A Vision for Management of Complex Models

Presentation by Ali

Discussion by Monir

March 9, 2009

Basic Terminology

- **Models**
  - SQL schema
  - OO interface
  - UML model
  - XML DTD
- **Mappings**
  - DTD-to-DTD
  - SQL schema-to-SQL schema
  - ER-to-SQL Schema

Applications of Mapping

- **DB design** by mapping ER model to SQL schema
- **Web site design** via models that map DB to page layout
- **Program design** by generating templates or code from a UML model
- **Generate data warehouse loading programs** from mappings of data sources to DW schema

Motivation

- Need for:
  - Transformation of data from one model to another
  - Managing change in models
- Aim:
  - Reduce programming work

Discussion question

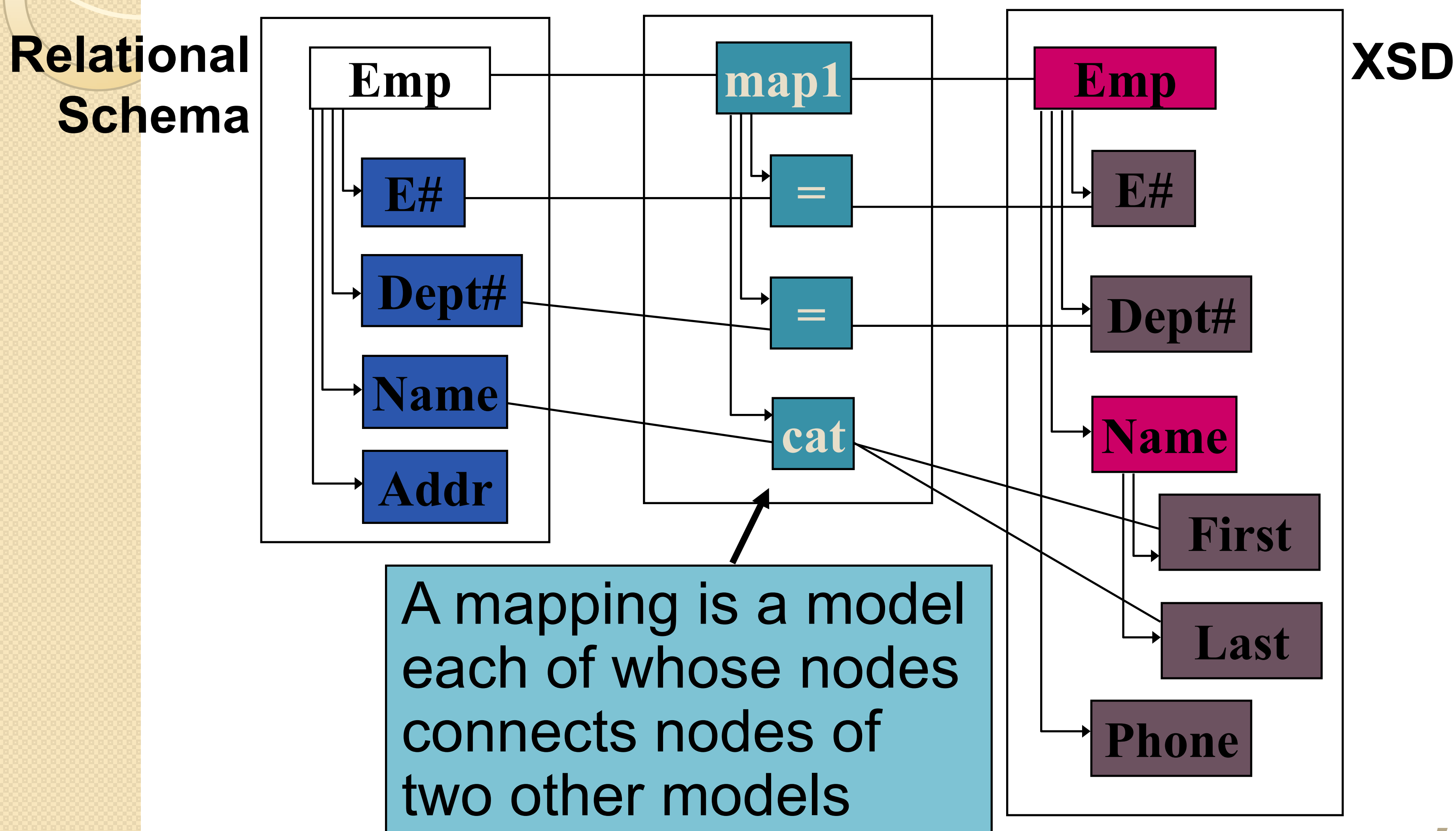
- Assume you just have a little background about database, now do you think, if having a Model Management System is feasible? Why or why not? For example think about finding an algorithm to find the best possible matching between two schemas?? Talk about the problems in designing.

Proposal

- Models and mappings are objects
- Define generic high-level operations on models and mappings

A Model for Model Management

A model is a directed graph with one root.



Challenge

- Developing a mechanism for representing models

Some Operations

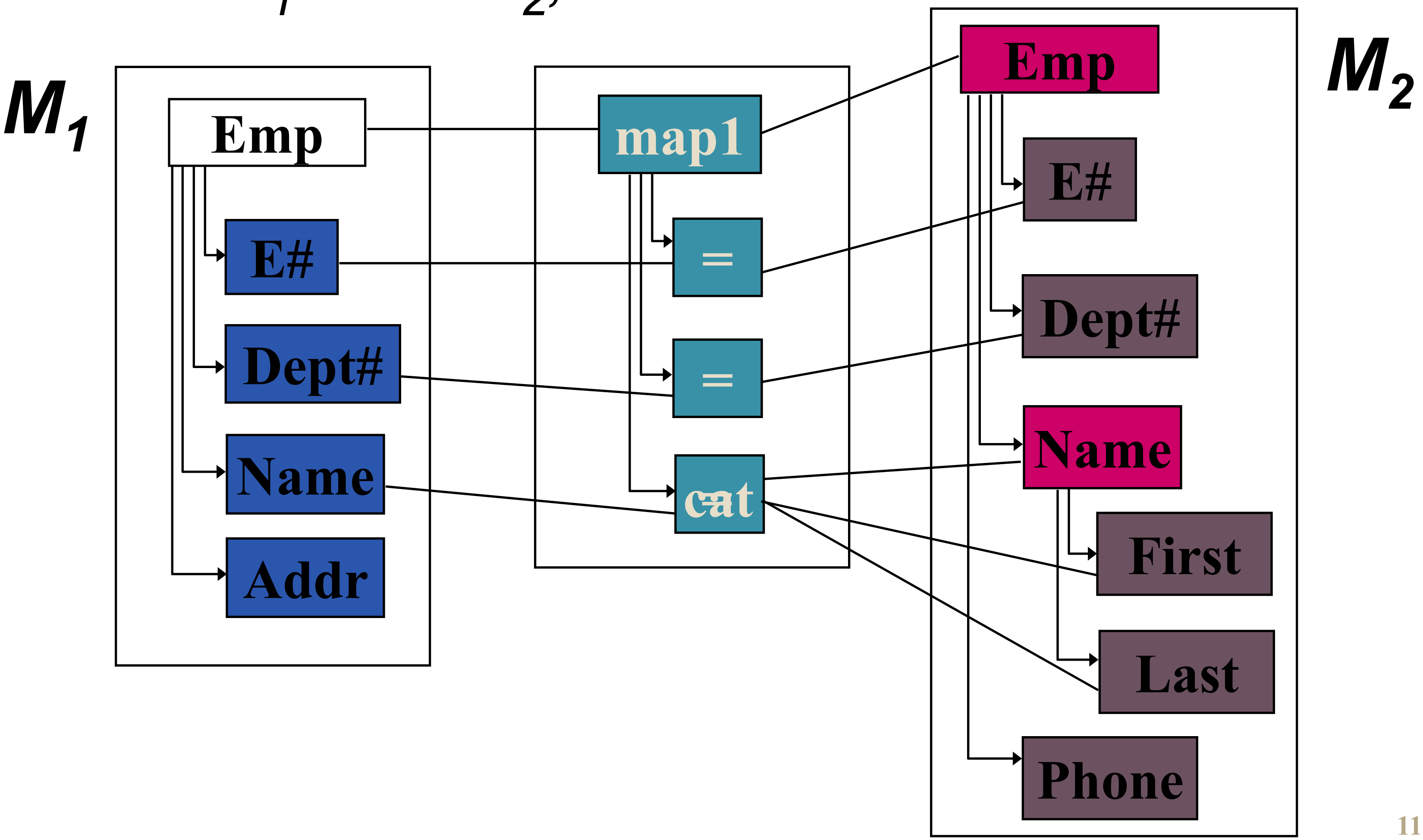
- Match(M_1, M_2, $\cong$, *map*)
- Compose(map_1, map_2)
- Merge(M_1, M_2, *map*)
- Enumerate(M)

Mapping between models

- Types:
  - Best mapping
  - Best mapping consistent with prior knowledge
  - Extend the partial mapping
- Challenge:
  - Developing an algorithm for finding extended mappings

Match

- Match(M_1, M_2, ≅) returns the best mapping between M_1 and M_2, w.r.t. to ≅

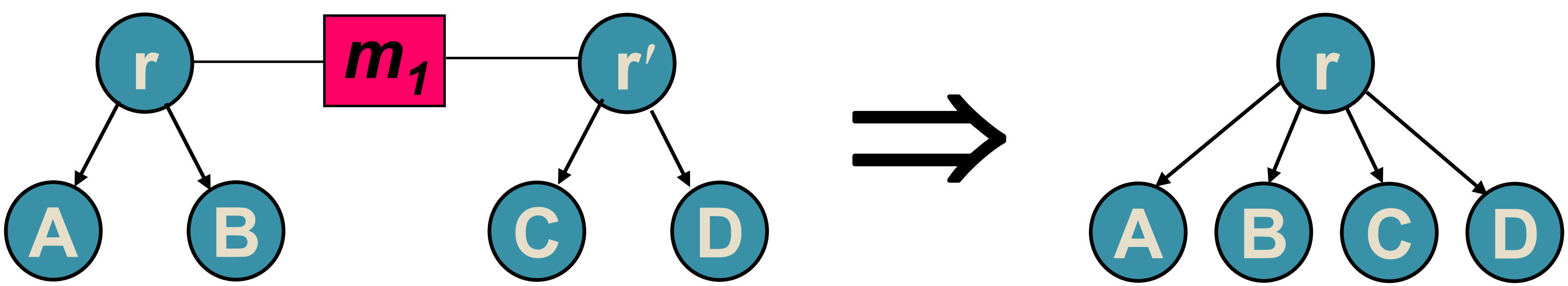


Composition

- Notation – map3 = "*map1 • map2*"
  - Map1: M1 $\longrightarrow$ M2
  - Map2: M2 $\longrightarrow$ M3
  - Map3: M1 $\longrightarrow$ M3
- Easy for single-valued functions
  - just use ordinary function composition

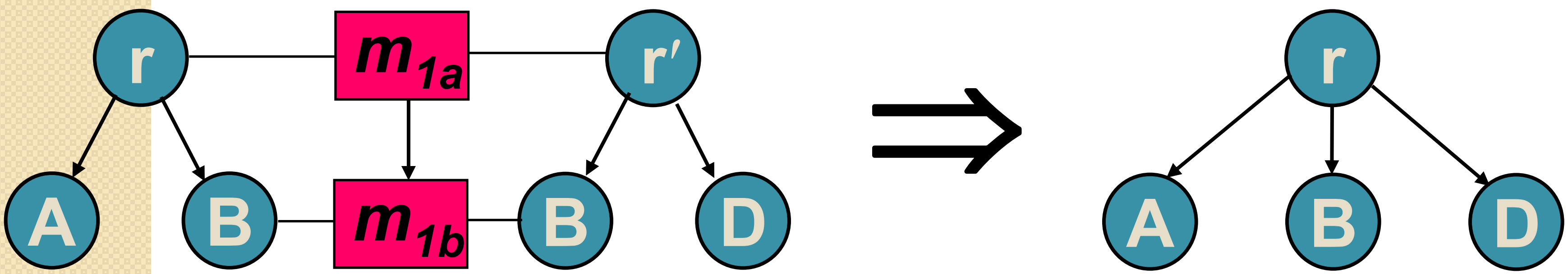
Merge(M_1, M_2, *map*)

- Move content of M_2 into M_1 model
  - Use *map* to guide the Merge
- If it's a union, just add children of M_2's root under M_1's root



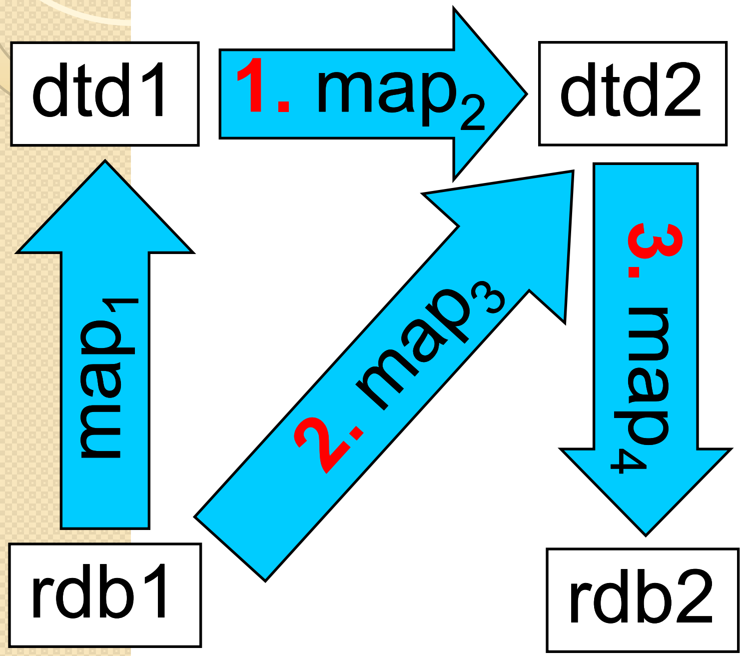
Merge (cont'd)

- To avoid copying target object m_2 that's already in M_1, connect m_2 to m_1 in *map*



- Challenge:
  - Proposing a semantic For Merge

Example



1. map_2= Match(dtd1, dtd2)

2. $map_3 = map_1 \bullet map_2$

3. <map_4, rdb2 > = Copy(map_3^{-1})

Final Challenges

- Designing an algebra of useful operations

Discussion

- Thinking about the operators used in the MMS, which operators do you think can be performed easier and which do you think are more difficult. Please rank them and say your reasons.
  - 1)Enumerate 2)Match 3) Merge 4)Compose

Model Management 2.0: Manipulating Richer Mappings

Why is mapping hard?

- Heterogeneity
- Impedance mismatch

And it is getting harder

- More data models
  - XSD, RDF, OWL
- More programming languages

Two Types of Mappings

- Engineered Mapping:
  - precisely specified.
  - tested for applications.
- Approximate Mapping:
  - imprecision is tolerable
  - there is no well defined notion of correct answer.

Discussion

- This paper talks about two kinds of mapping: engineered mapping and approximate mapping, and after that it focus on engineered mapping. Now can you think about the applications of approximate mapping?? Which applications do you think that just need approximate mapping instead of engineered mapping??

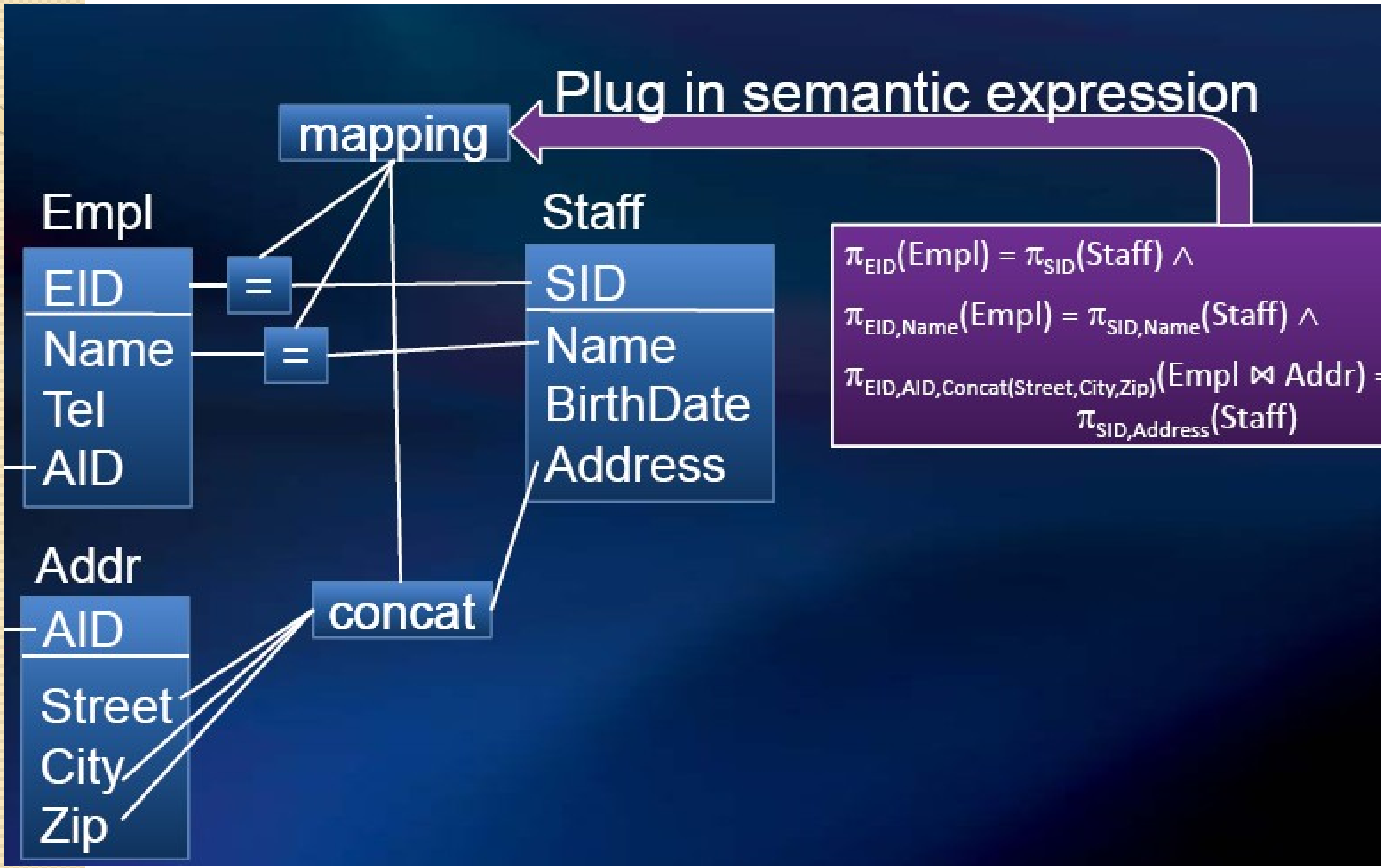
Model Management 1.0

- Research focus on more powerful operations
- Hence better tools

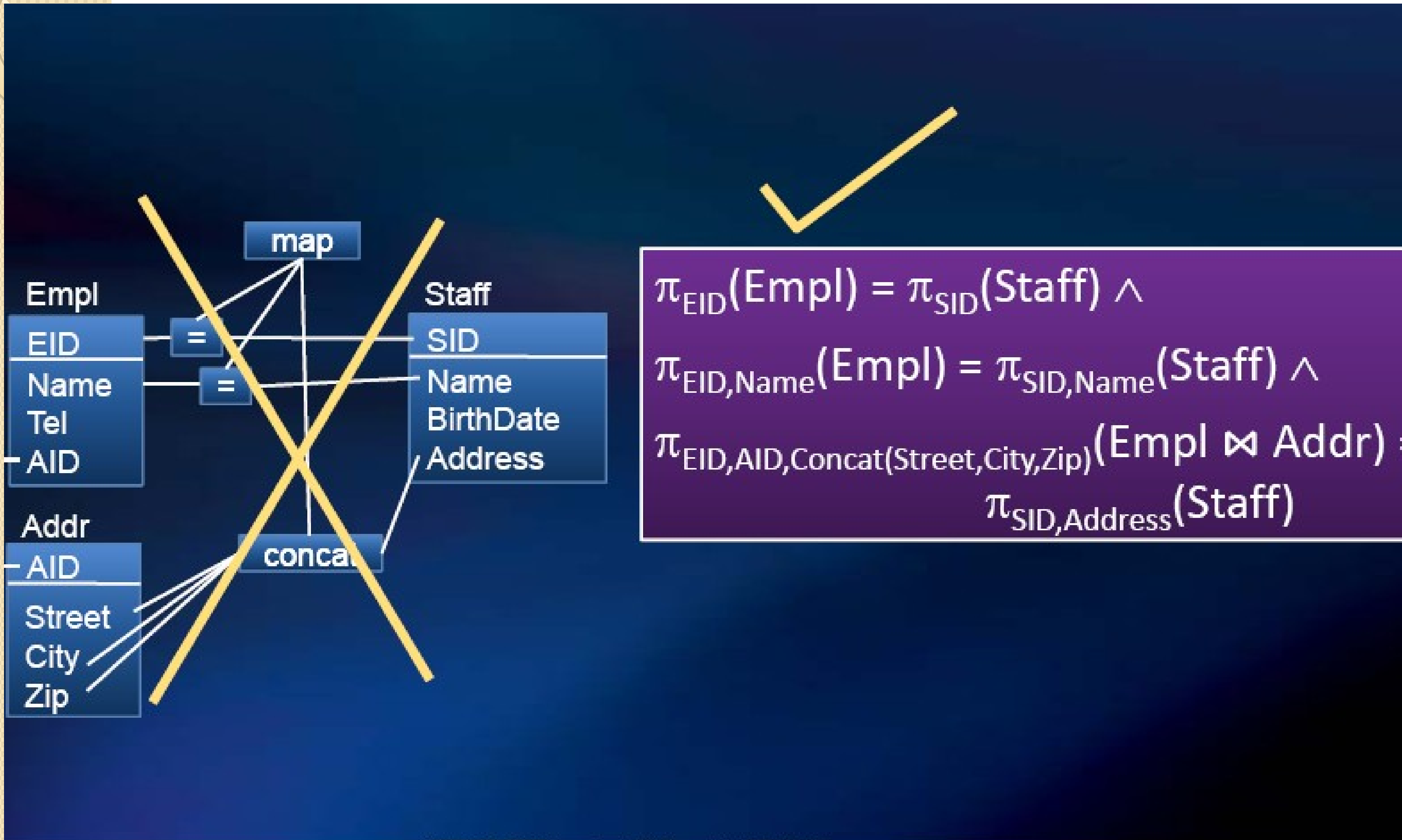
- Good News
  - Lots of progress on operations
  - Some practical applications

- Bad News
  - Still waiting for the first reasonably-complete practical implementation

Version 1.0: Mapping is a structure



Version 2.0: Just use the expression



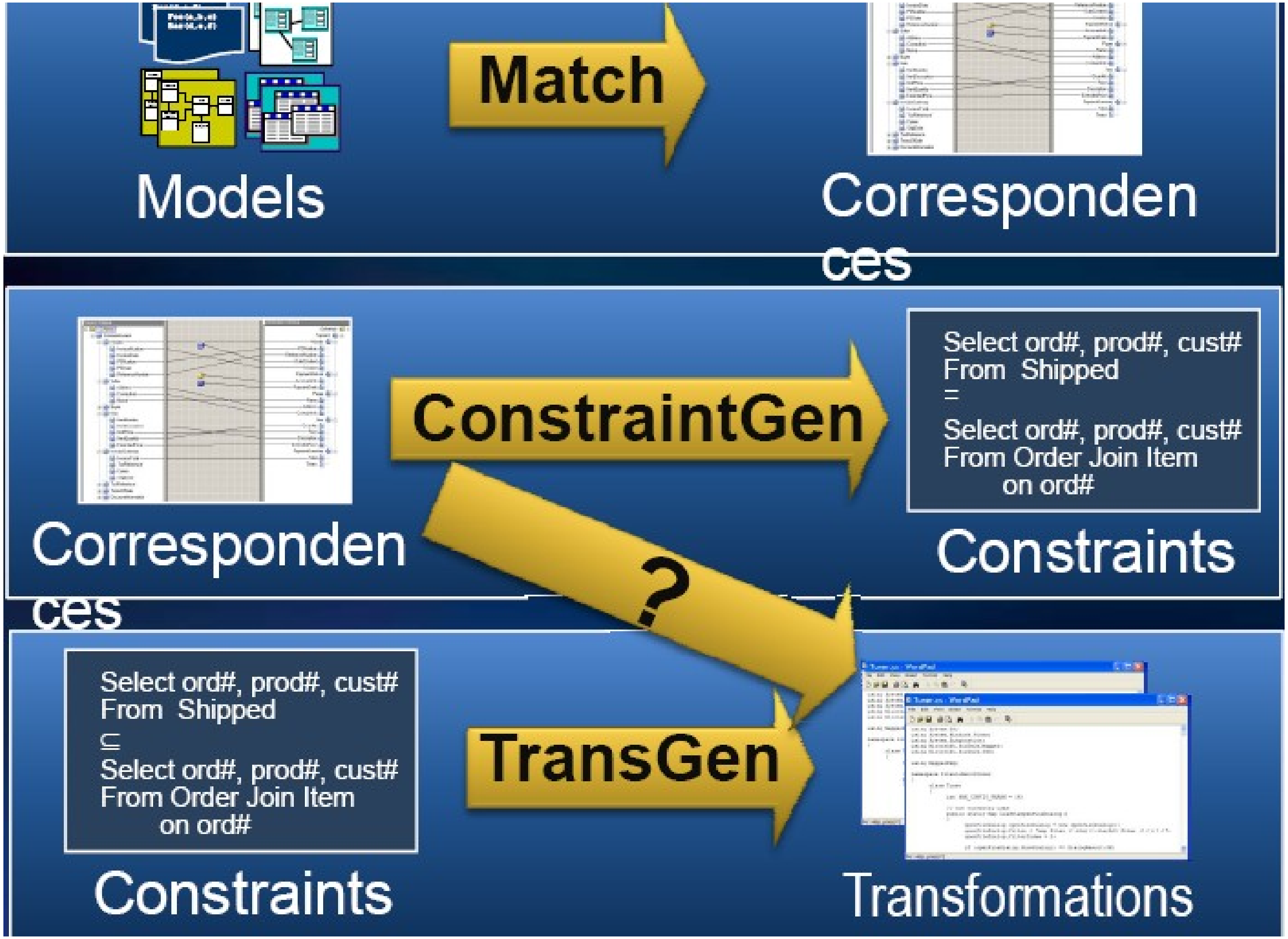
Scenarios

- **Create mappings (Given two schemas, generate a mapping)**
  - Correspondences (schema matching)
  - Mapping constraints (ConstraintGen)
  - Transformations (TransGen)
- **Evolve mappings**

Create Mappings

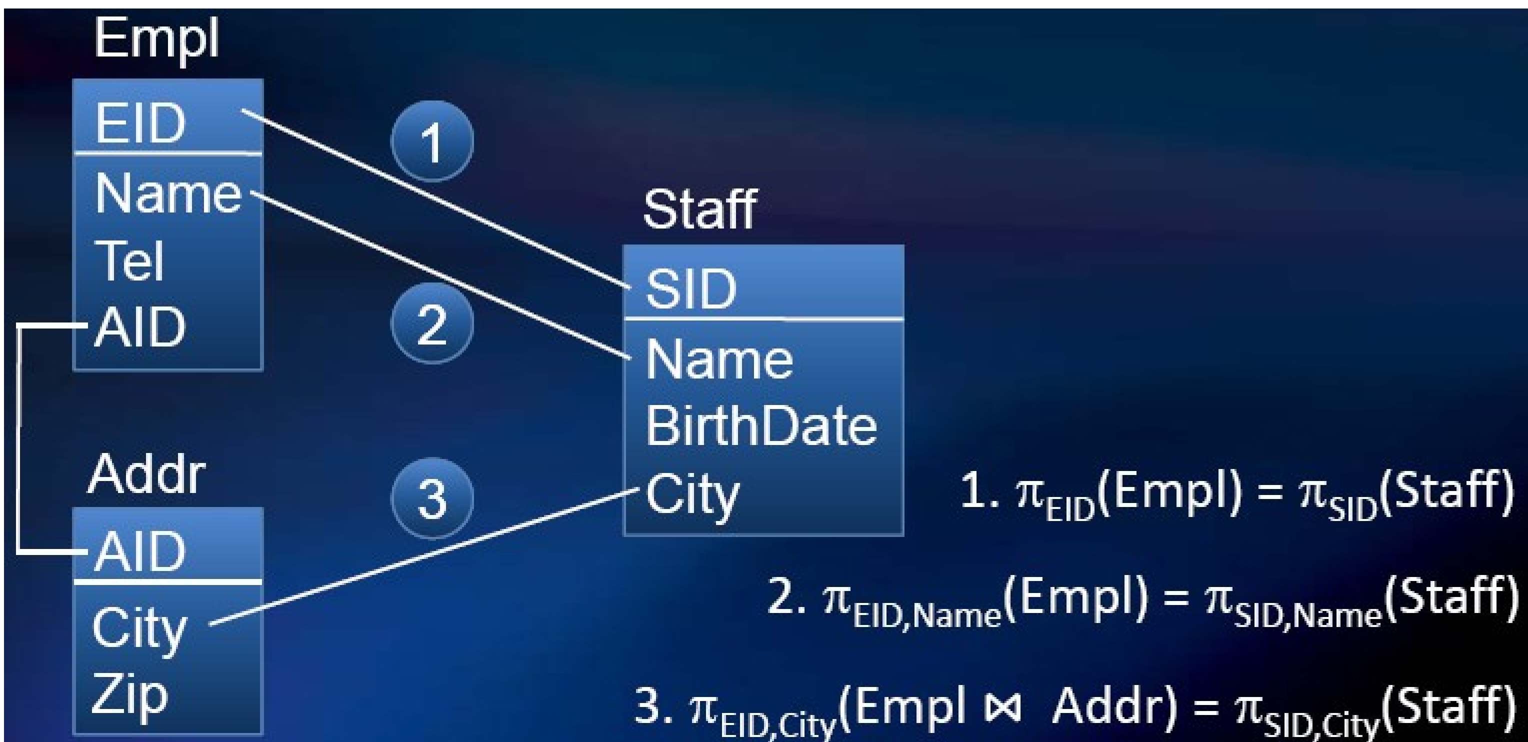
- Three steps:
  - Element correspondences
    - Exploit lexical analysis of element names, data types, previous matching, etc.
  - Mapping constraints relate instances of schemas
    - E.g., equality of relational expressions
  - Transformation is an executable mapping constraint
    - Constructs target instances from source instances
    - E.g., SQL query

Code generation Scenarios

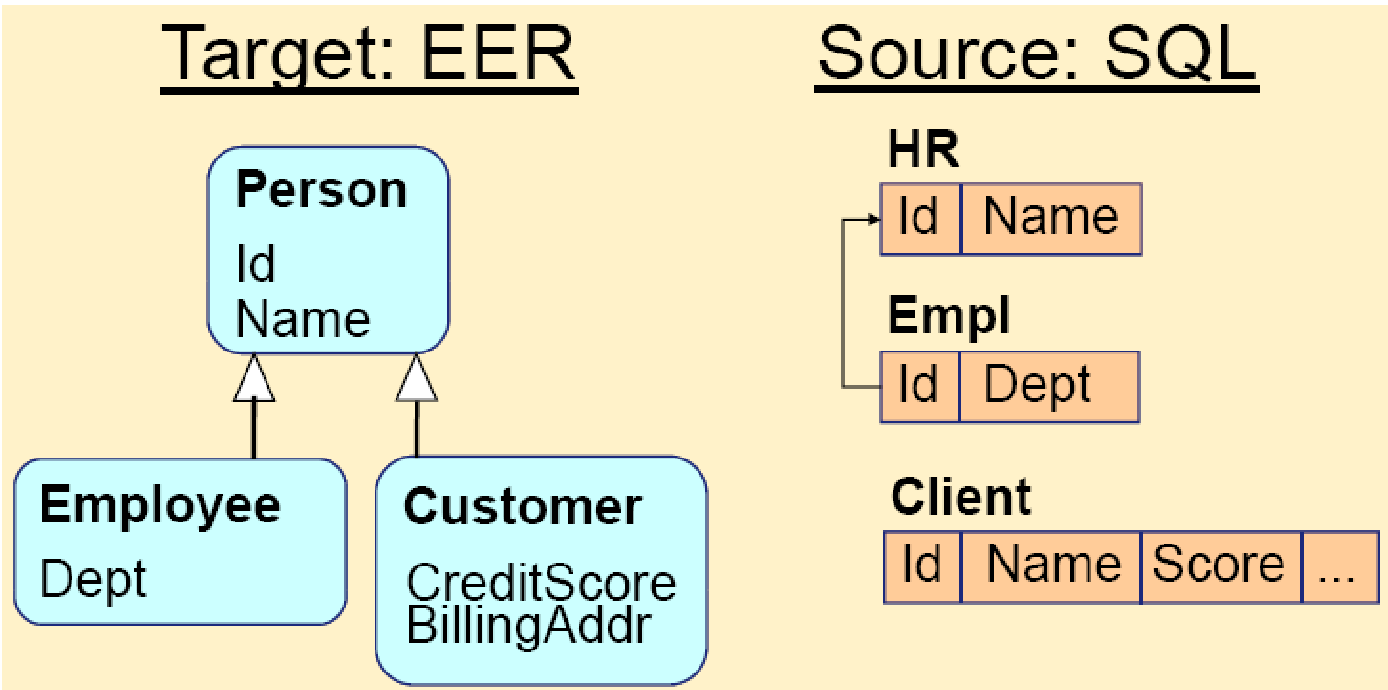


Correspondences → Constraints

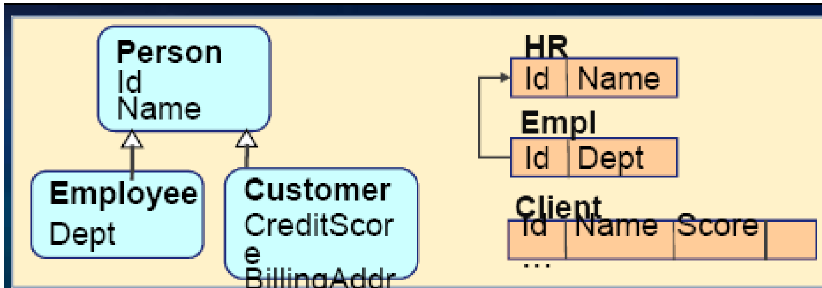
- Directly interpret correspondences as mapping constraints
- If it's a tree schema and keys correspond



Another example



Source: ER target: SQL



Mapping Constraints

```
SELECT p.Id, p.Name
FROM Persons AS p
WHERE p IS OF (ONLY Person)
  OR p IS OF (ONLY Employee)
```

=

```
SELECT Id, Name
FROM dbo.HR
```

```
SELECT e.Id, e.Dept
FROM Persons AS e
WHERE e IS OF Employee
```

=

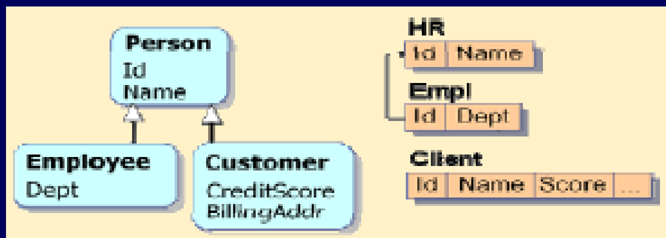
```
SELECT Id, Dept
FROM dbo.Empl
```

```
SELECT c.Id, c.Name,
   c.CreditScore, c.BillingAddr
FROM Persons AS c
WHERE c IS OF Customer
```

=

```
SELECT Id, Name,
         Score, Addr
FROM dbo.Client
```

Constraints → Transformations



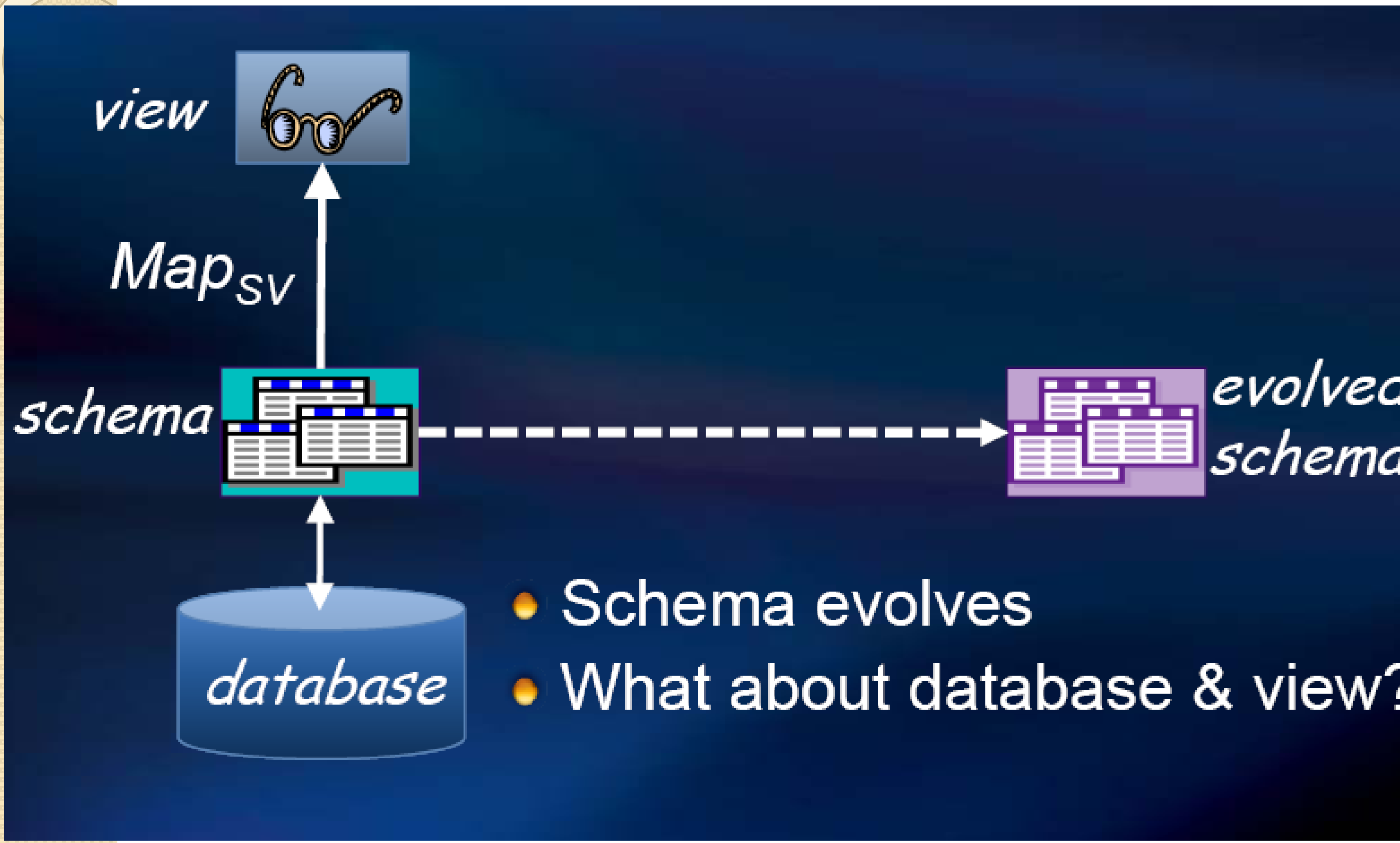

```
SELECT p.Id, p.Name
FROM Persons AS p
WHERE p IS OF (ONLY Person)
  OR p IS OF (ONLY Employee)
```
=
```
SELECT Id, Name
FROM dbo.HR
```

```
SELECT e.Id, e.Dept
FROM Persons AS e
WHERE e IS OF Employee
```
=
```
SELECT Id, Dept
FROM dbo.Empl
```

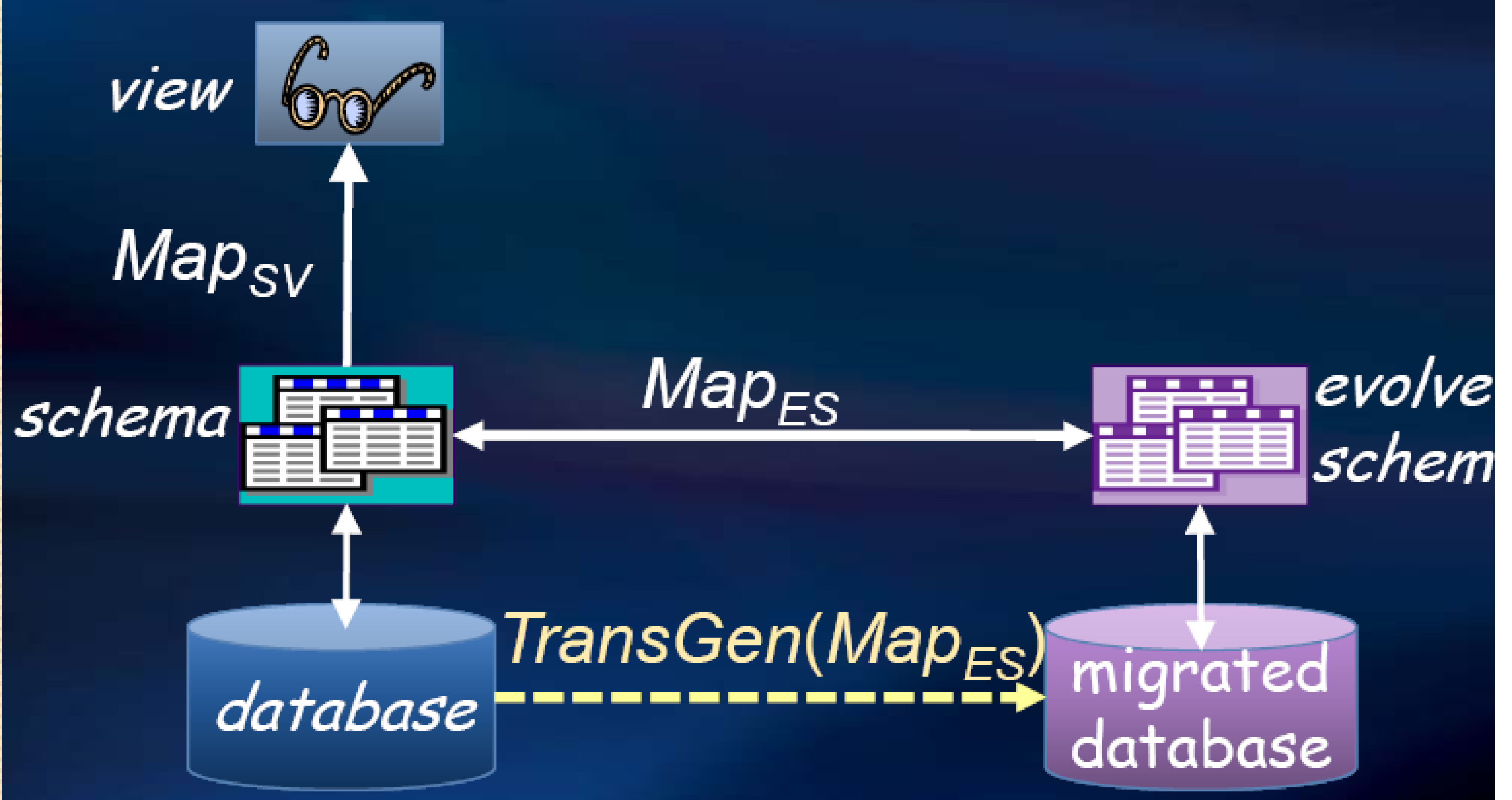
```
SELECT c.Id, c.Name,
  c.CreditScore, c.BillingAddr
FROM Persons AS c
WHERE c IS OF Customer
```
=
```
SELECT Id, Name,
           Score, Addr
FROM dbo.Client
```

```
SELECT VALUE
    CASE
      WHEN (T5._from2 AND NOT(T5._from1))  THEN Person(T5.Person_Id,
T5.Person_Name)
      WHEN (T5._from1 AND T5._from2)
        THEN Employee(T5.Person_Id, T5.Person_Name, T5.Employee_Dept)
        ELSE Customer(T5.Person_Id, T5.Person_Name, T5.Customer_CreditScore,
            T5.Customer_BillingAddr)
    END
  FROM (  (SELECT T1.Person_Id, T1.Person_Name, T2.Employee_Dept,
                 CAST(NULL AS SqlServer.int) AS Customer_CreditScore,
                 CAST(NULL AS SqlServer.nvarchar) AS Customer_BillingAddr, False A
                 _from0,
                 (T2._from1 AND T2._from1 IS NOT NULL) AS _from1, T1._from2
          FROM  ( SELECT  T.Id AS Person_Id, T.Name AS Person_Name, True AS
                 _from2
                 FROM HR AS T) AS T1
                 LEFT OUTER JOIN (
                   SELECT  T.Id AS Person_Id, T.Dept AS Employee_Dept, True AS
                   _from1
                   FROM dbo.Empl AS T) AS T2
                 ON T1.Person_Id = T2.Person_Id )
          UNION ALL (
                 SELECT  T.Id AS Person_Id, T.Name AS Person_Name,
                       CAST(NULL AS SqlServer.nvarchar) AS Employee_Dept,
                       T.Score AS Customer_CreditScore, T.Addr AS
Customer_BillingAddr,
                       True AS _from0, False AS _from1, False AS _from2
                 FROM Client AS T)
        ) AS T5
```

Schema Evolution



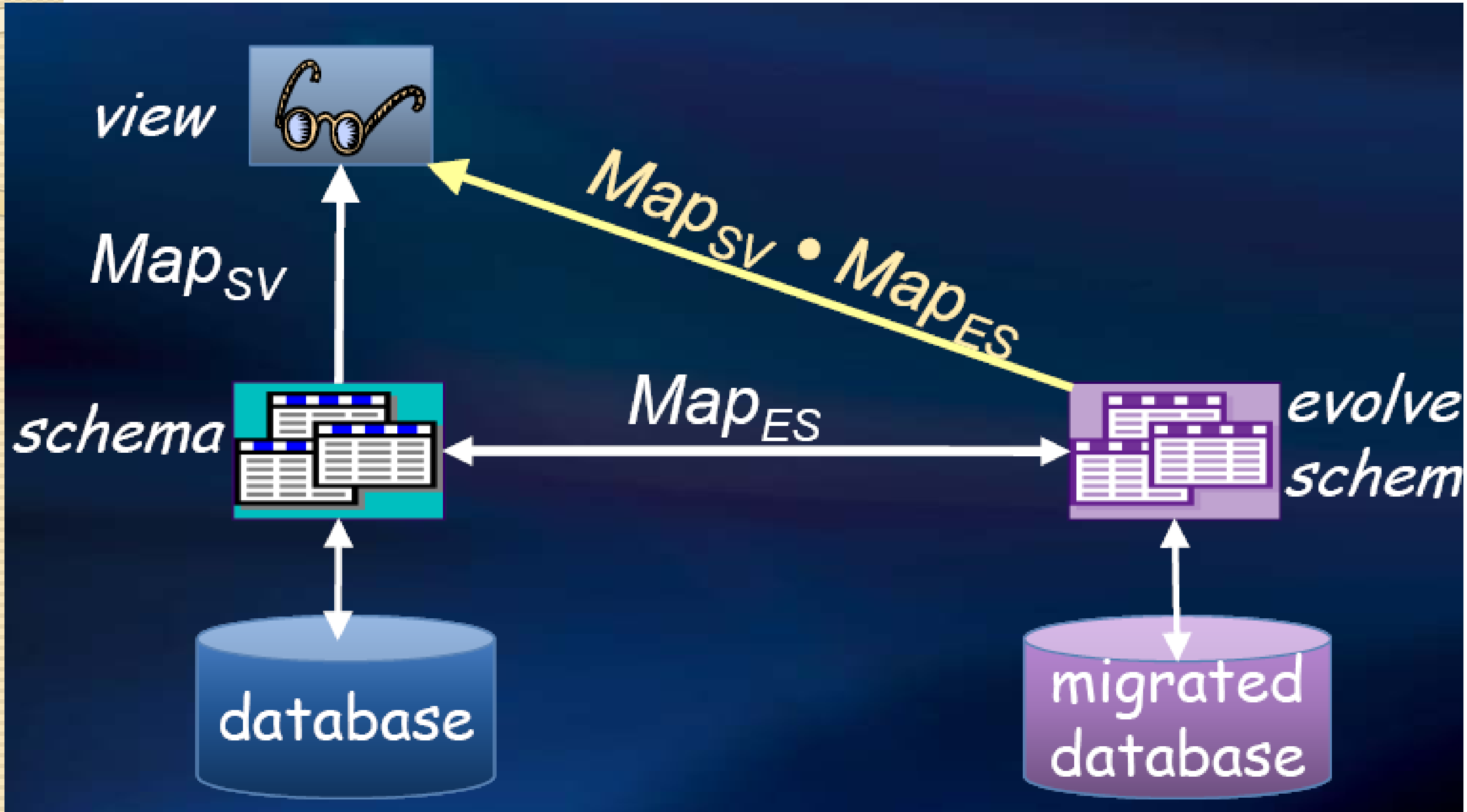
Data Migration



Create mapping: *schema* ⇔ *evolved schema*

Generate a transformation

View Migration



Compose Map_{SV} and Map_{ES} to connect *view* to *evolved schema*

Discussion

- This paper gives a revised vision of MMS compared to the original one. Now which one do you prefer, the first vision or the second one?? Do you think that we need such a complex system or do you still think the original MMS can be useful and applicable??

Thanks