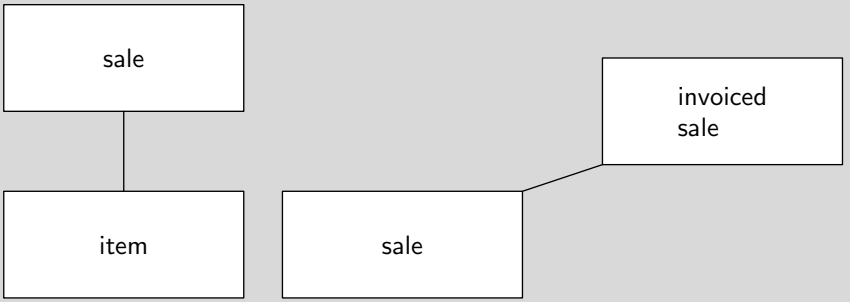
Applying Semantic Principles in a Relational Environment

Xplain Concepts

Xplain supports two types of relationships: aggregation and generalization.



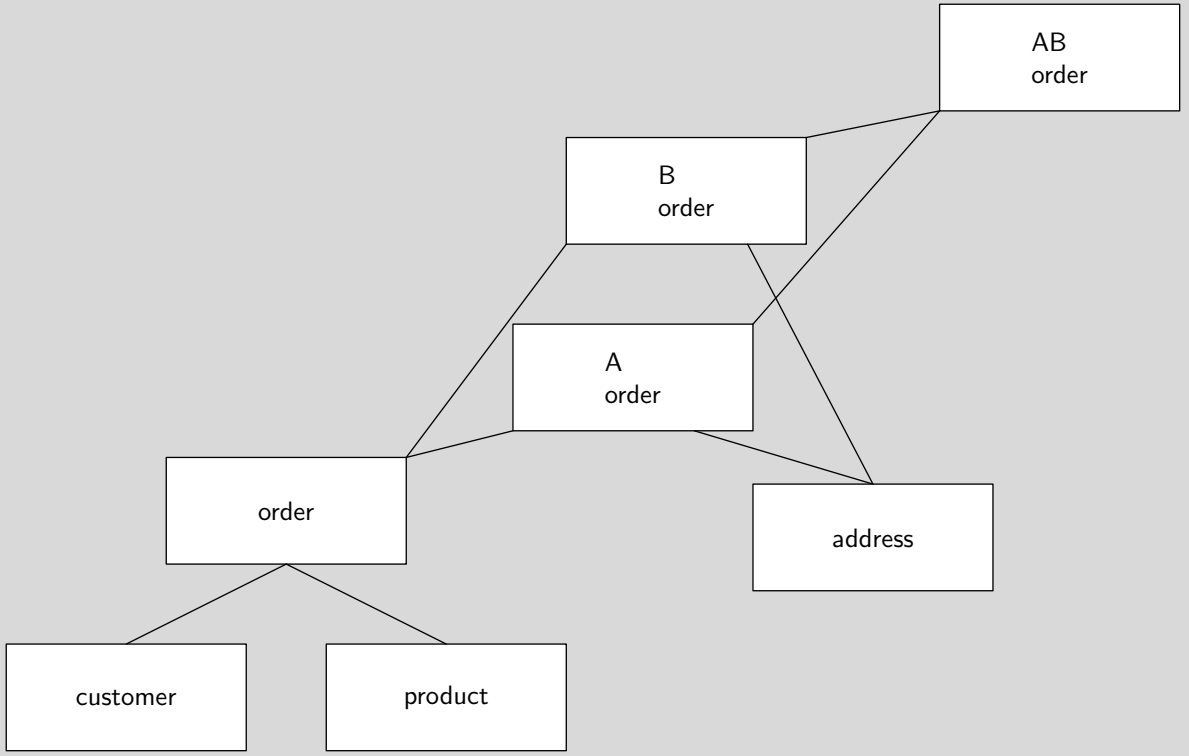
aggregation generalization

Some remarks about the schematic language:

- A rectangle signifies a type.
- Aggregation is represented by a line connecting the centers of two facing rectangles.
- Generalization is represented by a line connecting facing corners.

Xplain Concepts
Xplain's advantages
Conversion of `base`
Conversion of `type`
Conversion of constraints
Conversion of `extend`
Conclusion
close

Larger schematic example



Xplain Concepts
Xplain's advantages
Conversion of base
Conversion of type
Conversion of constraints
Conversion of extend
Conclusion
close

Data definition statements

Xplain supports base types and types.

```
base price (R4,2).
base week (I2) (1..52).

type item (A4) = description, stock, price.
type sale (A4) = week, day, item, number, amount.

init default sale its number = 1.
```

Xplain Concepts
Xplain's advantages
Conversion of base
Conversion of type
Conversion of constraints
Conversion of extend
Conclusion
close

Data modification statements

Xplain supports the familiar insert, update and delete statements.

```
insert sale "1" its
  week = 10, day = "Mon", item = "I51",
  amount = "49.90".

update item "I51" its
  price = "51.50".

delete item
  where stock = 0.
```

Xplain Concepts
Xplain's advantages
Conversion of base
Conversion of type
Conversion of constraints
Conversion of extend
Conclusion
close

Data retrieval statements

Data is retrieved with `get`, the `extend` command is unique for Xplain. With `extend` an additional, temporary column is added to a table.

```
extend item with turnover =
  total sale its amount
  per item.

get item its turnover.
```

Xplain's advantages

- Good fit in OO environments: supports same relationships.
- Xplain diagrams are easy to read.
- Xplain is type based, not set based.
- Xplain's language is orthogonal: there's just one way to express a thing.
- Xplain solutions can be reused: no single large query that attempts to everything.

With `xplain2sql`, the Xplain language can be used in relational environments.

`xplain2sql`, written in Eiffel, is available for any platform that supports Standard C.

Conversion of `base`

- Output domains, if domains are enabled and the target SQL implementation supports domains.
- Else, the data representation is remembered and used in `create table` statements.

```
base number (I4).
base day (A3) ("Mon","Tue","Wed","Thu","Fri","Sat").
```

```
create domain Tnumber as smallint;

create domain Tday as character(3) not null
  check (value in ('Mon','Tue','Wed',
                   'Thu','Fri','Sat'));
```

Conversion of `type`

Xplain code:

```
type item (A4) = description, stock, price.
```

InterBase SQL code:

```
create table item (
  id_item character(4) not null primary key,
  description Tdescription not null,
  stock Tstock not null,
  price Tprice not null);
```

Conversion of constraints

Three levels of constraints:

- Inherent constraints: enforced, except exclusive specialization and convertibility.
- Static constraints: not supported, i.e. the `assert` statement. `assert` is hard, database should have before commit transaction triggers.
- Dynamic constraints: `init` supported, `check` not yet supported.

Conversion of `init`

Converting `init` constraints is hard. Ideally:

- The target SQL implementation supports the ANSI SQL `default` constraint, which most do.
- And it should have before-insert triggers.
- And after-insert triggers.

show Xplain code

Xplain Concepts
Xplain's advantages
Conversion of `base`
Conversion of `type`
Conversion of constraints
Conversion of `extend`
Conclusion
close

Conversion of `extend`

Adding a temporary column to a table requires:

- The SQL implementation should have the notion of a temporary table.
- The SQL implementation should support subqueries.
- The SQL implementation should have a function like coalesce (ANSI-92 standard).

Many SQL implementations do not fulfill these requirements.

Conversion of `extend`: example

Microsoft SQL 7 code:

```
select
  [id_item]
    (coalesce(select sum([sale].[amount])
       from [sale]
       where
         [sale].[id_item] = [access_type].[id_item]
    ), 0)) as [turnover]
  into [#item.turnover]
  from [item] access_type

select [item].[id_item], [turnover]
  from [item]
  join [#item.turnover] on
    [#item.turnover].[id_item] = [item].[id_item]
```

show Xplain code

Conclusion

`xplain2sql` converts most Xplain statements to equivalent SQL, if the target SQL implementation fulfills certain requirements.

In the near future `xplain2sql` will be extended to support the `assert` and `check` statements.

Moreover, we want to bring Xplain style of modeling to XML by generating XML schema's from an Xplain data model.

Xplain Concepts
Xplain's advantages
Conversion of `base`
Conversion of `type`
Conversion of constraints
Conversion of `extend`
Conclusion
close