Database Systems Seminar

Senthil Kumar Gurusamy

Papers

Compiling Mappings to Bridge Applications and Databases

- Sergey Melnik, Atul Adya, Philip A. Bernstei

Anatomy of the ADO .NET Entity Framework

- Atul Adya, José A. Blakeley, Sergey Melnik, S. Muralidhar, and the ADO.NET Team

What is ORM??

- A methodology for object oriented systems to hold data in database, with transactional control and yet express it as program objects when needed

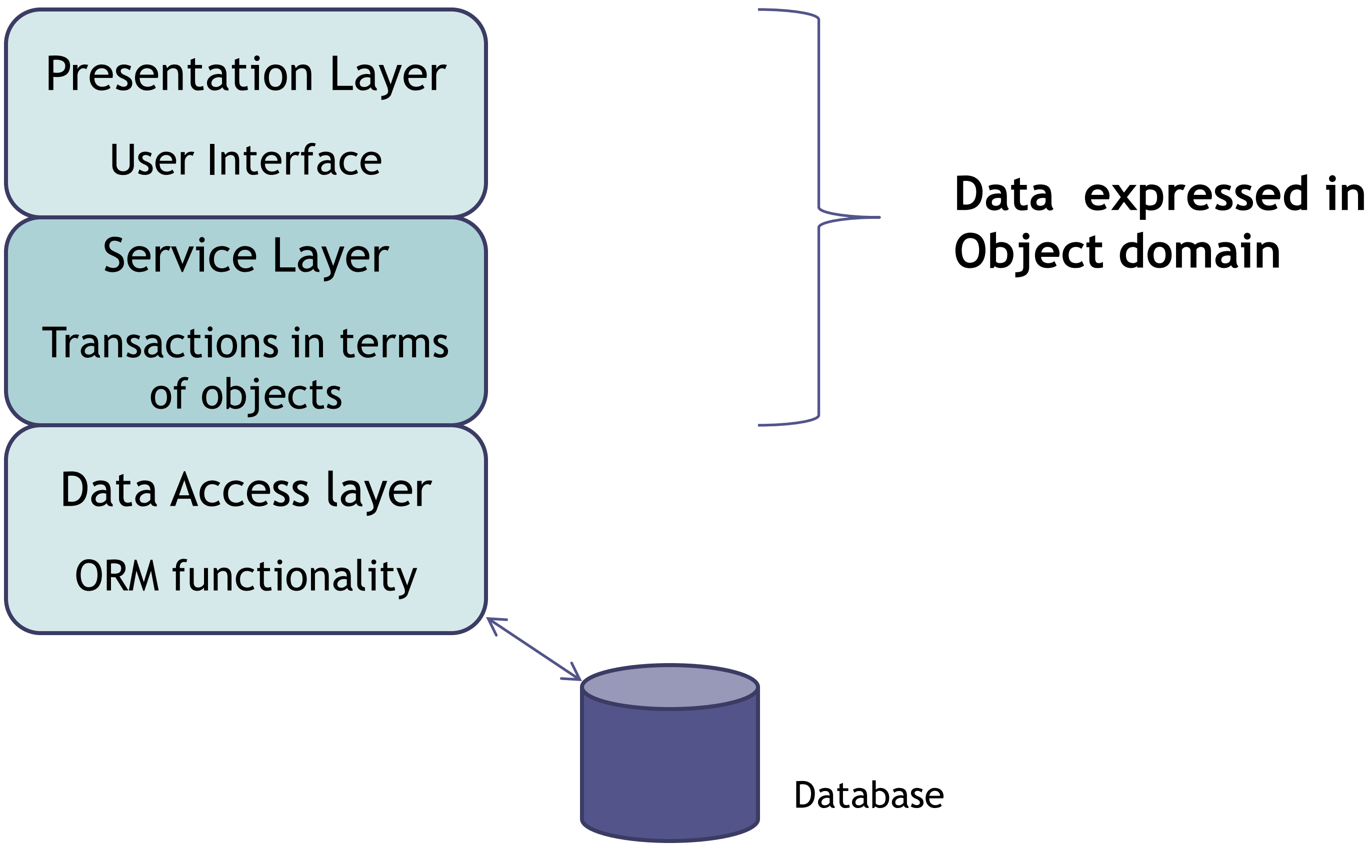
- Avoid bundles of special code

- Essential for multilayered database applications

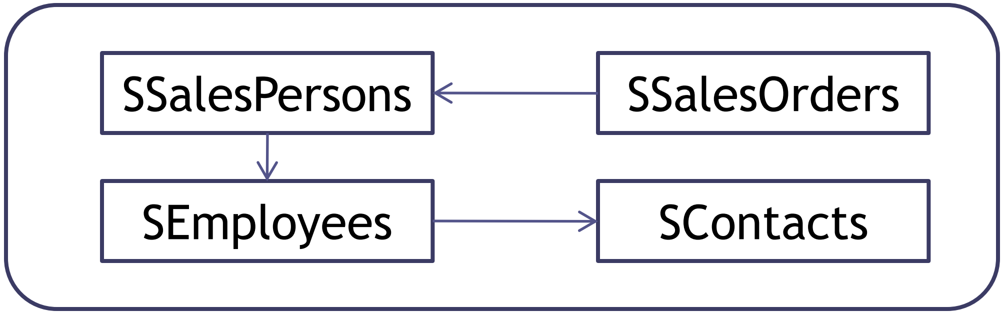
Why ORM ?

- Impedance mismatch between programming language abstractions and persistent storage
- Data independence i.e., data representation can evolve irrespective of the layer
- Independent of DBMS vendor
- Bridge between application and database

Layered Database Application



Sample Relation Schema




```
create table SContacts(ContactId int primary key,
            Name varchar(100),
            Email varchar(100),
            Phone varchar(10));

create table SEmployees( EmployeeId int primary key references SContacts(ContactId),
            Title varchar(20),
            HireDate date);

create table SSalesPersons(SalesPersonId int primary key references
            SEmployees(EmployeeId),
             Bonus int);

create table SSalesOrder(SalesOrderId int primary key,
            SalesPersonId int references SSalesPersons(SalesPersonId));
```

Traditional Embedded Data Access Queries

```
void EmpsByDate(DateTime date) {
using( SqlConnection con = new SqlConnection (CONN_STRING) ) {
    con.Open();
    SqlCommand cmd = con.CreateCommand();
        cmd.CommandText = @"
    SELECT SalesPersonID, FirstName, HireDate
    FROM SSalesPersons sp
        INNER JOIN SEmployees e
        ON sp.SalesPersonID = e.EmployeeID
        INNER JOIN SContacts c
        ON e.EmployeeID = c.ContactID
    WHERE e.HireDate < @date";
    cmd.Parameters.AddWithValue("@date",date);
    DbDataReader r = cmd.ExecuteReader();
    while(r.Read()) {
        Console.WriteLine("{0:d}:\t{1}", r["HireDate"],
        r["FirstName"]);
} } }
```

Entity SQL

```
void EmpsByDate (DateTime date) {
  using( EntityConnection con =
    new EntityConnection (CONN_STRING) ) {
      con.Open();
      EntityCommand cmd = con.CreateCommand();
       cmd.CommandText = @"
             SELECT VALUE sp FROM ESalesPersons sp
             WHERE sp.HireDate < @date";
      cmd.Parameters.AddWithValue ("@date", date);
      DbDataReader r = cmd.ExecuteReader();
      while (r.Read()) {
             Console.WriteLine("{0:d}:\t{1}", r["HireDate"], r["FirstName"])
} } }
```

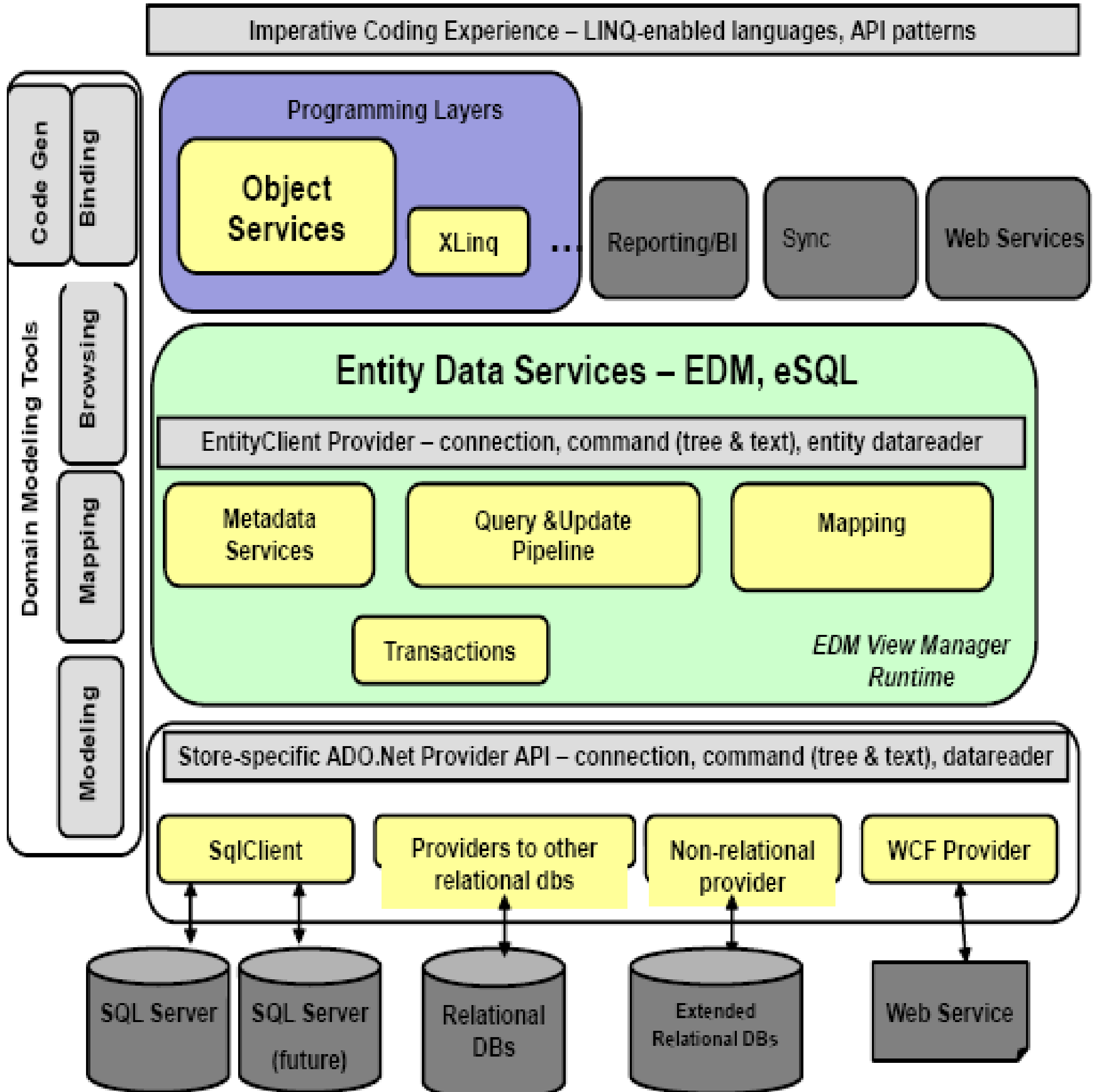
LINQ

```
void EmpsByDate(DateTime date) {
   using (AdventureWorksDB aw =
     new AdventureWorksDB()) {
     var people = from p in aw.SalesPersons
                          where p.HireDate < date
                           select p;
     foreach (SalesPerson p in people) {
          Console.WriteLine("{0:d}\t{1}", p.HireDate,
                    p.FirstName );
} } }
```

O/R mismatch - Improvements

- 1980s: Persistent programming languages
  - One or two commercial products
- 1990s: OODBMS
  - No widespread acceptance
- "Objects & Databases: A Decade in Turmoil"
  - Carey & DeWitt (VLDB'96), bet on ORDBMS
- 2000: ORDBMS go mainstream
  - DB2 & Oracle implement hardwired O/R mapping
  - O/R features rarely used for business data
- 2002: client-side data mapping layers
- Today: ORM Frameworks – ADO .NET EDM Framework, hibernate, JPA, Toplink, etc.

ADO .NET Entity Framework Architecture



Components of the Framework

- Data Source providers
  -Provides data to EDM Layer services from data sources
  -Support for different types of sources
- Entity Data Services
  -EDM
  -Metadata services
- Programming Layers
- Domain Modeling Tools
  -tools for schema generation, creating mapping fragments

Object Services

- .NET CLR
  -Common Language runtime
  - allows any program in .NET language to interact with Entity Framework
- Database connection, metadata
- Object State Manager
  -Tracks in-memory changes
  - construct the change list input to the processing infrastructure
- Object materializer
  - Transformations during query and update views between entity values from the conceptual layer and corresponding CLR Objects

Interacting with Data in EDM Framework

- Entity SQL
  - Derived from standard SQL
  - with capabilities to manipulate EDM instances

- LINQ
  -Language-integrated query
  - Expressions of the programming language itself
  -Supported in MS programming languages(VB, C#)

•CRUD
  - Create, Read, Update and Delete operations on objects

Domain modeling Tools

Some of the design time tools included in the framework

- Model designer
  -Used to define the conceptual model interactively
  - generate and consume model descriptions
  - Synthesize EDM models from relational metadata

- Mapping Designer
  - conceptual model to the relational database map
  -This map is the input to the mapping compilation which generates the query and update views

- Code generation
  - Set of tools to generate CLR classes for the entity types

Query Pipeline

- Breaks down Entity SQL or LINQ query into one or more elementary, relational-only queries that can be evaluated by the underlying data store

Steps in query Processing

- Syntax & Semantic analysis
  - Parsed, analyzed using Metadata services component
- Conversion to a canonical Command Tree
  - Converted to Optimized tree
- Mapping view Unfolding
  - Translated to reference the underlying db tables

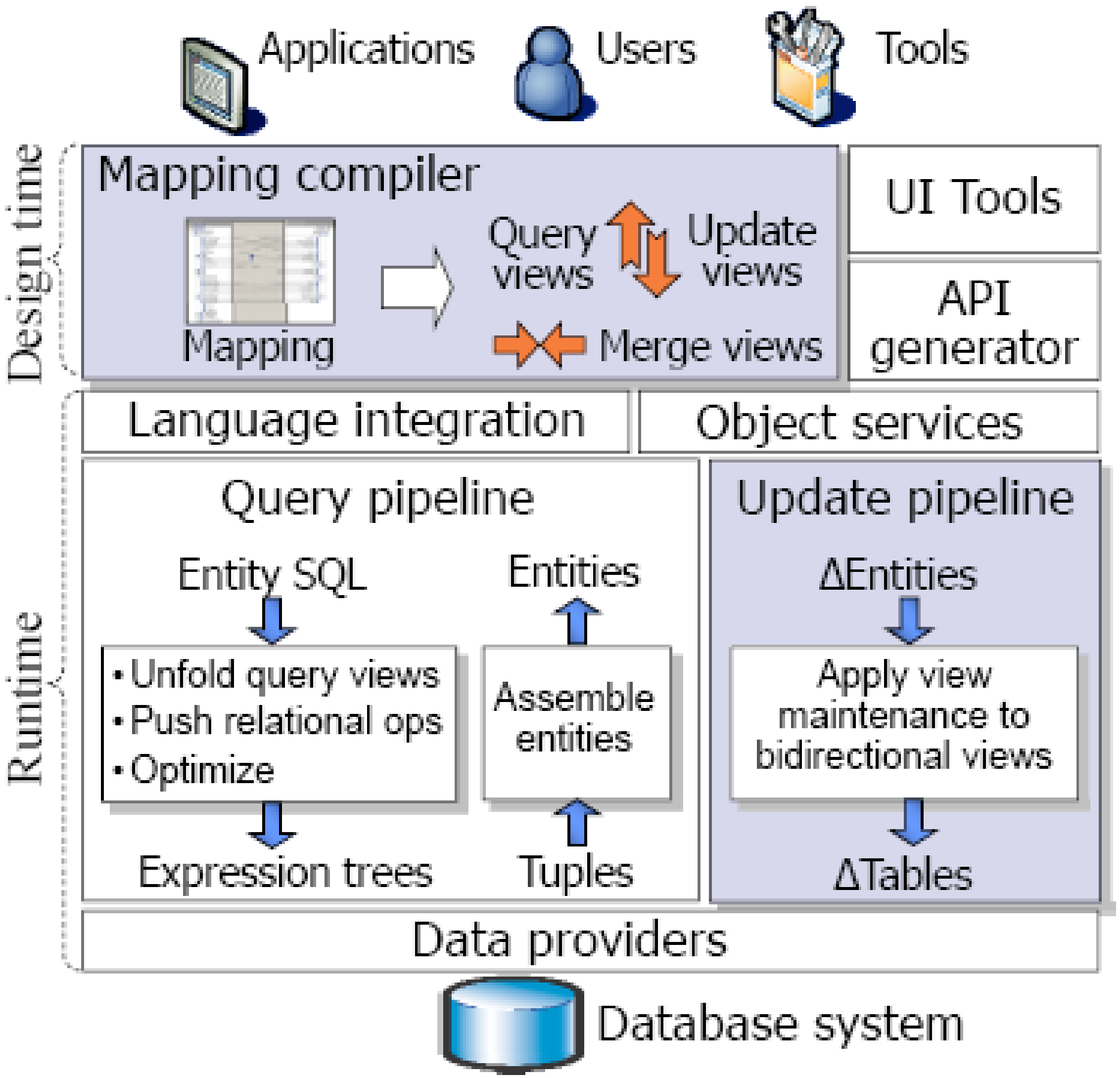
Steps Contd.

- Structured Type Elimination
  - References to structured data(ancestor, constructors)
- Projection Pruning
  - Elimination of unreferenced expressions
- Nest Pull-up
  - Nested query is bubbled to the top
- Transformations
  - Redundant operations are eliminated by pushing down other operators
- Translation to Provider Specific Commands
- Command Execution
- Result Assembly
- Object Materializaton
  - Results are materialized into appropriate programming language objects

Special Features of the Framework

- Allows higher level of abstraction than relational model
- Leverages on the .NET data provider model
- Allows data centric services like reporting on top of the conceptual model
- Together with LINQ reduces impedance mismatch significantly

System Architecture



Bidirectional views

- Mappings relate entities with relations
- Mappings together with the database are compiled into views
- Drives the runtime engine
- Speeds up mapping translation
- Updates on view are enforced using update translation techniques

Bidirectional View Generation

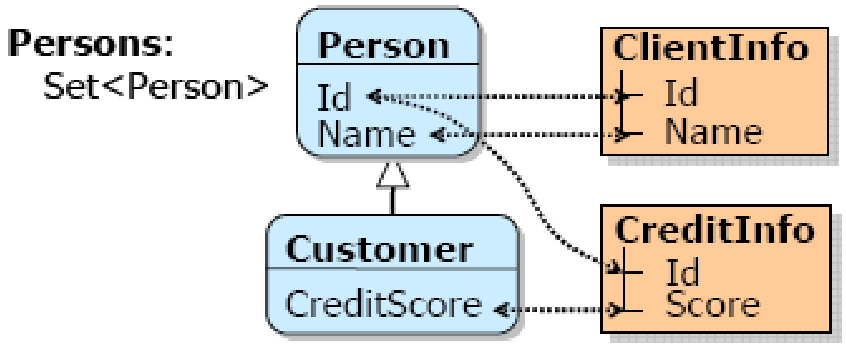
- Query View
  - Express entities in terms of tables
- Update Views
  - Express tables in terms of entities

Entities = QueryViews(Tables)
Tables = UpdateViews(Entities)

Entities = QueryViews(UpdateViews(Entities))

This ensures entity can be persisted and re-asssembled from db in a lossless manner

Compiler Mapping




```
SELECT p.Id, p.Name
FROM Persons p
```
=
```
SELECT Id, Name
FROM ClientInfo
```

```
SELECT c.Id, c.CreditScore
FROM Persons c
WHERE c IS OF Customer
```
=
```
SELECT Id, Score
FROM CreditInfo
```

- Mapping is specified using a set of mapping fragments
- Each fragment is of the form $Q_{Entities} = Q_{Tables}$

Query & Update views

To reassemble Persons from relational tables

Query view

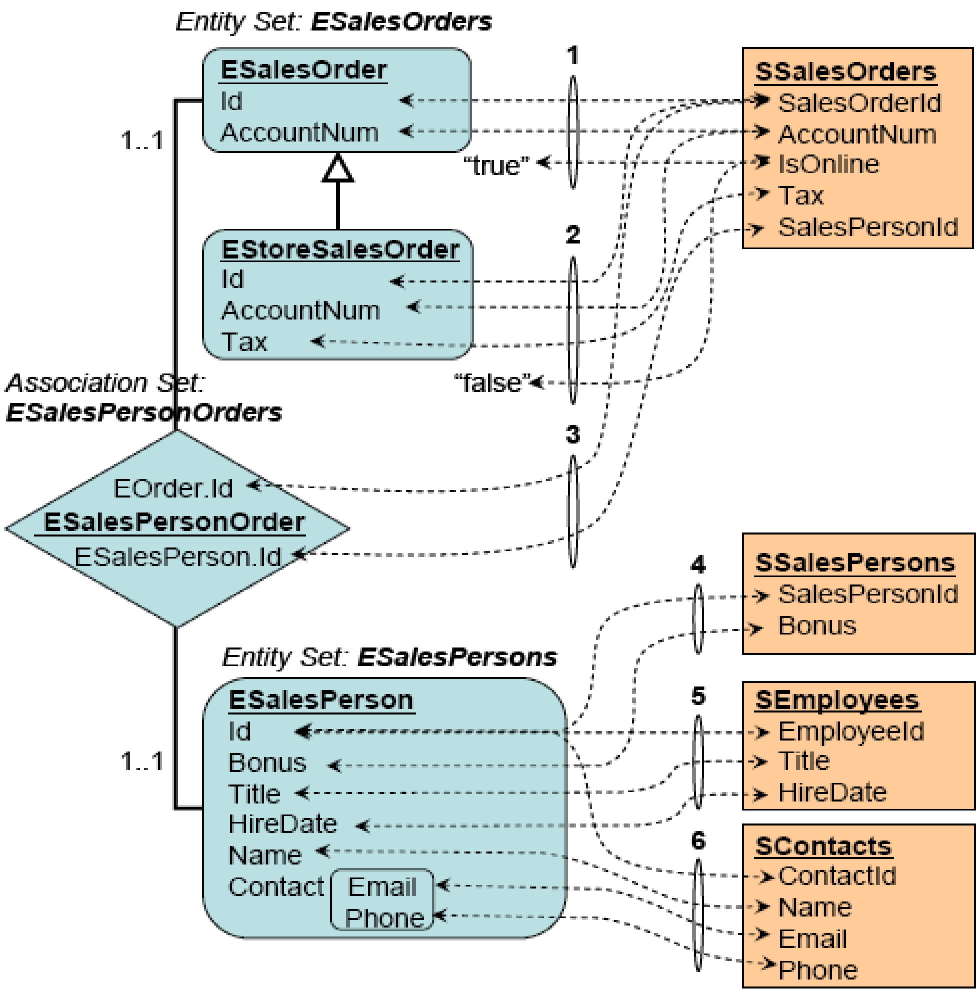
```
Persons =
 SELECT
       CASE  WHEN T2.from2
                 THEN Customer(T1.Id, T1.Name, T2.CreditScore)
                 ELSE Person(T1.Id, T1.Name)  END
 FROM  ClientInfo AS T1
 LEFT OUTER JOIN (
          SELECT Id, Score AS CreditScore,
                   True AS from2
          FROM CreditInfo) AS T2
ON T1.Id = T2.Id
```

Update views

```
ClientInfo = SELECT p.Id, p.Name
                  FROM Persons p
```

```
CreditInfo = SELECT c.Id, c.CreditScore
                  FROM Persons c
                  WHERE c IS OF Customer
```

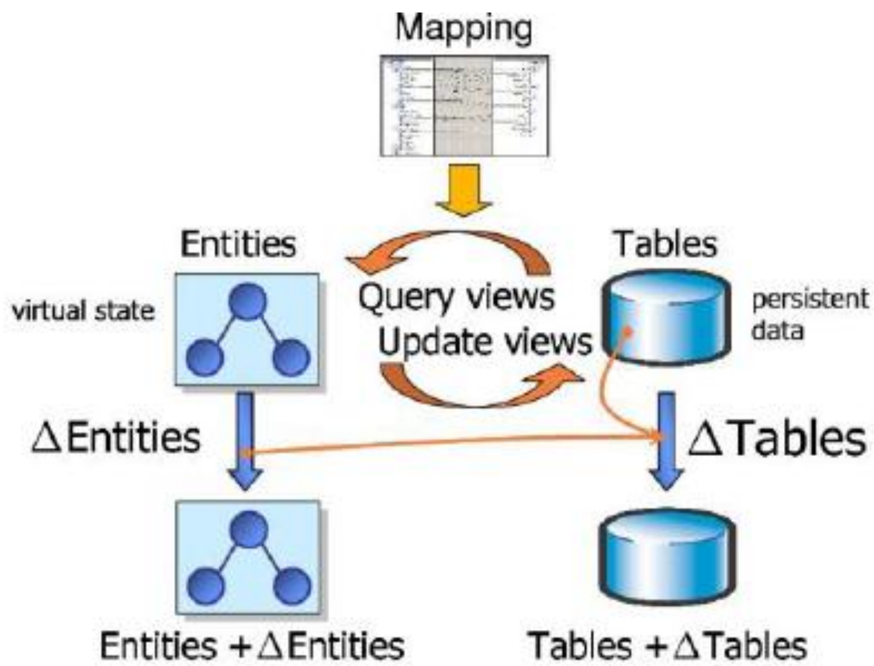
Specification of Mappings - Schema



Specification of Mappings - Mappings

SELECT o.Id, o.AccountNum FROM **ESalesOrders** o WHERE o IS OF (ONLY ESalesOrder)	=	SELECT SalesOrderId, AccountNum FROM **SSalesOrders** WHERE IsOnline = "true"
SELECT o.Id, o.AccountNum, o.Tax FROM **ESalesOrders** o WHERE o IS OF EStoreSalesOrder	=	SELECT SalesOrderId, AccountNum, Tax FROM **SSalesOrders** WHERE IsOnline = "false"
SELECT o.EOrder.Id, o.ESalesPerson.Id FROM **ESalesPersonOrders o**	=	SELECT SalesOrderId, SalesPersonId FROM **SSalesOrders**
SELECT p.Id, p.Bonus FROM **ESalesPersons** p	=	SELECT SalesPersonId, Bonus FROM **SSalesPersons**
SELECT p.Id, p.Title, p.HireDate FROM **ESalesPersons** p	=	SELECT EmployeeId, Title, HireDate FROM **SEmployees**
SELECT p.Id, p.Name, p.Contact.Email, p.Contact.Phone FROM **ESalesPersons** p	=	SELECT ContactId, Name, Email, Phone FROM **SContacts**

Update Translation



1. View maintenance:
   ΔTables = ΔUpdateViews(Entities, ΔEntities)
2. View Unfolding:
   ΔTables = ΔUpdateViews(QueryViews(Tables), ΔEntities)

Steps in Update Translation:

- Change list Generation
  -List of changes per entity set is created
  - Represented as lists of deleted and inserted elements
- Value Expression Propagation
  - Transforms the list of changes obtained from view maintenance into sequence of algebraic base table insert and delete expressions against the underlying affected tables

Steps in Update Translation(cont’d):

- Stored Procedure Calls Generation
  -Produces the final sequence SQL statements on relational schema (INSERT, DELETE, UPDATE)

- Cache Synchronization
  - After updates, the cache state is synchronized with the new db state

Update translation Example – Update query

```
using(AdventureWorksDB aw = new AdventureWorksDB()) {
 // People hired more than 5 years ago
var people = from p in aw.SalesPeople
                 where p.HireDate <
                 DateTime.Today.AddYears(-5) select p;
foreach(SalesPerson p in people) {
          if(HRWebService.ReadyForPromotion(p)) {
                   p.Bonus += 10;
                   p.Title = "Senior Sales Representative";
} }
aw.SaveChanges();
}
```

Update Translation – Value Expressions

ΔSSalesPersons= SELECT p.Id, p.Bonus
FROM ΔESalesPersons As p

ΔSemployees = SELECT p.Id, p.Title
FROM ΔESalesPersons AS p

ΔSContacts = SELECT p.Id, p.Name, p.Contact.Email,
p.Contact.Phone FROM ΔESalesPersons AS p

BEGIN TRANSACTION
UPDATE [dbo].[SSalesPersons] SET [Bonus]=30
WHERE [SalesPersonID]=1
UPDATE [dbo].[SSEmployees] SET [Title]= N'Senior Sales
Representative'
WHERE [EmployeeID]=1
END TRANSACTION

Mapping Compilation problem

• Improper proper specification of Mapping fragments will lead to the mapping not satisfying the Data Round-tripping Criterions

$$map \circ map^{-1} = Id(C)$$

•Application developers cannot be entrusted with task of checking for Data round-tripping criterion

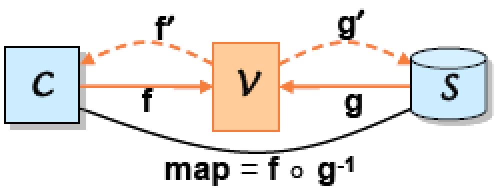
• Hence Mapping Compilation has to done by EDM model

Bipartite Mappings

Mapping fragments are defined as follows:

$\Sigma_{map} = \{ Q_{c1} = Q_{s1}, .. , Q_{cn} = Q_{sn} \}$

where Q_c is the query over the client schema and Q_s is the query over store schema



Thus, $\Sigma_{map} = f \circ g'$

Where the view f: $C \rightarrow V$

view g: $S \rightarrow V$

View Generation & Mapping Compilation

1. Subdivide the mapping into independent set of fragments
2. Perform mapping validation by checking the condition Range(f) $\subseteq$ Range(g)
3. Partition the entity set based on mapping constraints
4. Compile the relevant mappings on each partition
5. Regroup the generated views
6. Eliminate unnecessary self joins

Paritioning Scheme

```
procedure PartitionVertically(p, Tp,map)
   Part := Ø // start with an empty set of partitions
   for each type T that is derived from or equal to Tp do
      P := {σp IS OF (ONLY T)}
      for each direct or inherited member A of T do
         if map contains a condition on p.A then
            if p.A is of primitive type then
                  P := P × Dom(p.A, map)
            else if p.A is of complex type TA then
                  P := P × PartitionVertically(p.A, TA,map)
         end if
      end for
      Part := Part ∪ P
   end for
return Part
```

Role of Dom(p, map)

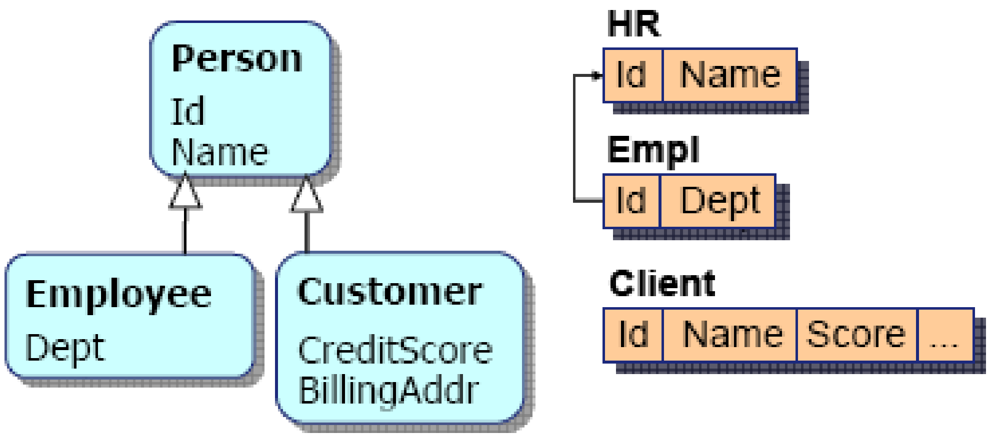
Suppose the mapping constraints contain conditions,

(p=1) and (p IS NOT NULL) on path p of type integer

$cond_1$:= (p=1)
$cond_2$:= (p IS NULL)
$cond_3$:= NOT (p=1 OR p IS NULL)

Every pair of conditions in Dom(p, map) is mutually exclusive conditions

Partitioning Example



Above schema and BillingAddr is nullable property with complex type Address. Type Address has subtype USAddress

P_1 : σ_e IS OF (ONLY Person)
P_2 : σ_e IS OF (ONLY Customer) AND e.BillingAddr IS NULL
P_3 : σ_e IS OF (ONLY Customer) AND e.BillingAddr IS OF (ONLY Address)
P_4 : σ_e IS OF (ONLY Customer) AND e.BillingAddr IS OF (ONLY USAddress)
P_5 : σ_e IS OF (ONLY Employee)

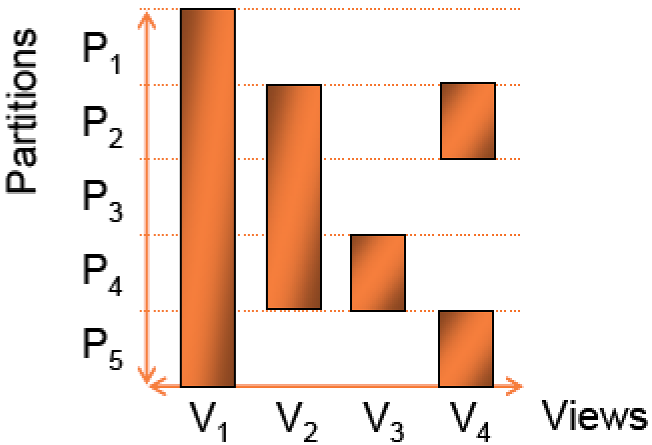
Reconstructing partitions from views

<u>**procedure**</u> RecoverPartitions($\mathbf{P}_{\text{exp}}, \mathbf{P}, \mathbf{V}$)
  Sort $\mathbf{V}$ by increasing number $|V|$ of partitions per view and
      by decreasing number $|Attrs(V)|$ of attributes per view
  <u>**for**</u> <u>**each**</u> partition $P \in \mathbf{P}_{\text{exp}}$ <u>**do**</u>
    $Pos := \emptyset;\ Neg := \emptyset;$ // keeps intersected & subtracted views
    $Att := Attrs(P);$ // attributes still missing
    $PT := \mathbf{P};$ // keeps partitions disambiguated so far
    // Phase 1: intersect
    <u>**for**</u> ($i = 1;\ i \leq n$ <u>**and**</u> $|PT| > 1$ <u>**and**</u> $|Att| > 0;\ i$++) <u>**do**</u>
      <u>**if**</u> $P \in V_i$ <u>**then**</u>
        $Pos := Pos \cup V_i;\ PT := PT \cap V_i$
        $Att := Att - Attrs(V_i)$
      <u>**end**</u> <u>**if**</u>
    <u>**end**</u> <u>**for**</u>

Reconstructing partitions from views

```
// Phase 2: subtract
for (i = n; i ≥ 1 and |PT| > 1; i--) do
  if P ∉ V_i then
    Neg := Neg ∪ V_i; PT := PT ∩ V_i
  end if
end for
if |PT| = 1 and |Att| = 0 then
  Recovered[P] := (Pos, Neg)
end if
end for
return
```

Reconstruction Example



$$V_1 = \pi(P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$$
$$V_2 = \pi(P_2 \cup P_3 \cup P_4)$$
$$V_3 = \pi(P_4)$$
$$V_4 = \pi(P_2 \cup P_5)$$

$$P_1 = (V_1)\ \overline{\ltimes}\ (V_2 \cup V_4)$$
$$P_2 = (V_4 \bowtie V_2 \bowtie V_1)$$
$$P_3 = (V_2 \bowtie V_1)\ \overline{\ltimes}\ (V_4 \cup V_3)$$
$$P_4 = (V_3 \bowtie V_1)$$
$$P_5 = (V_4 \bowtie V_1)\ \overline{\ltimes}\ (V_2)$$

Grouping Partitioned views

The entire entity set is obtained by grouping views using $\cup_a$, $\bowtie$, ⟕

$$
\begin{aligned}
E &= P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5 \\
&= V_1 \text{ ⟗ } V_2 \text{ ⟗ } V_3 \text{ ⟗ } V_4 \\
&= (V_1 \text{ ⟕ } (V_2 \text{ ⟕ } V_3)) \text{ ⟕ } V_4 \\
&= ((V_1 \text{ ⟕ } V_2) \text{ ⟕ } (V_3 \cup^a V_4))
\end{aligned}
$$

$\cup^a$ - denotes union without duplicate elimination

Evaluation

Experimental evaluation of the Entity framework was done focusing on mapping compiler for the following parameters

Correctness:

Using automated suite, thousands of mappings was generated by varying some objects. The compiled views are verified by deploying the entire data access stack to query and update sample databases.

Evaluation (cont’d)

Efficiency:

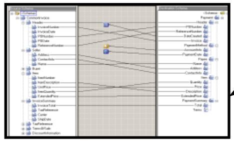
- Compiling the independent mapping fragments on partitions alone takes exponential time.
- Recovering partitions from views takes O(n log n)
- All other steps take O(n) time
- The number of independent fragments were less
- So, the few second delay at start time and restarts was acceptable

Evaluation (contd)

Performance:

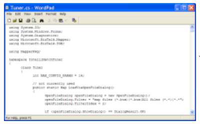
- Mapping compilation anchors both client-side rewriting and server-side execution
- Implied constraints were used fully to generate simplified views
- Major overheads: object instantiation, caching, query manipulations and delta computation for updates
- These overheads dominated only for small datasets

Contributions



Mapping

compile



Bidirectional views

- Declarative mapping language
  - -Allows non-expert users to specify complex O/R mappings
  - -Formal semantics

- Mapping compilation
  - - Guarantees correctness

- Mechanism for updatable views
  - - Large class of updates, not O/R specific
  - - Leverages view maintenance technology

QUESTIONS ????

THANK YOU