neg \forall t (\neg F(t))$
    - $\forall t (F_1(t) \land F_2(t)) \equiv \neg \exists t (\neg F_1(t) \lor \neg F_2(t))$
    - $\forall t (F_1(t) \lor F_2(t)) \equiv \neg \exists t (\neg F_1(t) \land \neg F_2(t))$
    - $\exists t (F_1(t) \land F_2(t)) \equiv \neg \forall t (\neg F_1(t) \lor \neg F_2(t))$
    - $\exists t (F_1(t) \lor F_2(t)) \equiv \neg \forall t (\neg F_1(t) \land \neg F_2(t))$
More considerations on **evaluating TRC**: What happens to **quantifiers** and **negation**?

- again: **open world**!

Consider relation students

- $\exists t \ (t.\text{sex} = \text{‘m’}) \equiv \text{true}$
  
  - $t$ can represent any tuple, and there can be a tuple for that the condition holds, e.g. $<7312, \text{Scott Summers, m}>$ or $<-1, \&cjndks, \text{m}>$

- $\exists t \ (\text{Student}(t) \land t.\text{sex} = \text{‘m’}) \equiv \text{false}$
  
  - there is no male tuple in the Student relation

- $\forall t \ (t.\text{sex} = \text{‘f’}) \equiv \text{false}$

- $\forall t \ (\neg \text{Student}(t) \lor t.\text{sex} = \text{‘f’}) \equiv \text{true}$
  
  - all tuples are either female or they are not in Student
  
  - *All tuples in the relation are girls.*
List the names of all students that took some exam.

\[ \pi_{\text{firstname}} \left( \text{Student} \bowtie_{\text{mat_no}=\text{student}} \text{exam} \right) \]

\{ t_1.\text{firstname} | \\
\text{Student}(t_1) \land \exists t_2(\text{exam}(t_2) \land t_1.\text{mat_no} = t_2.\text{student}) \}
• Consider the TRC query: \{ t \mid \neg \text{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
7.1 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**

<table>
<thead>
<tr>
<th>mat_no</th>
<th>Name</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1776</td>
<td>Leni Zauber</td>
<td>f</td>
</tr>
<tr>
<td>8024</td>
<td>Jeanne Gray</td>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Open World</th>
<th>Closed World</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exists t \ (t.\text{sex} = 'm')$</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>$\exists t \ (\text{Student}(t) \land t.\text{sex} = 'm')$</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>$\forall t \ (t.\text{sex} = 'f')$</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>$\forall t \ (\neg\text{Student}(t) \lor t.\text{sex} = 'f')$</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
• 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 …
• 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.
• 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
Chamberlin and Boyce’s first result was a query language called **SQUARE**

- **Specifying queries as relational expressions**
- 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
7.1 SQUARE & SEQUEL

• Retrieve the names of all female students
  – TRC: \{ t.name | Student(t) \land t.sex = 'f' \}
  – SQUARE: \text{name}\text{Student}_{sex} ('f')

• Get all exam results better than 2.0 in course 101
  – TRC: \{ t.result | exam(t) \land t.course = 101 \land t.result < 2.0 \}
  – SQUARE: \text{result}\text{exam}_{course, result} (101, <2.0)
• Get a list of all exam results better than 2.0 along with the according student name

  – TRC:
    \[
    \{ \text{t}_1.\text{name}, \text{t}_2.\text{result} \mid \text{Student}(\text{t}_1) \land \text{exam}(\text{t}_2) \\
    \land \text{t}_1.\text{mat}_\text{nr} = \text{t}_2.\text{student} \land \text{t}_2.\text{result} < 2.0 \}
    \]

  – SQUARE:

    name result Student mat_nr \(\circ\) student exam result (<2.0)

    Also, \(\cup\), \(\cap\), and \(\setminus\) can be used to combine SQUARE queries.
7.1 SQUARE & SEQUEL

• 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 \( \circ \) student exams crs_nr result (102, <2.0)
  – SQUARE’s syntax is too mathematical and artificial
• In 1974, Chamberlin & Boyce proposed **SEQUEL**
  – *Structured English Query Language*
  – 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

• Fundamental keywords
  – SELECT: what attributes should be retrieved?
  – FROM: what relations are involved?
  – WHERE: what conditions should hold?
• Get all exam results better than 2.0 for course 101
  
  – SQUARE:
    \[
    \text{result} \text{ exam}_{\text{course result}} (101, < 2.0)
    \]
  
  – SEQUEL:
    \[
    \text{SELECT result FROM exam WHERE course = 101 AND result < 2.0}
    \]
• Get a list of all exam results better than 2.0, along with the according student names

  – SQUARE:
    
    \[
    \text{name result} \quad \text{Student mat_no} \quad \circ \quad \text{student result} \quad \text{exam result} \quad (< 2.0)
    \]

  – SEQUEL:
    
    \[
    \text{SELECT name, result}
    \]
    
    \[
    \text{FROM Student, exam}
    \]
    
    \[
    \text{WHERE Student.mat_no = exam.student}
    \]
    
    \[
    \text{AND result < 2.0}
    \]
• 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)
    • Structured Query Language
  – patented in 1985 by IBM
• Since then, SQL has been adopted by all(?) relational database management systems

• This created a need for standardization:
  – 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
  – \( \{ \ x_1, \ldots, x_n \mid \text{CONDITION}(x_1, \ldots, x_n) \ \} \)
  – \( x_1, \ldots, x_n \) 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:** \( (x \, \theta \, y) \)
    - \( x_i \) and \( x_j \) are domain variables
    - \( \theta \) 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
    - \( \theta \) 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
Retrieve first name and last name of all female students.

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

\[ \{ \text{t.firstname, t.lastname | Student(t) \land t.sex = 'f'} \} \]

\[ \{ \text{fn, ln | } \exists \text{mat, s (Student(mat, fn, ln, s) \land s = 'f')} \} \]

<table>
<thead>
<tr>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
<tr>
<td>2832</td>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
</tr>
<tr>
<td>4512</td>
<td>Lex</td>
<td>Luther</td>
<td>m</td>
</tr>
<tr>
<td>5119</td>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
</tr>
<tr>
<td>6676</td>
<td>Erik</td>
<td>Magnus</td>
<td>m</td>
</tr>
<tr>
<td>8024</td>
<td>Jeanne</td>
<td>Gray</td>
<td>f</td>
</tr>
<tr>
<td>9876</td>
<td>Logan</td>
<td></td>
<td>m</td>
</tr>
</tbody>
</table>

\begin{align*}
\{ \text{mat, fn, ln, s | Student(mat, fn, ln, s) \land s='f'}\} \\
\{ \text{fn, ln | } \exists \text{mat, s (Student(mat, fn, ln, s) \land s='f')} \} \\
\end{align*}
7.2 DRC: Examples

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

Relational algebra: $\pi_{\text{firstname}} (\text{Student} \bowtie_{\text{mat_no}=\text{student}} \text{exam})$

TRC: $\{ t_1.\text{firstname} |$ $\text{Student}(t_1) \land \exists t_2 (\text{exam}(t_2) \land t_2.\text{student} = t_1.\text{mat_no}) \} $

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

<table>
<thead>
<tr>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
<tr>
<td>2832</td>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
</tr>
<tr>
<td>4512</td>
<td>Lex</td>
<td>Luther</td>
<td>m</td>
</tr>
<tr>
<td>5119</td>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>9876</td>
<td>100</td>
<td>3.7</td>
</tr>
<tr>
<td>2832</td>
<td>102</td>
<td>2.0</td>
</tr>
<tr>
<td>1005</td>
<td>101</td>
<td>4.0</td>
</tr>
<tr>
<td>1005</td>
<td>100</td>
<td>1.3</td>
</tr>
</tbody>
</table>
• Reconsider last lecture: Algebra Division

– \( R \div S \)
  – Read relational algebra division as a “\textit{forall}” statement

– Given relation \( R \) and \( S \):
  – \( R(a_1, \ldots, a_n, b_1, \ldots, b_m) \)
  – \( S(b_1, \ldots, b_m) \)

– \( R \div S = \{ a_1, \ldots, a_n \mid \forall b_1, \ldots, b_m (\neg S(b_1, \ldots, b_m) \lor R(a_1, \ldots, a_n, b_1, \ldots, b_m)) \} \)

= \{ 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)) \} \)
### Division:

\[
SC = \rho_{SC} (\pi_{\text{matnr, lastname, crsnr}} (\text{Student} \bowtie_{\text{matnr=student exam}}))
\]

<table>
<thead>
<tr>
<th>matnr</th>
<th>lastname</th>
<th>crsnr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Kent</td>
<td>100</td>
</tr>
<tr>
<td>1000</td>
<td>Kent</td>
<td>102</td>
</tr>
<tr>
<td>1001</td>
<td>Lane</td>
<td>100</td>
</tr>
<tr>
<td>1002</td>
<td>Luther</td>
<td>102</td>
</tr>
<tr>
<td>1002</td>
<td>Luther</td>
<td>100</td>
</tr>
<tr>
<td>1002</td>
<td>Luther</td>
<td>101</td>
</tr>
<tr>
<td>1003</td>
<td>Xavier</td>
<td>103</td>
</tr>
<tr>
<td>1003</td>
<td>Xavier</td>
<td>100</td>
</tr>
</tbody>
</table>

\[
CCK = \rho_{CCK} (\pi_{crsnr} \sigma_{\text{lastname='Clark Kent'}} SC)
\]

<table>
<thead>
<tr>
<th>crsnr</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>102</td>
</tr>
</tbody>
</table>

\[
SC \div CCK = \{ \text{matnr, lastname} | \forall \text{crsnr} (CCK(\text{crsnr}) \rightarrow SC(\text{matnr, lastname, crsnr})) \}
\]

Result contains all those students who took at least the same courses as Clark Kent.
7.2 Domain Relational Calculus

• In DRC, when dealing with equality, you can shorten your expressions a bit:

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

• Which is also useful for joins:

\[
\{fn \mid \exists mat, ln, r \ (Student(mat, fn, ln, r) \\
\land \exists st, co, r \ (exam(st, co, r) \land st = mat))\} = \{fn \mid \exists mat, ln, r \ (Student(mat, fn, ln, r) \land \exists co, r \ (exam(mat, co, r)))\}
\]

• As a shorthand, sometimes also a **formally inaccurate** notation is used:
  
  – use implicit existential quantifiers
  
  – \{fn, ln \mid Student(mat, fn, ln, 'f')\}
  
  – \{fn \mid Student(mat, fn, ln, s) \land \exists co, r(exam(mat, co, r))\}
  
  – Don’t do this in the exam please!
• 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)**
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.
7.2 QBE

• 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.

• **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

\[ \pi_{\text{mat_no}} \sigma_{\text{lastname} = \text{‘Parker’}} \text{Student} \]

\{ mat | \exists \text{fn, ln, s} (\text{Student(mat, fn, \text{‘Parker’}, s))} \}

<table>
<thead>
<tr>
<th>Student</th>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td></td>
<td></td>
<td>Parker</td>
<td></td>
</tr>
</tbody>
</table>

\{ st | \exists \text{co, r} (\text{exam(st, co, r)} \land r > 2.0) \}

<table>
<thead>
<tr>
<th>exam</th>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td></td>
<td></td>
<td>&gt; 2.0</td>
</tr>
</tbody>
</table>

\{ st | \exists \text{co, r} (\text{exam(st, co, r)} \land \text{co} \neq 102) \}

<table>
<thead>
<tr>
<th>exam</th>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td></td>
<td>\neq 102</td>
<td></td>
</tr>
</tbody>
</table>
Add a row if you need to connect conditions

- Get the matriculation number of students who took exams in courses 100 and 102.
- \{st | \exists r (\text{exam}(st, 100, r)) \land \exists r(\text{exam}(st, 102, r))\}

_st is the example in query-by-example!

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

<table>
<thead>
<tr>
<th>exam</th>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>P._st</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_st</td>
<td>102</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P.</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P.</td>
<td>102</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Get the matriculation number of those students who took at least one course that also the student 1005 took.

<table>
<thead>
<tr>
<th>exam</th>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>P._co</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>_co</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

• Get the average results of each student.

<table>
<thead>
<tr>
<th>exam</th>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.P._st</td>
<td>_res</td>
<td>_res</td>
<td>P.AVG._res</td>
</tr>
</tbody>
</table>
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?*

<table>
<thead>
<tr>
<th>Student</th>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>_st</td>
<td></td>
<td>P.</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>exam</th>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>_st</td>
<td></td>
<td></td>
<td>&lt; 1.3</td>
</tr>
</tbody>
</table>
• Besides the DML aspect for querying also the DDL aspect is covered
  – single tuple insertion
  – or from other tables by connecting them with domain variables
  – insert (I.), delete (D.), or update (U.)
    • update directly in columns

<table>
<thead>
<tr>
<th>Student</th>
<th>mat_no</th>
<th>firstname</th>
<th>lastname</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>1005</td>
<td>Clark</td>
<td>Kent</td>
<td>m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>exam</th>
<th>student</th>
<th>course</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2832</td>
<td>102</td>
<td>U._res</td>
<td>+1.0</td>
</tr>
</tbody>
</table>
• 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
• Example: *What results did the student with last name Parker get in the exams?*
• 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)
• Tuple relational calculus
  – SQUARE, SEQUEL

• Domain relational calculus
  – Query-by-example (QBE)

• 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 relational complete
7.3 Relational Completeness

• Which parts are relational complete?
  – **basic** relational algebra
    • just five basic operations
    • Selection $\sigma$, Projection $\pi$, Renaming $\rho$, Union $\cup$, Set Difference $\setminus$, Cartesian Product $\times$
  – **safe** TRC queries
  – **safe** DRC queries
• Also, **extended basic relational algebra** is relationally complete
  – Intersection ∩, Theta Join \( \bowtie_{(\theta \text{-cond})} \), Equi-Join \( \bowtie_{(\equiv \text{-cond})} \),
    Natural Join \( \bowtie \), Left Semi-Join \( \ltimes \), Right Semi-Join \( \triangleright \),
    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 \( \leftjoin \), Right Outer Join \( \rightjoin \), Full Outer Join \( \fulljoin \),
    Aggregation \( \sum \)
  – these operations **cannot be expressed** with either DRC, TRC, nor with Basic Relational Algebra
7 Next Lecture

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