Tutorial
Christian-Albrechts University Kiel,
Information Systems Engineering

Visual SQL
An ER-Based Introduction
to Database Programming

Bernhard Thalheim

Christian Albrechts University Kiel, Germany
Computer Science Institute, Information Systems Engineering
thalheim@is.informatik.uni-kiel.de
Introduction

Outline

Visual SQL as database description language

Visual SQL and intelligent diagramming of queries

Visual SQL and specification of volatile data containers

Translation profile of Visual SQL specifications to SQL’92, SQL’99 and SQL’2003

Visual SQL and SQL tuning

Conclusion and tool support
Visualization is not the silver bullet

Visualization may mislead

Misleading comparisons: *Gravitation decreases by the square of the distance.*

Moore’s, Gilder’s or Metcalfe’s laws without context

*Metcalfe: The value of a network is proportional to the square number of nodes.*

Coloring schemes, e.g., red color for *attention* in some cultural environments...

Representation of complex structures, e.g., in medicine

Exclusive reasoning on representations, e.g., in ER diagrams

Software measures based on metrics without explicit quality criteria that have been deduced from the requirement and the environment

Simplicity of mind maps, topic maps or tree-structured ontologies, e.g., Carl von Linne’s biological classification

TV, mass media, movie “information”, e.g., war pictures, interpretation without background, rewritten history, physics in TV
Visual SQL

in a nutshell

Object-relational diagram with essential types and attributes

Comparison and aggregation operators beside the classical functions of the relational algebra

Views based on a sub-graph representation

Retrieval language using output ticks and sub-diagrams

Update language based on the visual representation

Path language similar to XPath (but on semantically correct grounds)

Fully fledged semantics based on HERM logical calculus

Graphical representation of constraints and their enforcement policy

Potentially explicit representation of trigger suites and stored procedures
1. **SQL**: Extend the request utterance for disambiguation, elimination of ellipses, closed-world semantics, reduction of fuzziness

2. **SQL**: Reformulation of the query into an existential form

3. Map the request terms to schema types under consideration of translation policy, using auxiliary tables and translation to algorithms

4. **SQL**: Map result concepts to SQL answer types
Our Database Schema Considered

English version and enlarged see last (special) page
**Good old SQL**

Provide data on students who have successfully completed those and only those courses which have successfully been given or which are currently given by the student’s supervisor?

```
SELECT P1.Name, P1.BirthDate, P1.Address,
P2.Name AS "Name of supervisor"
FROM Person P1, Professor P2, Student S1, Supervisor, Lecture L,
Enroll E
WHERE P1.Name = Student.Name AND P1.BirthDate = Student.BirthDate
AND S1.StudNo = E.StudNo
AND E.Result NOT NULL
AND S1.StudNo = Supervisor.StudNo
AND Supervisor.Name = Professor.Name
AND Supervisor.BirthDate = Professor.BirthDate
AND P2.Name = Professor.Name AND P2.BirthDate = P2.BirthDate
AND L.Name = Professor.Name AND L.BirthDate = Professor.BirthDate
AND L.CourseNo
IN (SELECT E2.CourseNo
    FROM Enroll E2
    WHERE S1.StudNo = E2.StudNo AND
          E2.Result NOT NULL )
AND
E.CourseNo
IN (SELECT L2.CourseNo
    FROM Lecture L2
    WHERE 
          L2.Name = P2.Name AND
          L2.BirthDate = P2.BirthDate );
```
Provide data on students who have successfully completed those and only those courses which have successfully been given or which are currently given by the student’s supervisor?
**Advantages of Visual SQL**

Visual SQL is more natural and fits better to linguistic environments.

Syntactic and semantic quality raises for complex queries.

Object-relational technology can be better treated on the basis of Visual SQL.

Simple maintenance and correction of query formulations.

Easy correction and trace of errors in queries.
Schema Definition through Visual SQL

Person
- Name: char[40]
- BirthDate: date
- BirthPlace: char[20]
- Address: char[60]

Student
- StudNo: char[7]
- Name: char[40]
- BirthDate: date

Professor
- Name: char[40]
- BirthDate: date
- Specialization: varchar[50]
- In.InstName: char[15]

SProgramm
- SName: char[20]
- ResponsInst: char[15]

Enroll
- StudName: char[20]
- From: date
- To: date

Supervisor
- StudNo: char[7]
- SupervName: char[40]
- SupervBirthDate: date
- From: date
Attribute Definition in Visual SQL

Name - unique for the type

Data type with storage types and representation types, default values, operations and predicates

Involvement into IC (primary, secondary, foreign) keys, cardinalities of relationships, ...

Meaning of the attribute for the type

Semantics - domain constraints, null values (does not exist, unknown, inconsistent, many, no information), default values with meaning, cardinalities (minimal, maximal, average)

IC enforcement policy - checking mode (immediate, deferred), triggering, scope, checking time (before, after), row/statement level
Foreign Keys and FD’s in Visual SQL

**Institute**

- **RoomSecretary** → **Phone**
- **Head**
- **InstName** → **Faculty**
- **BudgetNo** → **Postbox**

**(Partial) Foreign Key**

- **Professor**
- **OrgUnit**

**Foreign Key**
### Data Types for Attributes with Conditions

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_SProgramm</td>
<td>π</td>
<td>TIMESTAMP (Def = CURRENT)</td>
</tr>
<tr>
<td>SProgrammName</td>
<td>π (O)</td>
<td>char(18)</td>
</tr>
<tr>
<td>Supervisor</td>
<td>π (O)</td>
<td>char(18)</td>
</tr>
<tr>
<td>Advisor</td>
<td>π</td>
<td>varchar(18) ¬0 (Def = 'not assigned')</td>
</tr>
<tr>
<td>ID_Institute</td>
<td>π (O)</td>
<td>char(10) ¬0</td>
</tr>
</tbody>
</table>

(\text{\textit{i}}_{\text{\text{SProgramm}}} = \text{NoAction}, \text{\textit{d}}_{\text{\text{Institute}}} = \text{Restrict}, \text{\textit{u}}_{\text{\text{Institute}}} = \text{Cascade}, \text{\textit{u}}_{\text{\text{SProgramm}}} = \text{NoAction})

INDEX
Constraint Definition in Visual SQL

\[
(1,.) : \ (i_{Stud} = C, d_{Enr} = R, u_{Stud} = R, u_{Enr} = R) \\
(.,2) : \ (i_{Enr} = R, u_{Stud} = C, u_{Enr} = R)
\]

<table>
<thead>
<tr>
<th>Student</th>
<th>char[7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudNo</td>
<td>〇</td>
</tr>
<tr>
<td>Name</td>
<td>〇</td>
</tr>
<tr>
<td>BirthDate</td>
<td>〇</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enroll</th>
<th>char[7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stud.StudNo</td>
<td>〇</td>
</tr>
<tr>
<td>SName</td>
<td>〇</td>
</tr>
<tr>
<td>From</td>
<td>date</td>
</tr>
<tr>
<td>To</td>
<td>date</td>
</tr>
</tbody>
</table>

CHECK ( To > From )
Assignment of Rights

RoomPlanning

\[ \sqrt{1} \text{CourseNo} \]

\[ \sqrt{2} \text{Lecturer} \]

\[ \sqrt{3} \text{Room} \]

\[ \sqrt{4} \text{Time} \]

GRANT INSERT, UPDATE TO Panzer

WITH GRANT OPTION
Modification of Database Objects

Insert of Values in Tables

<table>
<thead>
<tr>
<th>Student</th>
<th>INSERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudNo</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td></td>
</tr>
</tbody>
</table>

VALUES

<table>
<thead>
<tr>
<th>VALUES</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>0815</td>
</tr>
<tr>
<td>“Alfons ”</td>
<td>“Amanda ”</td>
</tr>
<tr>
<td>29.2.1982</td>
<td>24.12.1982</td>
</tr>
</tbody>
</table>
Modification by Queries

**Student**
- StudNo
- Name
- BirthDate

**INSERT**

```
SELECT A,B,C FROM ... WHERE ...
```
Modification of Values in Tables

UPDATE Student

StudNo = 21203
StudNo = 0815

Name
BirthDate
Parallel Modification of Several Values in Tables

<table>
<thead>
<tr>
<th>Person</th>
<th>UPDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name = “Gerhard” “Lappus”</td>
</tr>
<tr>
<td>BirthDate</td>
<td>BirthDate = 4.12.1951</td>
</tr>
<tr>
<td>BirthPlace</td>
<td>BirthPlace = “München”</td>
</tr>
<tr>
<td>Address</td>
<td>Address = “0355 Cottbus-Sielow”</td>
</tr>
</tbody>
</table>
Modification with Values from a Query

```
Enroll
---
Stud.StudNo
SName
From
To

UPDATE

Student
---
√ StudNo
Name
BirthDate

From = 7.10.2002
NULL

= “Hans Meyer”
= 13.12.1982
```
Modification with Visual SQL Queries

```
DELETE

NOT EXISTS

Enroll
√ Stud.StudNo
SName
From
To
```
Querying in Visual SQL

Institute

<table>
<thead>
<tr>
<th>InstName</th>
<th>LIKE &quot;%_nformatik%&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone</td>
<td>OR₁</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Faculty</td>
<td>AND₂</td>
</tr>
<tr>
<td>Head</td>
<td></td>
</tr>
<tr>
<td>Postbox</td>
<td></td>
</tr>
<tr>
<td>BudgetNo</td>
<td></td>
</tr>
<tr>
<td>RoomSecr</td>
<td></td>
</tr>
</tbody>
</table>

LIKE "%LG 2%"

LIKE "K%"
Counting Distinct Objects

Count the return of courses, number of participants, and average grading for winter term 2002/2003 for all courses and lecturers.
Counting within Groups with Side Condition

ORDER BY Lecturer

GROUP BY CourseNo, Lecturer

Enroll

StudNo

CourseNo

Semester

Lecturer

Result

\( \sqrt{NoOfPart} = \text{COUNT}(*) \)

\( \sqrt{AVG(\text{Result})} \)

HAVING \( \text{AVG(\text{Result}) < 2.50} \)

Search for all courses and number of participants in the winter term 2002/2003 and lecturers with average grading at least “good”.

\( \text{WS2002/03} \)
What is the impact of the Cottbus University for Cottbus students?

```
SELECT CourseNo, Semester, Person.Name, Person.BirthDate
FROM Enroll, Student, Person
AND Person.BirthDate = Student.BirthDate
AND BirthPlace LIKE "%Cottbus%";
```
Shortening of Aliased Queries with Integrity Constraints

Which student (name) is supervised by whom (name) ?

Supervisor[Name,BirthDate] ⊆ Professor[Name,BirthDate] ⊆ Person[Name,BirthDate]
*Intelligent Shortening of Queries by Tools*

Which student *(name)* is supervised by whom *(name)*?

Supervisor[Name,BirthDate] ⊆ Professor[Name,BirthDate] ⊆ Person[Name,BirthDate]
Volatile data

Domain types  based on domain specifications with integrity constraints

Blocks  based sub-schema specification

Temporary tables  based on sub-schema specification with check-in and check-out facilities

View towers  based on parametric sub-schema specification for application data views and functionality views (e.g., OLAP views) with visualization of check option

Other data:

- Complex applications (regions, sequences, runs, arrays), trees, graphs, bounded recursion
- Complex groupings, partitions, relational operations, statistical functions, set operations, and aggregations
- Object-relational data, versions, alternatives, configurations

Non-volatile data:  relations (bags), indices, materialized views
Grouping for Temporary Data Computation

Get all student names whose average grading is not less than the average grading of all students.

```
GROUP BY Name

Person
  √ Name
  BirthDate
  ...

Student
  StudNo
  Name
  BirthDate
  ...

Enroll
  StudNo
  Semester
  CourseNo
  Result
  ...

HAVING AVG(Result) <= ANY

√1 AVG(Result)

Enroll
  StudNo
  Semester
  CourseNo
  Result
  ...

GROUP BY StudNo

Person
  Name
  BirthDate
  ...

Student
  StudNo
  Name
  BirthDate
  ...

Enroll
  StudNo
  Semester
  CourseNo
  Result
  ...

GROUP BY StudNo

Enroll
  StudNo
  Semester
  CourseNo
  Result
  ...

HAVING AVG(Result) <= ANY

√1 AVG(Result)

Enroll
  StudNo
  Semester
  CourseNo
  Result
  ...

GROUP BY StudNo

Enroll
  StudNo
  Semester
  CourseNo
  Result
  ...

HAVING AVG(Result) <= ANY

√1 AVG(Result)
```
Reasoning on Equivalent Block Queries

Which student has a clean record, i.e., all grades reached are “1”?
Reasoning on Equivalent Block Queries

Which student has a clean record, i.e., all grades reached are “1”?

\[
\begin{align*}
\text{Person} & \quad = \\
\text{Student S1} & \quad = \\
\{ H1 \} & \quad = \{ H2 \} \\
\text{Student S2} & \quad = \\
\text{Enroll H1} & \quad = \\
\text{Enroll H2} & \quad = 1
\end{align*}
\]
Reasoning on Equivalent Block Queries

Which student has a clean record, i.e., all grades reached are “1”?

Person
- Name
- BirthDate

Student S1
- StudNo
- Name
- BirthDate

Student S2
- StudNo
- Name
- BirthDate

Enroll H2
- CourseNo
- Semester
- Result

\[ \text{Result} \in \{2,3,4,5\} \]
Development of Temporary Relations for Querying

Which student has the best GPA (grade-point-average)?

\[
\text{TEMPORARY \ Achievem} \\
\sqrt{1} \ \text{StudNo} \ \text{char}(7) \\
\sqrt{1} \ \text{Name} \ \text{char}(40) \\
gpa \ \text{float}
\]

\[
\text{GROUP BY StudNo} \ \\
\text{Student S} \\
\sqrt{2} \ \text{StudNo} \\
\sqrt{2} \ \text{Name} \\
\text{BirthDate} \\
\text{...} \\
\text{NOT NULL} \\
\sqrt{2} \ gpa = \frac{\text{SUM}((5 - \text{Result}) \times \text{Credits})}{\text{SUM}(\text{Credits})}
\]

\[
\text{Enroll} \\
\text{StudNo} \\
\text{Semester} \\
\text{CourseNo} \\
\text{...} \\
\text{Result}
\]

\[
\text{Lecture} \\
\text{Lecturer} \\
\text{Credits} \\
\text{Semester} \\
\text{CourseNo} \\
\text{...}
\]

\[
\text{IN} \\
\sqrt{1} \ \text{MAX}(gpa)
\]
Which professor has the best average grading?

```
TEMPORARY Darling
   AverageRes  float
   Number      smallint
   Lecturer    char(40)
   BirthDate   date
```

```
Professor
   Name
   BirthDate
   InstName
   ...

Darling
   Name
   BirthDate
   Anzahl
   AverageRes

Enroll
   StudNo
   Semester
   CourseNo
   ...

Lecture
   Lecturer
   BirthDate
   Semester
   CourseNo
   ...

Result
   NOT NULL
   AVG(Result)
   Number = COUNT(Result)
   group by Lecturer, BirthDate

1 MIN(AverageRes)
```
Introduction of views

All names of students together with their supervisors.

Person P1
- √ Name
- BirthDate
- BirthPlace

Student S1
- StudNo
- Name
- BirthDate

Supervisor
- StudNo
- BetrName
- BetrBirthDate

Professor P3
- Name
- BirthDate

Person P2
- √ Name
- BirthDate

Database systems I, Summer term 07  © β  IS Engineering  Christian Albrechts University Kiel, Germany  35
Volatile values

Which project collaborator earns less than the average student project member?

```
StProjMem
| StudNo | CollID | From | ...
|--------|--------|------|------
Collaborator
| CollID | ...    | Income
|        |        | AVG(Income)
```

Collaborator
| √ CollID |
|          |
| √ Income |
Solving complex requests through views

For the planning of compound course we need information on free lecture halls in which \( n \) consecutive lectures can be given within a term.

CREATE VIEW RoomFree (Room, Day, Slot) AS

FROM Lecture V1, Lecture V2, Lecture V3
    NOT IN (SELECT Room, Day, Slot
             FROM Lecture
             WHERE Semester = "WS2004")

;

SELECT Room, Day, Slot AS Beginn_Free, 'To', Slot + (:n -1) AS Ende_Free
FROM RoomFree F
WHERE NOT EXISTS (SELECT *
                 FROM Lecture V
                 WHERE V.Room = F.Room AND V.Day = F.Day AND
                 V.Slot BETWEEN (F.Slot + 1) AND (F.Slot + (:n -1))
                 AND V.Semester = "WS2004"
                 )

;
NOT EXISTS

\[
\text{RoomFree} \quad F
\]

\[
\begin{align*}
\sqrt{1} & \text{ Room} \\
\sqrt{2} & \text{ Day} \\
\sqrt{3} & \text{ Slot}
\end{align*}
\]

\[
\text{Lecture}
\]

\[
\begin{align*}
\text{Room} \\
\text{Day} \\
\text{Slot} \\
\text{Semester}
\end{align*}
\]

\[
\text{BETWEEN } F.\text{Slot} + 1 \text{ AND } F.\text{Slot} + ( :n - 1)
\]

\[
= \quad \text{“WS2004”}
\]

\[
\text{NOT IN}
\]

\[
\text{Lecture}
\]

\[
\begin{align*}
\text{Room} \\
\text{Day} \\
\text{Slot} \\
\text{Semester}
\end{align*}
\]

\[
= \quad \text{“WS2004”}
\]
Visual SQL Translation Profiles

HERM: one extended ER model that supports compact representation and has a well-defined semantics

Object-relational model: ID-based treatment with complex attributes (reference values, structured values, collections (finite sets, finite lists, arrays)), reference semantics, behavior based on methods

Relational model: atomic attributes, relations, complex constraint treatment

SQL-92 model: atomic attributes, tables, restricted constraint treatment

Aim for Visual SQL mapping
to SQL-92 (e/i/f), SQL:1999, SQL:2003 mapping

• homogeneous

• bijective mapping

• for all types
Mappings Consider

Treatment of hierarchies

Controlled redundancy with corresponding functionality

Null and default values support restricting functionality of types

Enforcement of constraints beside key and domain constraints

Naming conventions and abbreviation rules

Set or pointer semantics

Utilization of weak types

Translation of complex attributes

Global or type-wise translation
**Treatment of Hierarchies**

**Event non-separation approach:** Types are separated from their subtypes.
- class inclusion constraints

**Event separation approach:** Hierarchy is partitioned into disjoint types.
- object belongs either to one or more of the subtypes or it belongs to the supertype and none of its subtypes
- exclusion constraints

**Union approach:** The hierarchy is merged into one type.
- additional attributes for type information

**Universal relation approach:** union approach + embedding relationship types

**Generalization and specialization**

**Strong specialization:** Subtypes have their specific attributes and inherit one key from the supertype

**Strong generalization:** Subtypes have all attributes.
- supertype has only the common key attributes and attributes specific for the supertype

**Mixed approach**
Controlled Redundancy

One way to improve performance is to reduce join operations.

Alternatives:

- Attributes can be added from one relation scheme to another and thereby eliminate access to one or more relations.

- Relations can be combined into one relation. Hence the number of relations to be accessed is reduced.

- ID-extension leads to addition of uniqueness constraints.
  ID extension for complex keys

Introducing controlled redundancy may result in additional integrity problems which we must resolve through other facilities.
Null and Default Value Support

14 kinds of incomplete data

- Currently unknown values
  combined with specific default values
  Gender: 0 (unknown), 1 (male), 2 (female), 9 (inapplicable).

- Domain-specific null values for ordinal (measure position) or
  cardinal numbers (quantity, magnitude).
  0 - common default value for all numeric domain types
  blank - default type for character types
  date and time: relative values

- Inapplicability of a characterization for a given object

Null values can be derived

Treatment of null and default values is different in DBMS
Enforcement and Treatment of Constraints

- Enforcement without restrictions, with restrictions or not applied
- Enforcement of constraints can be deferred until an event occurs
- Enforcement uses null values or default values

Facilities provided by DBMS:

- Key-based inclusion constraints: referential integrity constraints
- Uniqueness: uniqueness constraints and indexes
- Triggers, stored procedures
- Assertions or check conditions can be used in some DBMSs

The set of enforcement rules must be consistent in the whole schema

Treatment of cardinality constraints

Treatment of inherent constraints (component inclusion constraint, declarative or procedural treatment)
Naming and Abbreviation Conventions

- **Generate names** used in the relational schema
- **Abbreviation rules** while partially preserving the type name the attribute is originating from
  
  unique paths: omit all path components which are unique
  omitting full stop and concatenating names
  short name by the shortest extension

- **Type names**
  uppercase or lowercase names with/without accent.

- **Integrity constraint names**

- **Role extension** for roles of types
Set or Pointer Semantics

- Relationship types: pointers instead of key values
  some DBMS force the introduction of ID’s

- Optional additional attribute “ID”: ‘surrogate’ or ‘identifier’

- ID extension for hierarchies

- Maintenance of pointers
  cycles of pointers
  loosing value distinction and thus understandable counting abilities

- Pointer domain as one type or separated by types

be careful: pointer semantics is based on intuitionistic logic and topos semantics
Treatment of Weak Types

Identification extension: External identifiers can be eliminated by including the identifier into the weak type. The cardinality constraints are then changed to one-to-many constraints.

Embedding into parent types: Weak types can be eliminated through embedding into their parent type. This leads to relation schemes which are not normalized used for OLAP applications.

HERM subtyping and treatment of relationship components is the natural alternative for weak types.
Translation of Tuple Constructor

Flattening of complex attributes: The attribute is replaced by an attribute which eliminates the components and concatenates the components with or without use of a delimiter.

\[ \text{Addr(Zip,Town,Street(Name,No))} - \text{Address} \]

Leaf attribute generation: The attribute tree is represented by a set of attributes with complex attribute names representing the path from the root to the corresponding leaf. If the attribute has been an element of a key or is used in a dependency then the set of attributes is used instead of the original attribute.

\[ \text{Addr(Zip,Town,Street(Name,No))} - \{ \text{Addr.Zip, Addr.Town, Addr.Street.Name,Addr.Street.No} \} \]

Invariance of complex attributes: The attribute remains in its form.
Translation of List Constructor

Flattening of complex attributes: Introducing an attribute representing a string with components of the original domain with or without delimiters.

\[ \text{FirstNames} < \text{FirstName} > - \text{FirstNames} \]

Leaf attribute generation: If a cardinality restriction applies then the attribute can be represented by a tuple construction.
Extending keys by the new tuple type

Separate schema generation: Using an attribute denoting the order of elements in the list a

\[ \text{Ingredients} < \text{Ingredient} > \text{ in Recipe} \]

\[ \text{RecipeIngredient} = (\{ \text{RecipeName}, \text{OrderNo}, \text{Ingredient} \}) , \]
\[ \text{key(RecipeIngredient)} = \{ \text{RecipeName}, \text{OrderNo} \} \]

Invariance of complex attributes
Translation of Array / Vector Constructor

Flattening of complex attributes: see before

\(\text{AttendNo}_{26}^{1}(\text{Institute})\) transferred to \(\text{AttendNo}(\text{Institute, Value})\) or simply to \(\text{Institute, AttendNo}\)

Leaf attribute generation represented by a tuple construction

\(\text{EnrollmentSummary}\) with \(\text{AttendNo}_{6}^{1}(\text{Institute})\) and \(\text{Year - Year, Inst1, Inst2, Inst3, Inst4, Inst5, Inst6}\)

Separate schema generation: If high length then introduce a separate relation.

Invariance of complex attributes
Translation of Set Constructor

Flattening of complex attributes: see before
  Domain set restricted - bit representation

Leaf attribute generation: With cardinality restriction with \( n \) then tuple attribute with \( n \) components.
  \( \text{AcadTitles}\{\text{AcadTitle}\} - \text{AcadTitle1, AcadTitle2, AcadTitle3} \)

Separate schema generation: If no cardinality restriction then a new relation scheme can be introduced
  \( \text{Codes}\{\text{MaterialCode}\} \) in a type \( \text{Product} \) with \( \text{ProductID} \)
  \( \text{Code} \) with attributes \( \{\text{ProductID, MaterialCode}\} \) and
  \( \text{key(Code)} = \{\text{ProductID, MaterialCode}\} \).

Invariance of complex attributes

The **bag construction** can be translated with alternatives similar to the set construction.
Translation of Optional Parts

Separate representation with null values

Attaching to another attribute: The attribute is attached to another attribute in the schema

\[ \text{FamTitle} \] attached to \text{LastName}

Invariance of complex attributes

We can introduce other options depending on the constructors applicable to the object-relational model. The relational model has only sets of tuples. Attributes can be mapped either to tuples or to separate relations.
**Default Translation Options Used**

- *Event non-separation approach*
- *Strong specialization* for unary relationship types and *strong generalization* for cluster types
- *No redundancy* in types except referential constraints
- *Null value support* for all attributes which are not bounded through attribute inheritance
- *Enforcement of constraints* on the basis of *declarative approaches* if possible
- *Component inclusion constraints* on a declarative basis
- Application of *naming conventions*
- *Identification extension* whenever key attributes become too complex
- *Invariance* of complex attributes

CASE tools have their own default profile.
Translation Profile for Visual SQL to SQL-92

- Role extension whenever names clash
- Variables are only used if they are introduced in Visual SQL
- Additional attributes
- Shortening of labels
- Blocks as subqueries
- Set containment through (NOT) EXISTS or (NOT) IN
- Integrity constraints are either mapped to declarative constraints or triggers (depending on the DBMS)
- ID extension if required by the DBMS, e.g., Oracle
Translating Set Containment

\[ A = B \iff \forall x (x \in A \iff x \in B) \iff \forall x (x \not\in A \iff x \not\in B) \]

\[ A = B \iff \forall x (x \not\in A \rightarrow x \not\in B) \land \forall y (y \not\in B \rightarrow y \not\in A) \]

\[ A \subseteq B \iff \forall x (x \in A \rightarrow x \in B) \iff \forall x (x \not\in B \rightarrow x \not\in A) \]

Which students are enrolling lectures only pairwise?

\begin{tabular}{|c|c|c|}
\hline
Student S1 & Enroll E1 & Student S2 \\
\hline
StudNo & CourseNo & StudNo \\
Name & Semester & StudNo \\
BirthDate & & \\
\hline
\end{tabular}

\{ E1 \} = \{ E2 \}

\begin{tabular}{|c|c|c|}
\hline
Student S1 & Enroll E2 & Student S2 \\
\hline
StudNo & CourseNo & StudNo \\
Name & Semester & StudNo \\
BirthDate & & \\
\hline
\end{tabular}
SELECT  S1.StudNo, S1.Name, S2.StudNo, S2.Name
FROM    Student S1, Student S2, Enroll E1, Enroll E2
        AND E1.StudNo < E2.StudNo
        AND NOT EXISTS (SELECT * FROM Enroll E3
                        WHERE E3.CourseNo || E3.Semester NOT IN
                            (SELECT E4.CourseNo || E4.Semester
                             FROM Enroll E4
                             WHERE E4.StudNo = E2.StudNo)
        AND NOT EXISTS (SELECT * FROM Enroll E5
                        WHERE E5.CourseNo || E5.Semester NOT IN
                            (SELECT E6.CourseNo || E6.Semester
                             FROM Enroll E6
                            AND E2.StudNo = E5.StudNo)
ORDER BY E1.StudNo, E2.StudNo;
valid for non-empty sets  different translation if sets may be empty
different treatment for null valued attributes
CREATE TEMPORARY TABLE Achievem
    (StudNo  char(7) PRIMARY KEY,
     Name    char(40),
     gpa     float );

INSERT INTO Achievem
    SELECT StudNo, Name,
           SUM((5-H.Result) * V.Credits) / SUM(V.Credits) AS GPA
    FROM Student, Enroll H, Lecture V
    WHERE Student.StudNo = H.StudNo AND
          H.Semester = V. Semester AND H.CourseNo = V.CourseNo
          AND H.Result IS NOT NULL
    GROUP BY StudNo;

SELECT StudNo, Name, gpa, "BestStudent" AS Student_Category
    FROM Achievem
WHERE gpa IN (SELECT MAX(gpa) FROM Achievem);

DROP TEMPORARY TABLE Achievem;
Translating the 2\textsuperscript{nd} Temporary Table Example

CREATE TEMPORARY TABLE Darling
    (Durchschn float,
     Lecturer char(40),
     BirthDate date,
     Anzahl smallint
     PRIMARY KEY (Lecturer, BirthDate, Anzahl));

INSERT INTO Darling
    SELECT Durchschn = AVG(Result), Lecturer, BirthDate,
           Anzahl = COUNT(Result)
    FROM Enroll H, Lecture V
    WHERE H.Semester = V. Semester AND H.CourseNo = V.CourseNo AND
         H.Result IS NOT NULL
    GROUP BY Lecturer, BirthDate;

SELECT Lecturer, BirthDate, InInstName AS Institute
FROM Professor P, Darling
WHERE P.Name = L.Lecturer AND P.BirthDate = L.BirthDate AND
    Durchschnitt = (SELECT MIN(AverageRes)
                     FROM Darling);

DROP TEMPORARY TABLE Darling;
**Recycling Complex Queries**

```
FROM Lecture AS VR1, Room AS R1
WHERE R1.MaxSize =
  (SELECT MIN (R2.MaxSize)
   FROM Room AS R2
   WHERE (R2.MaxSize > VR1.MaxStud)
   AND NOT EXISTS
     (SELECT *
      FROM LECTURE AS VR2
      WHERE (R2.MaxSize > VR2.MaxStud)
      AND (VR2.LectNo < VR1.LectNo)));
```

What is the result of the query?
Recycling Complex Queries

Is the previous query equivalent to the following one?

CREATE VIEW LectRoom AS SELECT *
FROM   Lecture, Room
WHERE  (MaxStud < MaxSize);

CREATE VIEW LectRoom1 AS SELECT *
FROM LectRoom AS V1
WHERE MaxSize =
(SELECT MIN (MaxSize)
FROM Room
WHERE RoomNo = V1.RoomNo);

SELECT * FROM LectRoom1 AS VR1
WHERE MaxStud =
(SELECT MAX(MaxStud) FROM LectRoom1
WHERE RoomNo = VR1.RoomNo);
ER Reasoning with Visual SQL

• Schema restructuring
  • Correcting constraints
  • Schema decomposition and pivoting
  • Detecting constraint inconsistencies

• Schema denormalization
  • Joining types
  • Joining corresponding queries

• ...

more see
Correcting Schemata (1)

Subcritical directed cycle: Tour - Start - City - Visit - Tour

Effected part Visit - Tour

Effected part Start - City

Effected part Visit - City
Correcting Schemata (2)

Subcritical directed cycle: Tour - Start - City - Visit - Tour

Effected part Visit - Tour : replace 3,4,7 by 3
Effected part Start - City : replace 4..10 by 4
Effected part Visit - City : replace 0..∞ by 12
Determining (OLAP) Shells Inside a Schema (1)

Constraints:
Lecture : \{ \text{Professor}, \text{Course} \} \rightarrow \{ \text{Assistent} \}
Lecture : \{ \text{Auditory} \} \rightarrow \{ \text{Building} \}
Lecture : \text{card}(\text{Lecture}, \text{Professor}) = (3,6)
Lecture : \text{Keys} = \{ \{ \text{Auditory, Date} \}, \{ \text{Professor, Date} \} \}
Lecture : \text{card}(\text{Lecture}, \text{Date}) = (4,5)
**Pivoting by Shells Inside a Schema (2) non-preserving**

- **Assistent**
  - ID

- **ServeLect**
  - Assistent
  - LectureID

(1,1)

- **Professor**
  - ID

(3,6)

- **Lecture**
  - ID
  - Professor
  - Course

CardLookup = 1..1

- **LectDetails**
  - LectureID
  - Date
  - Auditory

(4,5)

- **Auditory**
  - ID

- **Building**
  - ID

- **Course**
  - ID
Hartmann: ER’01
replace card(Assigns,Lab) by either = (0,30) or by = (0,∞)
Visual SQL and SQL Tuning

Develop a Visual SQL query

Create a Visual SQL query skeleton abstracting from unnecessary details

Derive an almost optimal query plan

Translate the Visual query in an SQL query

Check the query plan against DBMS execution plans

[ Alter SQL queries for best plan generation ]

[ Alter the application schema ]

*Example:* Get all order data in a order reporting database for our business unit (= 10) for those customers that use the phone number 5555555.
Gaining and adding statistics (selectivity and association cardinality)

```
SELECT SUM(COUNT(Phone_No)*COUNT(Phone_No))/
    (SUM(COUNT(Phone_No))*SUM(COUNT(*))) A1
FROM Customer
GROUP BY Phone_No;
A1: 0.000003
```
Derive an Almost Optimal Query Plan

Join order and query nesting:

- Highest selectivity: C
- Moving upwards: O ; OT
- Moving upwards: OD ; ODT
- Moving downwards: P ; S ; O

Support by indices:

- Customer(Phone_No)
- Orders(Customer_ID)
- Order_Details(Order_ID)
Check the Query Plan Against DBMS Execution Plan

Oracle execution plan through rule based optimizer ≈ DB2

PLAN

-------------------------------------------------------------
SELECT STATEMENT
SORT ORDER BY
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS
NESTED LOOPS

TABLE ACCESS FULL 4*CUSTOMER
TABLE ACCESS BY INDEX ROWID 1*ORDER
INDEX RANGE SCAN ORDER CUSTOMER ID
TABLE ACCESS BY INDEX ROWID 2*ORDER_DETAIL
INDEX RANGE SCAN ORDER_DETAIL_ORDER_ID
TABLE ACCESS BY INDEX ROWID 5*SHIPMENT
INDEX UNIQUE SCAN SHIPMENT_PKEY
TABLE ACCESS BY INDEX ROWID 6*ADDRESS
INDEX UNIQUE SCAN ADDRESS_PKEY
TABLE ACCESS BY INDEX ROWID 3*PRODUCT
INDEX UNIQUE SCAN PRODUCT_PKEY

order of nesting: CUSTOMER, ORDER, ORDER_DETAIL, SHIPMENT, ADDRESS, PRODUCT

but: full table scans of large tables
Oracle execution plan through cost based optimizer

PLAN

---------------------------------------------
SELECT STATEMENT
  SORT ORDER BY
    HASH JOIN
      TABLE ACCESS FULL 3*PRODUCT
    HASH JOIN
      HASH JOIN
        HASH JOIN
          HASH JOIN
            TABLE ACCESS FULL 4*CUSTOMER
            TABLE ACCESS FULL 1*ORDER
            TABLE ACCESS FULL 2*ORDER_DETAIL
            TABLE ACCESS FULL 5*SHIPMENT
            TABLE ACCESS FULL 6*ADDRESS

order of nesting:  PRODUCT, CUSTOMER, ORDER, ORDER_DETAIL, SHIPMENT, ADDRESS

but: full table scans of large tables and worse order of nesting and joins

better use: index on

CUSTOMER(PHONE_NO), ORDER(CUSTOMER_ID), ORDER_DETAILS(ORDER_ID)
Visual SQL - ER-Based Database Programming

Visual SQL and SQL Tuning

**Alter SQL Queries / Schema for Best Plan Generation**

Oracle execution plan through rule based and cost based optimizer
after miraculously changing `C.PHONE_NO = 5555555` to `C.PHONE_NO = '5555555'`

```sql
PLAN
-------------------------------------------------------------
SELECT STATEMENT
  SORT ORDER BY
  NESTED LOOPS
  NESTED LOOPS
  NESTED LOOPS
  NESTED LOOPS
    NESTED LOOPS
      TABLE ACCESS INDEX ROWID 4*CUSTOMER
      INDEX RANGE SCAN CUSTOMER PHONE_NUMBER
    TABLE ACCESS BY INDEX ROWID 1*ORDER
      INDEX RANGE SCAN ORDER CUSTOMER ID
    TABLE ACCESS BY INDEX ROWID 2*ORDER_DETAIL
      INDEX RANGE SCAN ORDER_DETAIL_ORDER_ID
    TABLE ACCESS BY INDEX ROWID 5*SHIPMENT
      INDEX UNIQUE SCAN SHIPMENT_PKEY
    TABLE ACCESS BY INDEX ROWID 3*PRODUCT
      INDEX UNIQUE SCAN PRODUCT_PKEY
  TABLE ACCESS BY INDEX ROWID 6*ADDRESS
    INDEX UNIQUE SCAN ADDRESS_PKEY
```

now: correct and optimal execution plan
Conclusion

Easy visual programming

Easy visual constraint maintenance

Simple development of views towers

Simple derivation of trigger frames

Simple development of stored procedures

Conceptual tuning

Formal foundations

Tool support

It’s time to realize Chen’s dream on ER-SQL.