2IS55 Software Evolution

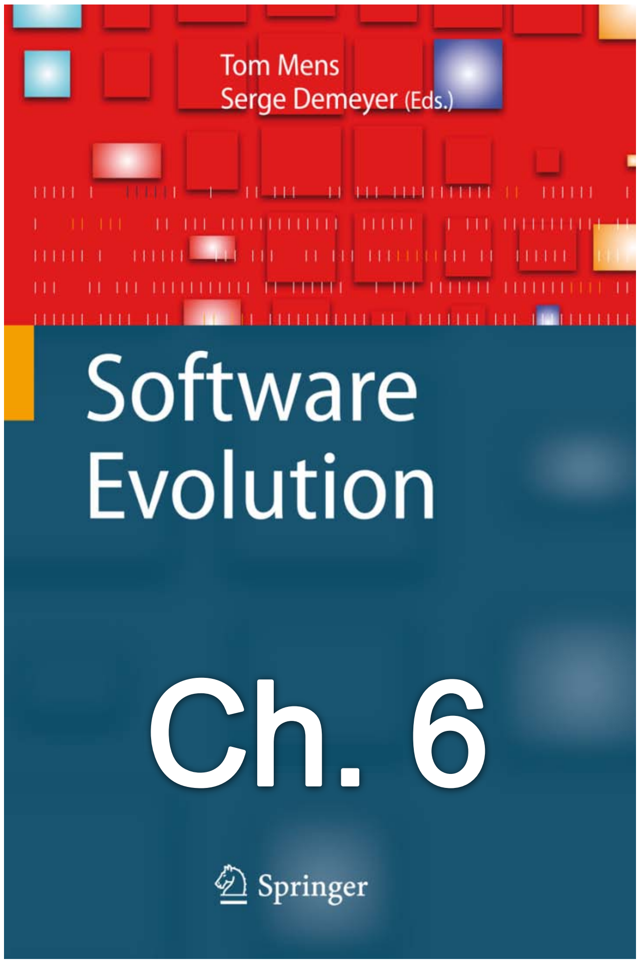
Implementing evolution: Database migration

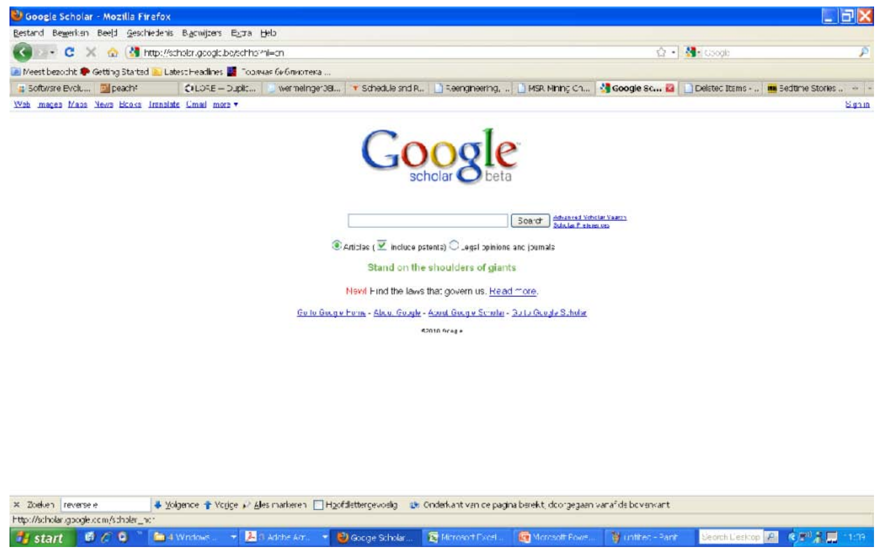
Alexander Serebrenik

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Sources





Technische Universiteit Eindhoven University of Technology

Last week

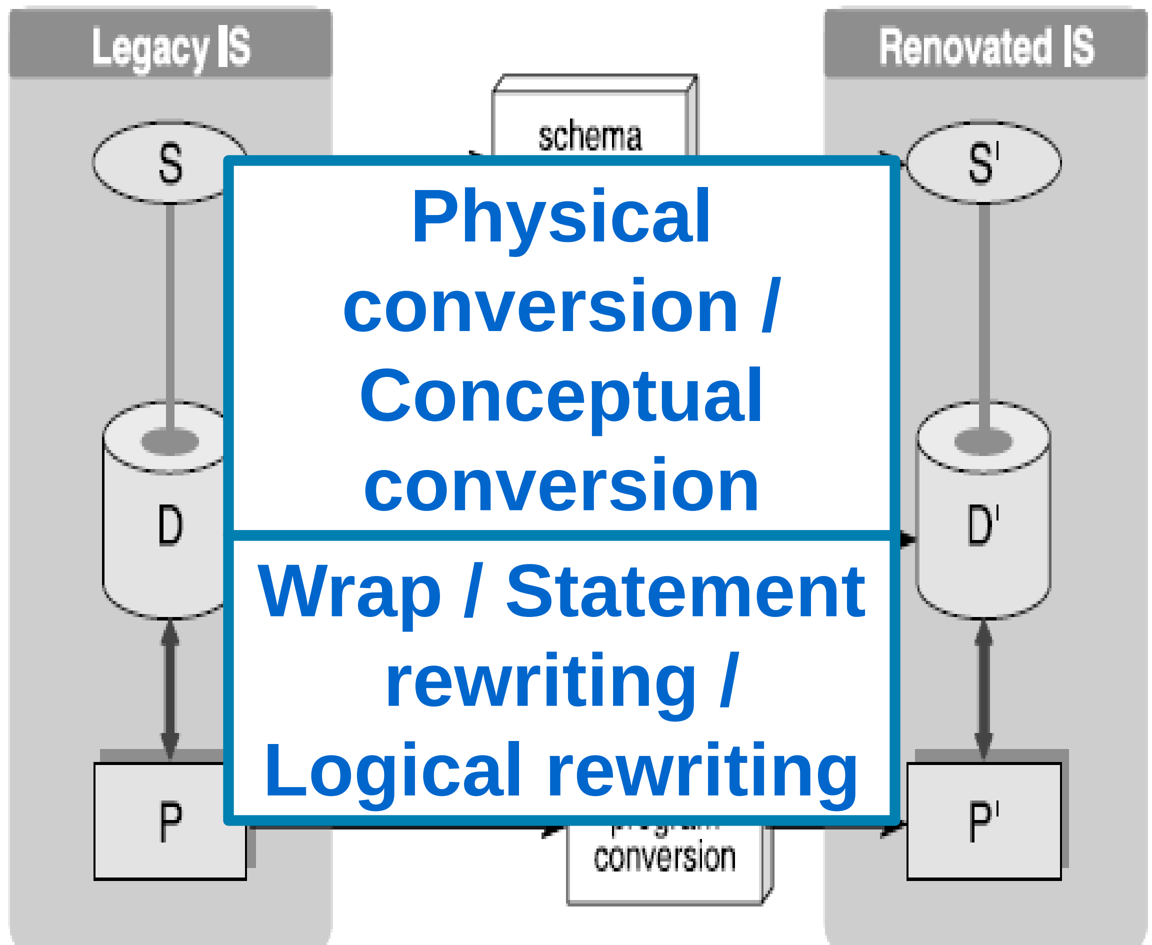
- **Assignment 7**
  - **Deadline: Today!**

- **Assignment 8**
  - **Individual**
  - **Opens today**
  - **Deadline: June 22**

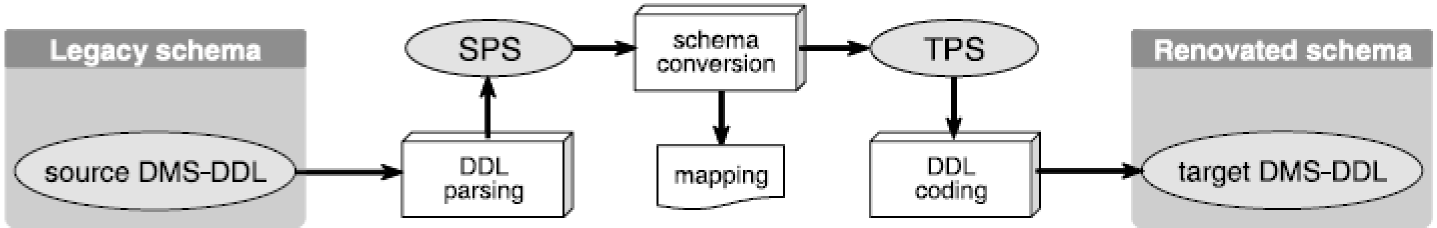
Recapitulation

- **Last week: introduction to DB migration**

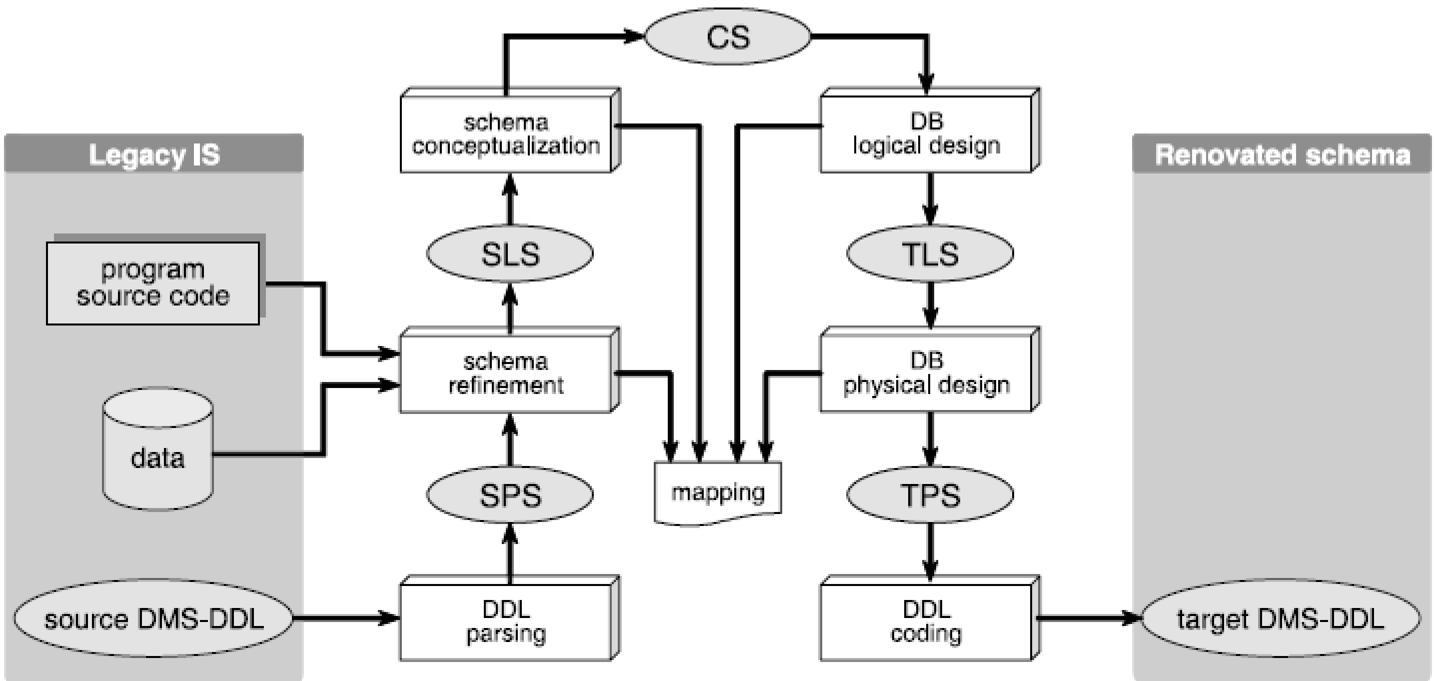
- **DB migration**
  - **Data is important, technology is outdated**
  - **S – DB schema**
  - **D – DB data**
  - **P – data manipulation programs**



Schema conversion: Physical vs Conceptual

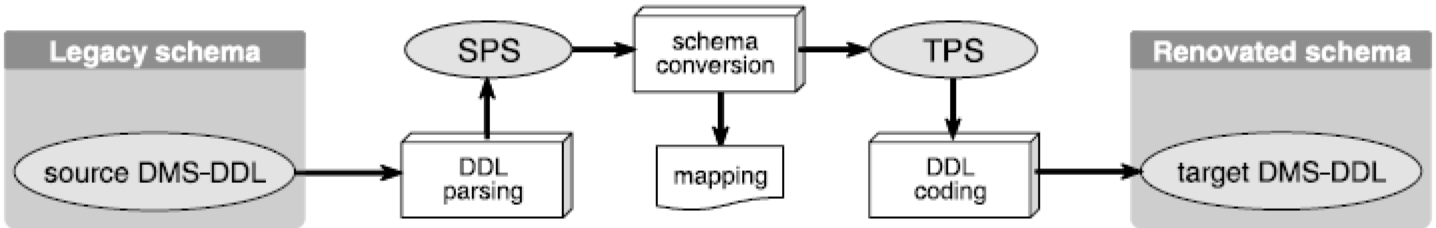


Physical



Conceptual

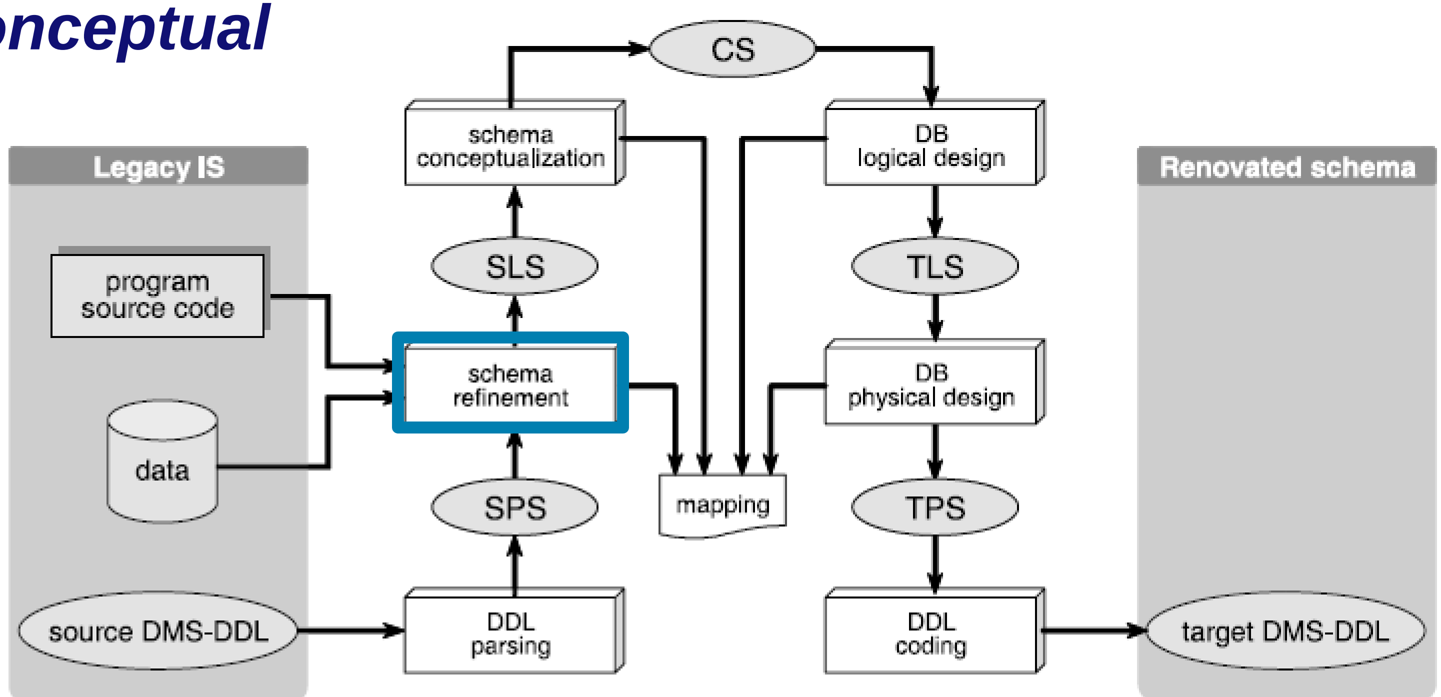
Schema conversion: Physical



- **Easy to automate**
  - **Existing work: COBOL ⇒ relational, hierarchical ⇒ relational, relational ⇒ OO**
  - **“Migration as translation” vs “migration as improvement”**
- **Semantics is ignored**
  - **Limitations of COBOL ⇒ Design decisions in the legacy system ⇒ Automatic conversion ⇒ the same design decisions in the new system**
    - **Risk: compromised flexibility**

Schema conversion: Physical vs Conceptual

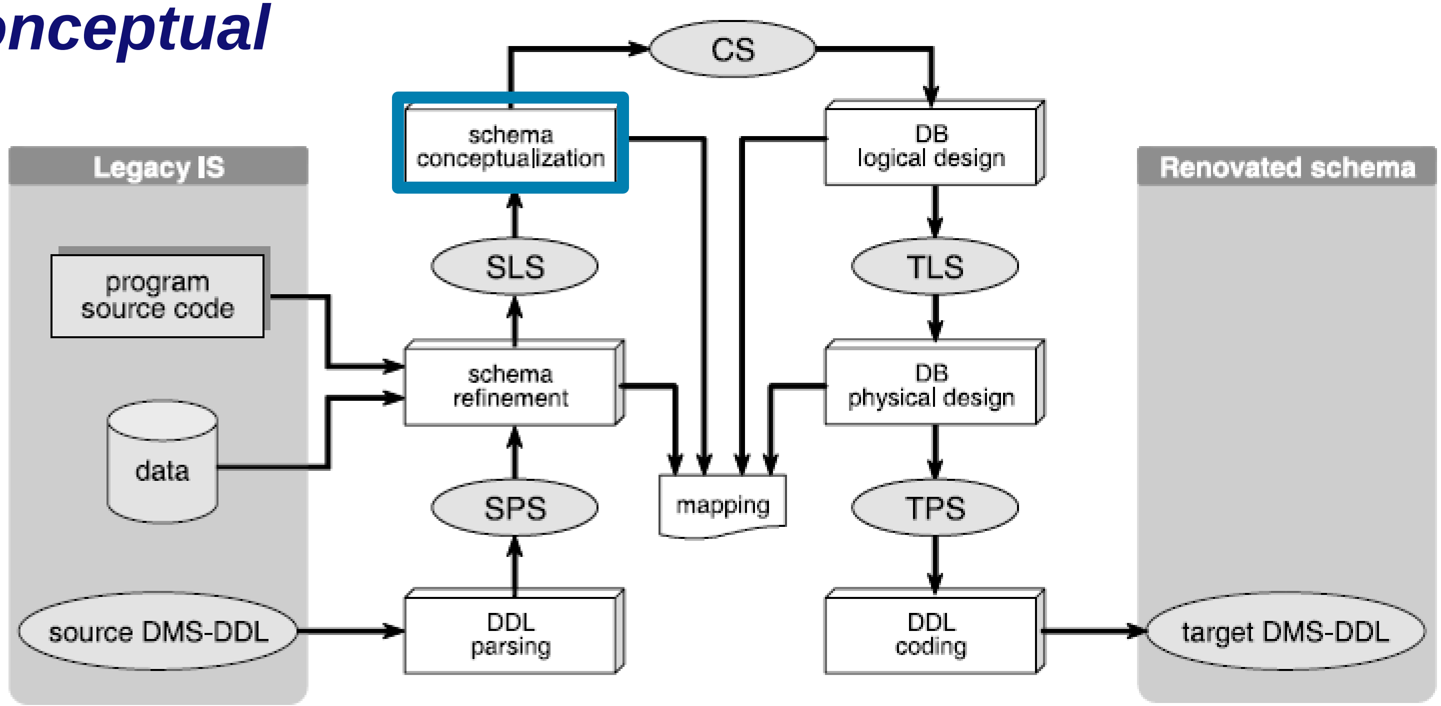
Conceptual



- **Refinement: Data and code may contain implicit constraints (field refinement, foreign key, cardinality) on the schema**

Schema conceptualization

Conceptual

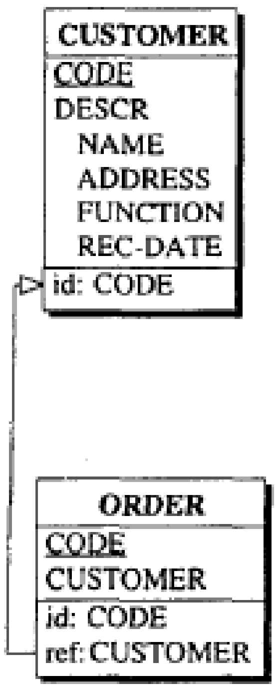


- **Conceptualization: Remove implementation details**

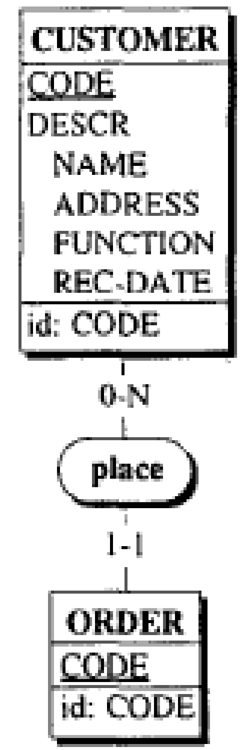
Conceptualization

- Preparation: “clean up” to understand
  - e.g., rename attributes, drop one-element compounds
- Untranslation: separate logic from limitations of technology
- De-optimization: separate logic from performance
- Conceptual normalization:
  - Entities vs. relations and attributes
  - Explicit IS-A relations

Untranslation: Foreign keys



- COBOL allows “direct access” via foreign keys
- ER requires a relationship set to connect two entities
- What would be the appropriate cardinality?
  - One customer can place multiple orders
  - Every order can be placed only by one customer



Technische Universiteit Eindhoven University of Technology

De-optimization

- **Recall:**

```
FD ORDER.
01 ORD.
  02 ORD-CODE PIC 9(10).
  02 ORD-CUSTOMER PIC X(12).
  02 ORD-DETAIL PIC X(200).
```

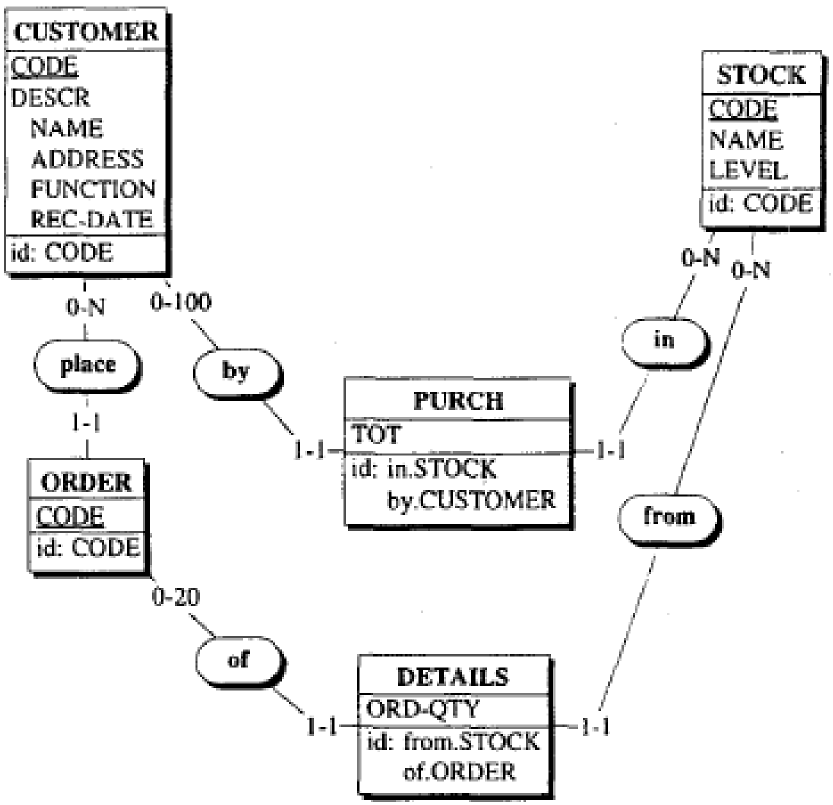
```
01 LIST-DETAIL.
  02 DETAILS OCCURS 20 TIMES
      INDEXED BY IND-DET.
    03 REF-DET-STK PIC 9(5).
    03 ORD-QTY PIC 9(5).
```

- **ORD-DETAIL is a complex multi-valued attribute**
  - **Highly efficient COBOL trick**
- **ORD-DETAIL cannot exist without an order**

- **How would you model this in ER?**
  - **Weak entity set**
  - **One-to-many relationship**

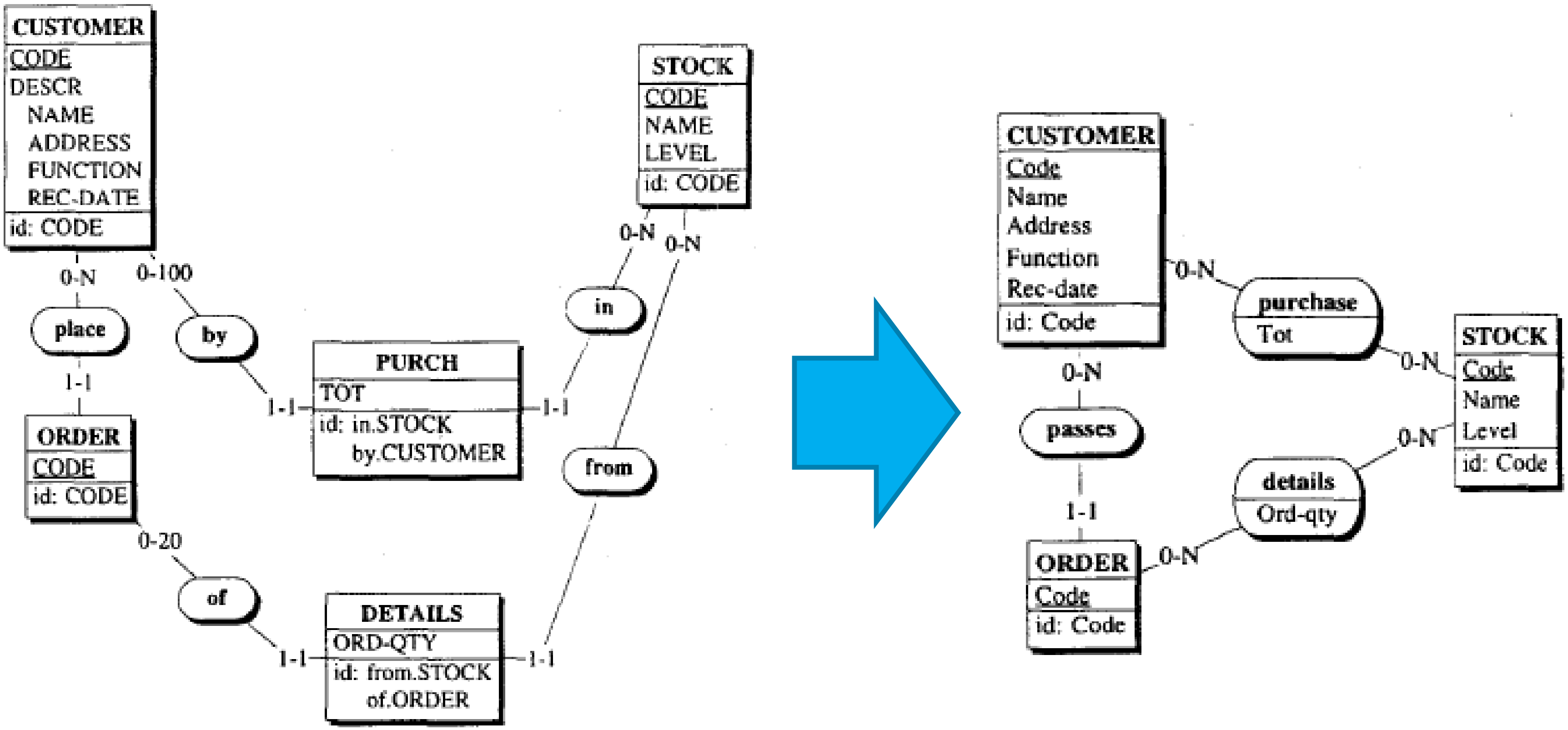
Technische Universiteit Eindhoven University of Technology

Conceptual normalization

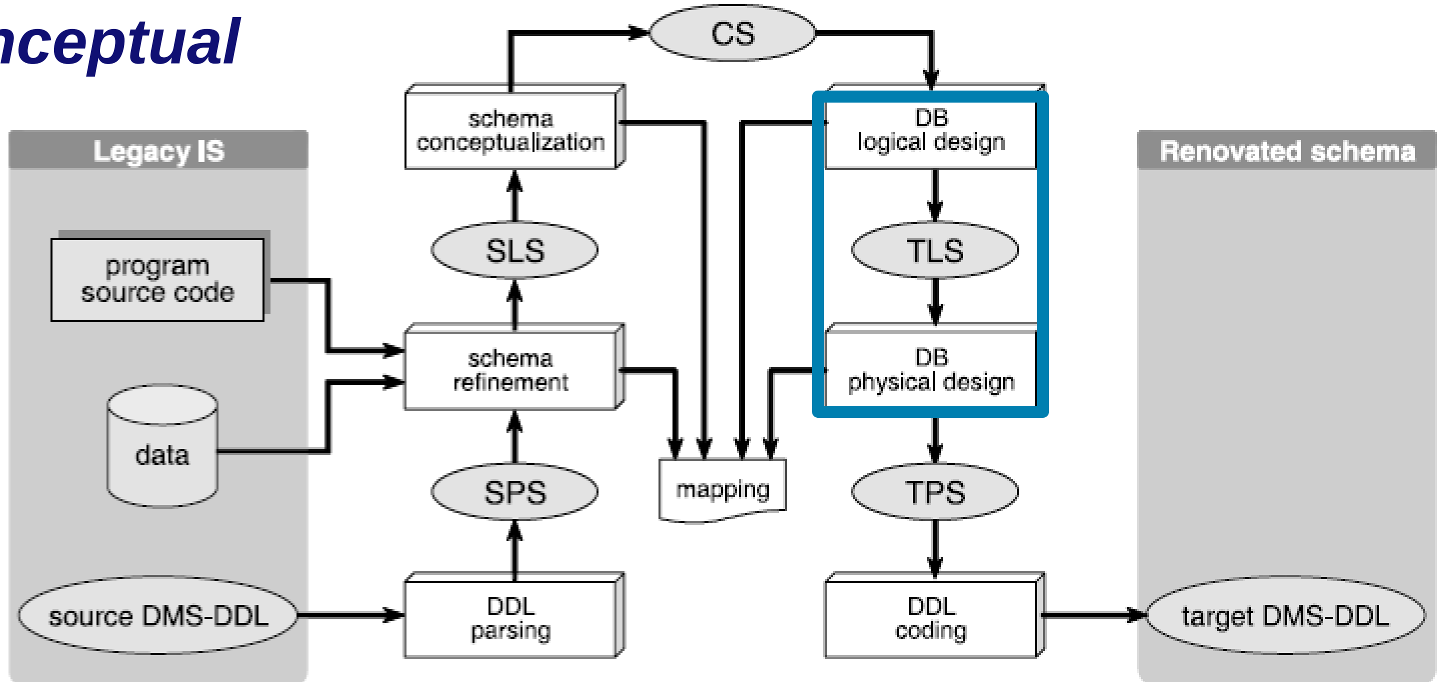


- What would you like to improve in this schema?
  - Are the cardinality constraints meaningful?
  - Which entities are, in fact, relations?
  - Are there unneeded structures?

Conceptual normalization

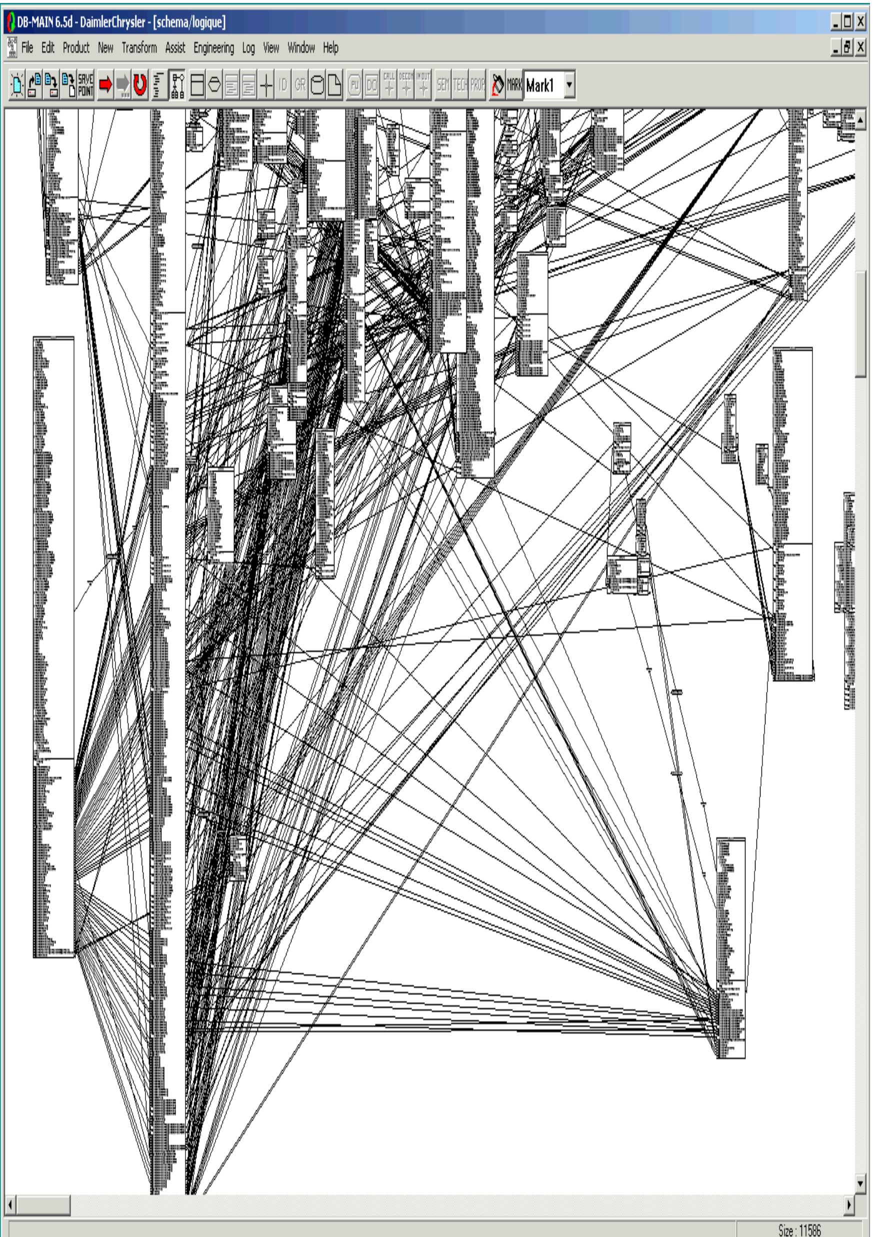


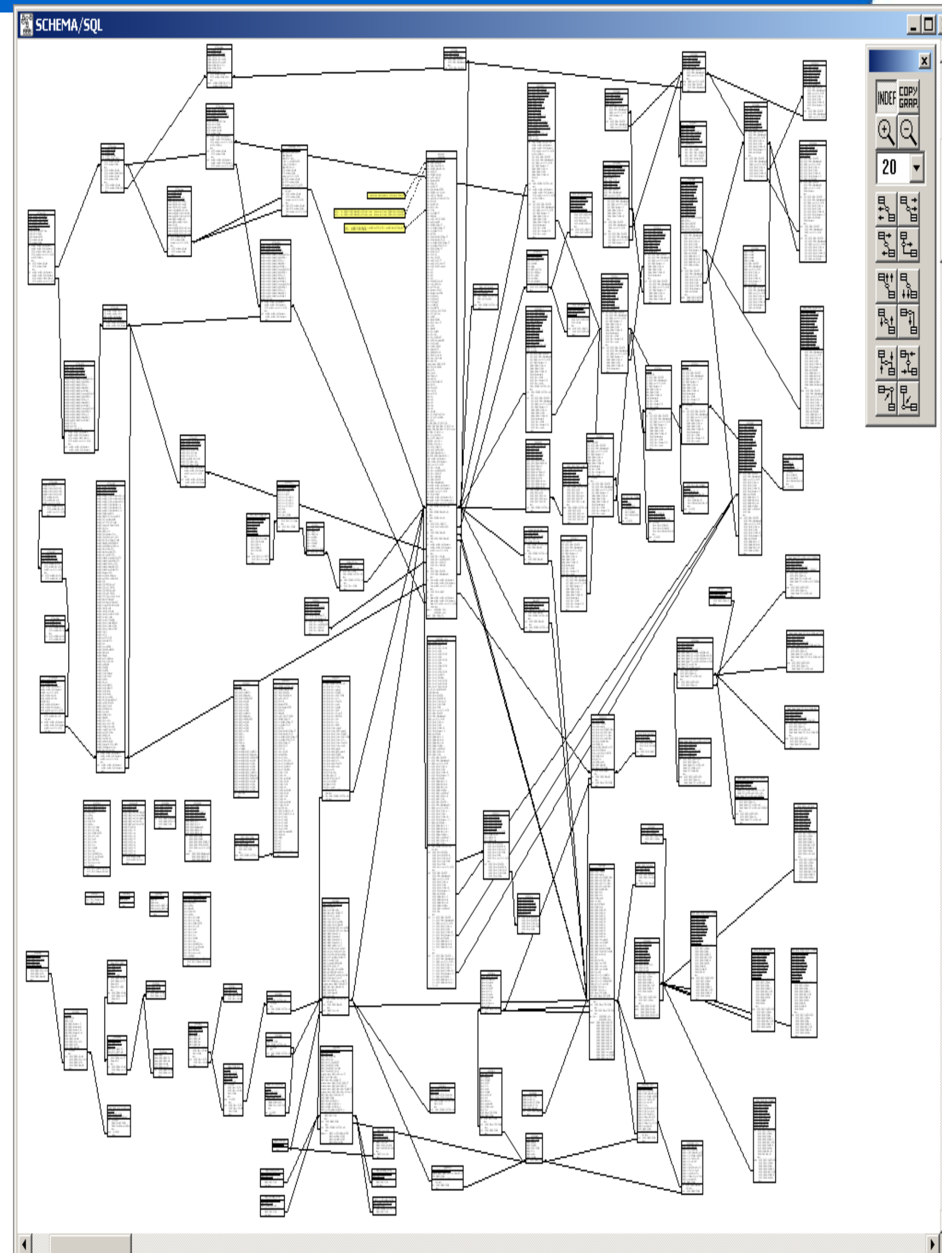
Conceptual



- **Logical design: schema concepts ⇒ DB tables**
- **Physical design: e.g., naming conventions**

Hainaut 2009: Before and After





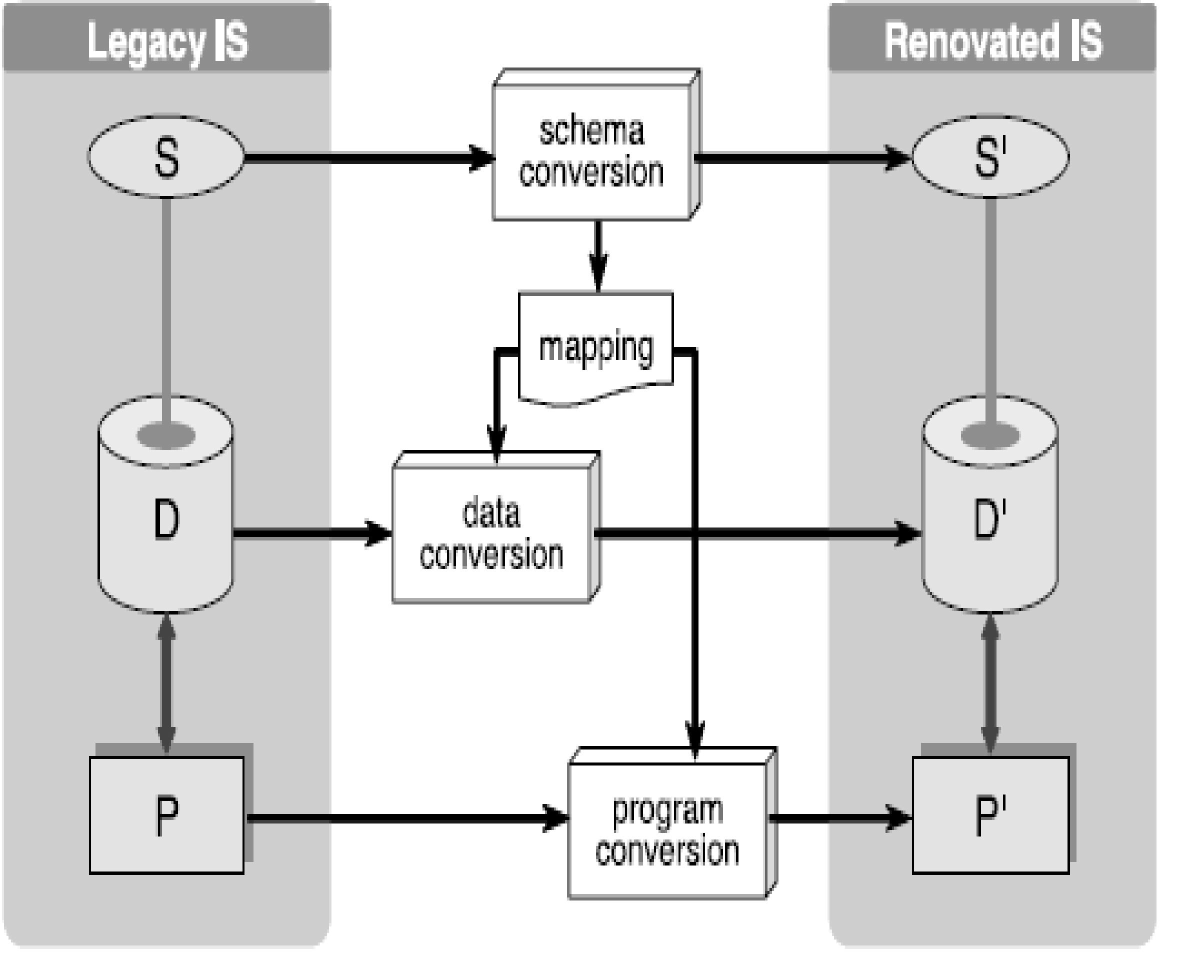
Another case study (Ch. 6)

	Physical IDS/II	Refined IDS/II	Conceptual	Relational DB2
# entity types	159	159	156	171
# relationship types	148	148	90	0
# attributes	458	9 027	2 176	2 118
max # att./entity type	8	104	61	94

- **Refined schema: decomposed attributes**
  - **Address = Street, Number, City, ZIP, State**
- **Schema refinement:**
  - **89 foreign keys, 37 computed foreign keys, 60 redundancies**
- **Relational DB2**
  - **↑entities: decomposition of arrays**

Recall…

- **So far we have considered DB schemas only**
- **Next step: data migration**



Technische Universiteit Eindhoven University of Technology

Data migration

- Strategy depends on the schema migration strategy

- Physical conversion: straightforward
  - Data format conversion

- Conceptual conversion
  - Data may violate implicit constraints
  - Hence, data cleaning is required as preprocessing
  - Once the data has been cleaned up: akin to physical conversion

What should be cleaned? 1 source [Rahm, Do]

- **Schema-level**
  - **Can be solved with appropriate integrity constraints**

Scope/Problem		Dirty Data	Reasons/Remarks
Attribute	Illegal values	bdate=30.13.70	values outside of domain range
Record	Violated attribute dependencies	age=22, bdate=12.02.70	age = (current date – birth date) should hold
Record type	Uniqueness violation	emp_1=(name="John Smith", SSN="123456") emp_2=(name="Peter Miller", SSN="123456")	uniqueness for SSN (social security number) violated
Source	Referential integrity violation	emp=(name="John Smith", deptno=127)	referenced department (127) not defined

- **Instance-level**

Scope/Problem		Dirty Data	Reasons/Remarks
Attribute	Missing values	phone=9999-999999	unavailable values during data entry (dummy values or null)
	Misspellings	city="Liipzig"	usually typos, phonetic errors
	Cryptic values, Abbreviations	experience="B"; occupation="DB Prog."	
	Embedded values	name="J. Smith 12.02.70 New York"	multiple values entered in one attribute (e.g. in a free-form field)
	Misfielded values	city="Germany"	
Record	Violated attribute dependencies	city="Redmond", zip=77777	city and zip code should correspond
Record type	Word transpositions	$name_1$= "J. Smith", $name_2$="Miller P."	usually in a free-form field
	Duplicated records	emp_1=(name="John Smith",...); emp_2=(name="J. Smith",...)	same employee represented twice due to some data entry errors
	Contradicting records	emp_1=(name="John Smith", bdate=12.02.70); emp_2=(name="John Smith", bdate=12.12.70)	the same real world entity is described by different values
Source	Wrong references	emp=(name="John Smith", deptno=17)	referenced department (17) is defined but wrong

What should be cleaned? Multiple sources

- **Which DB tuples refer to the same real-world entity?**

Customer (source 1)

CID	*Name*	*Street*	*City*	*Sex*
11	Kristen Smith	2 Hurley Pl	South Fork, MN 48503	0
24	Christian Smith	Hurley St 2	S Fork MN	1

Client (source 2)

Cno	*LastName*	*FirstName*	*Gender*	*Address*	*Phone/Fax*
24	Smith	Christoph	M	23 Harley St, Chicago IL, 60633-2394	333-222-6542 / 333-222-6599
493	Smith	Kris L.	F	2 Hurley Place, South Fork MN, 48503-5998	444-555-6666

Customers (integrated target with cleaned data)

No	*LName*	*FName*	*Gender*	*Street*	*City*	*State*	*ZIP*	*Phone*	*Fax*	*CID*	*Cno*
1	Smith	Kristen L.	F	2 Hurley Place	South Fork	MN	48503-5998	444-555-6666		11	493
2	Smith	Christian	M	2 Hurley Place	South Fork	MN	48503-5998			24	
3	Smith	Christoph	M	23 Harley Street	Chicago	IL	60633-2394	333-222-6542	333-222-6599		24

- **Scheme: name and structure conflicts**
- **Instance: data representation, duplication, identifiers**

How to clean up data?

- **Analyse:**
  - **Define inconsistencies and detect them**
- **Define individual transformations and the workflow**
- **Verify correctness and effectiveness**
  - **Sample/copy of the data**
- **Transform**
- **Backflow if needed**
  - **If the “old” data still will be used, it can benefit from the improvements.**

Data cleaning: Analysis

- **Data profiling**
  - **Instance analysis of individual attributes**
  - **Min, max, distribution, cardinality, uniqueness, null values**
    - **max(age) > 150? count(gender) > 2?**

- **Data mining**
  - **Instance analysis of relations between the attributes**
  - **E.g., detect association rules**
    - **Confidence(A $\Rightarrow$ B) = 99%**
    - **1% of the cases might require cleaning**

Data cleaning: Analysis (continued)

- Record matching problem:
  - Smith Kris L., Smith Kristen L., Smith Christian, …
- Matching based on
  - Simplest case: unique identifiers (primary keys)
  - Approximate matching
    - Different weights for different attributes
    - Strings:
      - Edit distance
      - Keyboard distance
      - Phonetic similarity
    - Very expensive for large data sets

Define data transformations

- **Use transformation languages**
  - **Proprietary (e.g., DataTransformationService of Microsoft)**
  - **SQL extended with user-defined functions (UDF):**

```
CREATE VIEW Customer2(LName, FName, Street, CID) AS
SELECT LastNameExtract(Name),
       FirstNameExtract(Name),
       Street, CID)
FROM Customer

CREATE FUNCTION LastNameExtract(Name VARCHAR(255))
  RETURNS VARCHAR(255)
  RETURN SUBSTRING(Name FROM 28 FOR 15)
```

UDF: advantages and disadvantages

- **Advantages**
  - **Does not require learning a separate language**

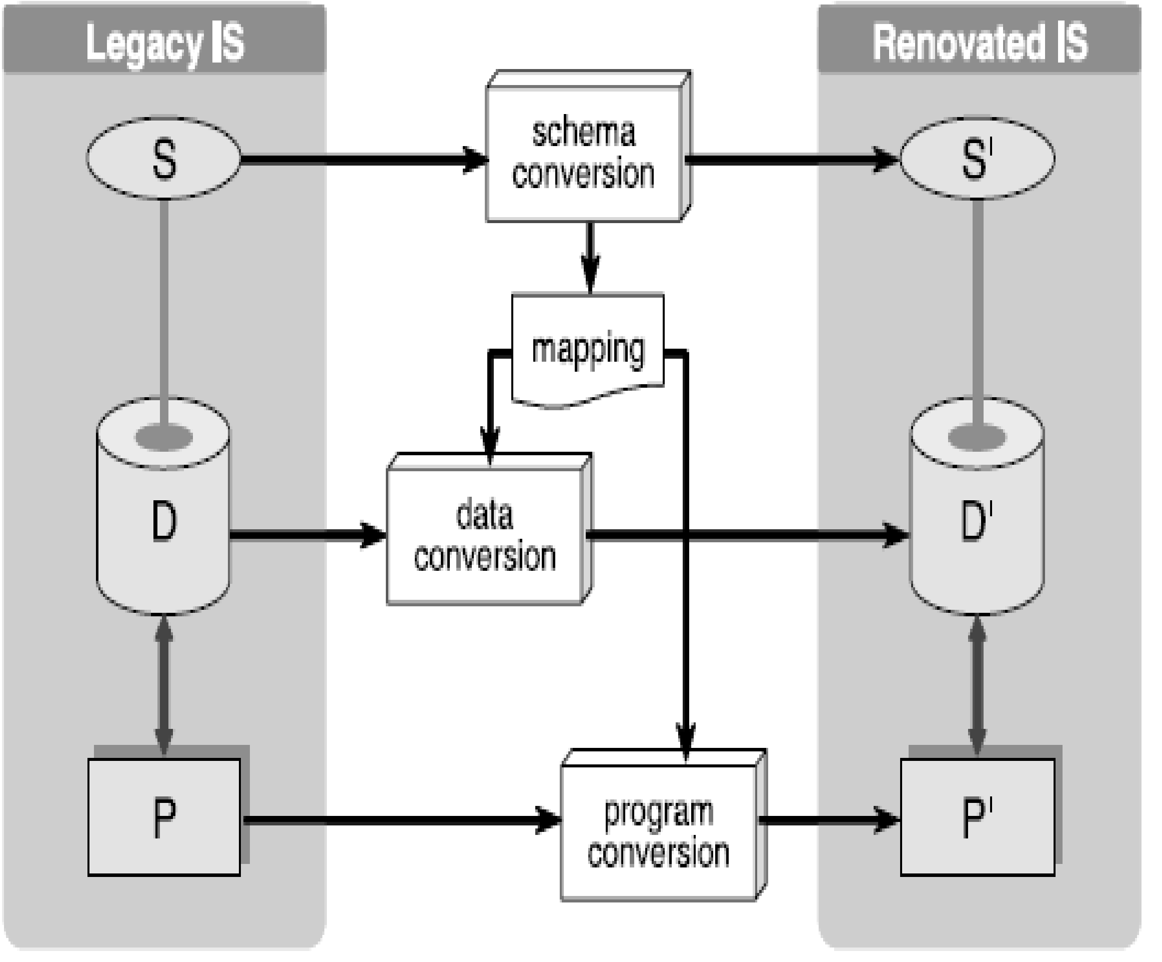
- **Disadvantages**
  - **Suited only for information already in a DB**
    - **What about COBOL files?**
  - **Ease of programming depends on availability of specific functions in the chosen SQL dialect**
    - **Splitting/merging are supported but have to be reimplemented for every separate field**
    - **Folding/unfolding of complex attributes not supported at all.**

Inconsistency resolution

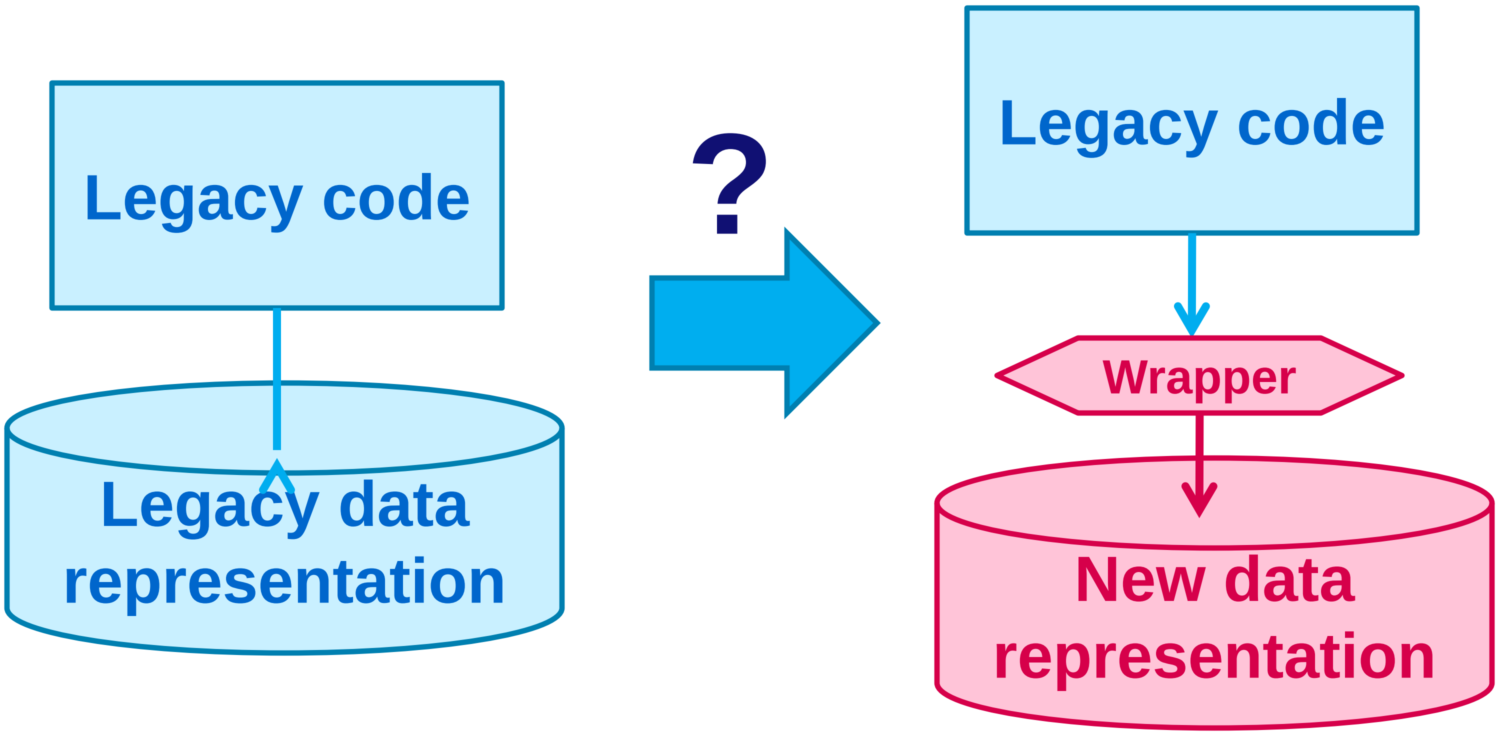
- If inconsistency has been detected, the offending instances
  - Are removed
  - Are modified so the offending data becomes NULL
  - Are modified by following user-defined preferences
    - One table might be more reliable than the other
    - One attribute may be more reliable than the other
  - Are modified to reduce the (total) number of modifications required to restore consistency

From data to programs

- **So far: schemas and data**
- **Next : programs**
  - **Wrapping**
  - **Statement rewriting**
  - **Program rewriting**



Wrappers



Legacy code
Legacy data representation
?
Legacy code
Wrapper
New data representation

Wrappers

- **Replace "standard" OPEN, CLOSE, READ, WRITE with wrapped operations**

```
DELETE-CUS-ORD.
    MOVE C-CODE TO O-CUST.
    MOVE 0 TO END-FILE.
    READ ORDERS KEY IS O-CUST
       INVALID KEY MOVE 1 TO END-FILE.
    PERFORM DELETE-ORDER UNTIL END-FILE = 1.
```

Actual implementation of "READ"

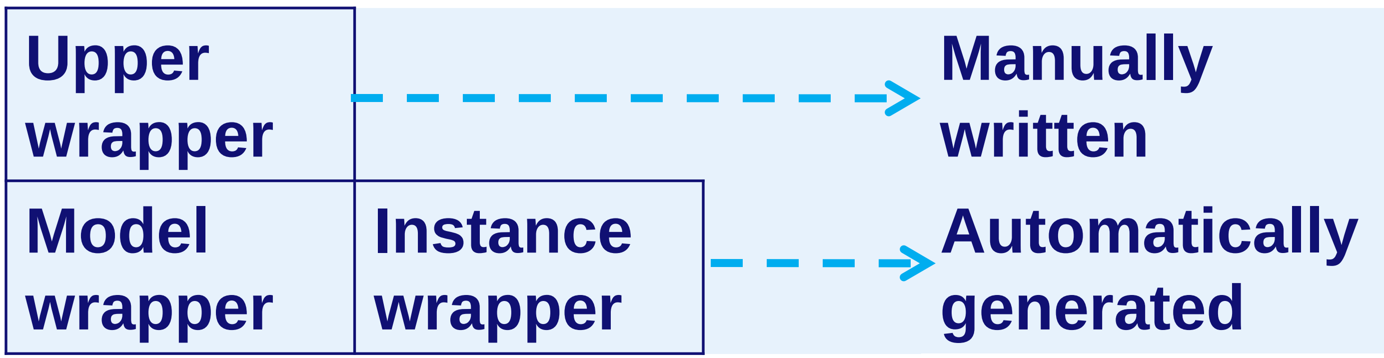
Start wrapping action "READ"

```
DELETE-CUS-ORD.
    MOVE C-CODE TO O-CUST.
    MOVE 0 TO END-FILE.
    SET WR-ACTION-READ TO TRUE.
    MOVE "KEY IS O-CUST" TO WR-OPTION.
    CALL WR-ORDERS USING WR-ACTION, ORD, WR-OPTION, WR-STATUS
    IF WR-STATUS-INVALID-KEY MOVE 1 TO END-FILE.
    PERFORM DELETE-ORDER UNTIL END-FILE = 1.
```

Wrappers

- [Thiran, Hainaut]: wrapper code can be reused

Cannot be expressed in the DB itself



Common to all DMS in the family: cursor, transaction

Specific to the given DB: query translation, access optimization

Wrapping: Pro and Contra

- **Wrapping**
  - **Preserves logic of the legacy system**
  - **Can be (partially) automated**
- **Physical + wrapper:**
  - **Almost automatic (cheap and fast)**
  - **Quality is poor, unless the legacy DB is well-structured**
- **Conceptual + wrapper:**
  - **More complex/expensive**
  - **Quality is reasonable: “First schema, then – code”**
  - **Possible performance penalty due to complexity of wrappers**
    - **Mismatch: “DB-like” schema and “COBOL like” code**

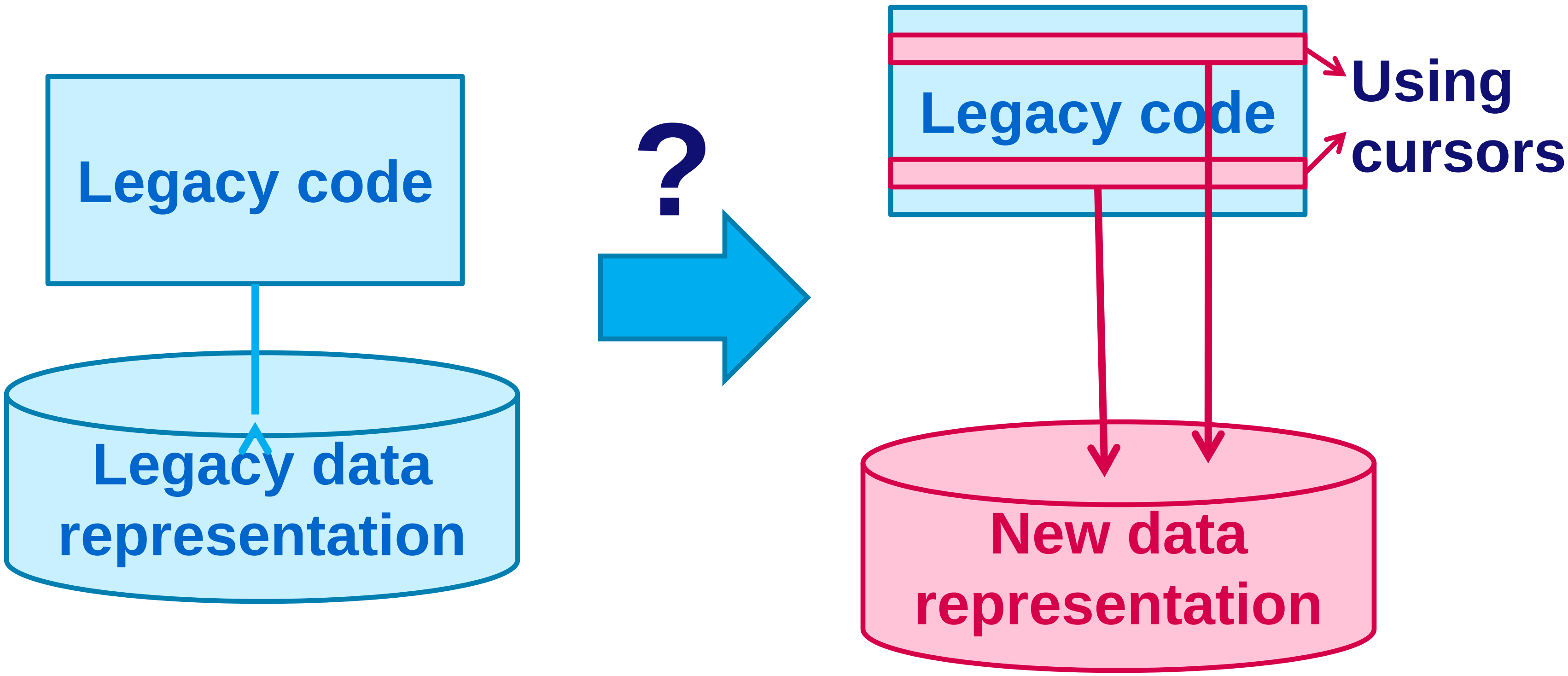
Wrapping in practice

Table 6.2. Program transformation results

	Migrated	Manually transformed
# programs	669	17
# copybooks	3 917	68
# IDS/II verbs	5 314	420

- **Wrappers**
  - **159 wrappers**
  - **450 KLOC**

Statement rewriting



Cursor?..

- **Control structure for the successive traversal of records in a query result**

- **Cursor declaration**

```
EXEC SQL DECLARE CURSOR ORD_GE_K1 FOR
  SELECT CODE, CUS_CODE
  FROM ORDERS WHERE CUS_CODE >= :O-CUST
  ORDER BY CUS_CODE
END-EXEC.
```

- **What will this cursor return? O_CUST = J12**

CUS_CODE	CODE
J11	12
J12	11
J13	14
K01	15

Why would you like to use such a cursor?

```
READ ORDERS KEY IS O-CUST
```

COBOL READ: Sequential reading starting from the first tuple with the given key

Cursor?..

- **Control structure for the successive traversal of records in a query result**

- **Cursor declaration**

```
EXEC SQL DECLARE CURSOR ORD_GE_K1 FOR
  SELECT CODE, CUS_CODE
  FROM ORDERS WHERE CUS_CODE >= :O-CUST
  ORDER BY CUS_CODE
END-EXEC.
```

- **Opening a cursor**

```
EXEC SQL OPEN ORD_GE_K1 END-EXEC
```

- **Retrieving data**

```
EXEC SQL
  FETCH ORD_GE_K1
  INTO :O-CODE, :O-CUST
END-EXEC
```

- **Closing cursor**

Statement rewriting

- **Replace “standard” OPEN, CLOSE, READ, WRITE with explicit SQL operations**

```
DELETE-CUS-ORD.
   MOVE C-CODE TO O-CUST.
   MOVE 0 TO END-FILE.
   READ ORDERS KEY IS O-CUST
     INVALID KEY MOVE 1 TO END-FILE.
   PERFORM DELETE-ORDER UNTIL END-FILE = 1.
```

```
DELETE-CUS-ORD.
  MOVE C-CODE TO O-CUST.
  MOVE 0 TO END-FILE.
  EXEC SQL
    SELECT COUNT(*) INTO :COUNTER
    FROM ORDERS WHERE CUS_CODE = :O-CUST
  END-EXEC.
  IF COUNTER = 0
     MOVE 1 TO END-FILE
  ELSE
     EXEC SQL OPEN ORD_GE_K1 END-EXEC
     MOVE "ORD_GE_K1" TO ORD-SEQ
     EXEC SQL
       FETCH ORD_GE_K1
       INTO :O-CODE, :O-CUST
     END-EXEC
     IF SQLCODE NOT = 0
        MOVE 1 TO END-FILE
     ELSE
        EXEC SQL OPEN ORD_DETAIL END-EXEC
        SET IND-DET TO 1
        MOVE 0 TO END-DETAIL
        PERFORM FILL-ORD-DETAIL UNTIL END-DETAIL = 1
     END-IF
  END-IF.
  PERFORM DELETE-ORDER UNTIL END-FILE = 1.
```

Statement rewriting

- **Replace "standard" OPEN, CLOSE, READ, WRITE with explicit SQL operations**

```
DELETE-CUS-ORD.


   READ ORDERS KEY IS O-CUST
     INVALID KEY MOVE 1 TO END-FILE.
```

O-CUST does not appear in ORDERS

```
DELETE-CUS-ORD.


    EXEC SQL
      SELECT COUNT(*) INTO :COUNTER
      FROM ORDERS WHERE CUS_CODE = :O-CUST
    END-EXEC.
    IF COUNTER = 0
       MOVE 1 TO END-FILE
    ELSE
       EXEC SQL OPEN ORD_GE_K1 END-EXEC
       MOVE "ORD_GE_K1" TO ORD-SEQ
       EXEC SQL
         FETCH ORD_GE_K1
         INTO :O-CODE, :O-CUST
       END-EXEC
       IF SQLCODE NOT = 0
          MOVE 1 TO END-FILE
       ELSE
          EXEC SQL OPEN ORD_DETAIL END-EXEC
          SET IND-DET TO 1
          MOVE 0 TO END-DETAIL
          PERFORM FILL-ORD-DETAIL UNTIL END-DETAIL = 1
       END-IF
    END-IF.
```

Statement rewriting

- **Replace "standard" OPEN, CLOSE, READ, WRITE with explicit SQL operations**

```
DELETE-CUS-ORD.

   READ ORDERS KEY IS O-CUST
```

```
DELETE-CUS-ORD.

     EXEC SQL OPEN ORD_GE_K1 END-EXEC
     MOVE "ORD_GE_K1" TO ORD-SEQ
     EXEC SQL
       FETCH ORD_GE_K1
       INTO :O-CODE, :O-CUST
     END-EXEC
     IF SQLCODE NOT = 0
        MOVE 1 TO END-FILE
     ELSE
        EXEC SQL OPEN ORD_DETAIL END-EXEC
        SET IND-DET TO 1
        MOVE 0 TO END-DETAIL
        PERFORM FILL-ORD-DETAIL UNTIL END-DETAIL = 1
     END-IF
```

- **Files can have multiple keys and multiple READ commands**

- **We need to remember which key/READ is used!**

```
IF ORD-SEQ = "ORD_GE_K1"
   EXEC SQL
     FETCH ORD_GE_K1 INTO :O-CODE,:O-CUST
   END-EXEC
```

Statement rewriting

- **Replace "standard" OPEN, CLOSE, READ, WRITE with explicit SQL operations**

```
DELETE-CUS-ORD.

   READ ORDERS KEY IS O-CUST
```

- **Prepare the cursor for READing**
- **READ the data**

```
DELETE-CUS-ORD.

    EXEC SQL OPEN ORD_GE_K1 END-EXEC

    EXEC SQL
      FETCH ORD_GE_K1
      INTO :O-CODE, :O-CUST
    END-EXEC
    IF SQLCODE NOT = 0
       MOVE 1 TO END-FILE
    ELSE
       EXEC SQL OPEN ORD_DETAIL END-EXEC
       SET IND-DET TO 1
       MOVE 0 TO END-DETAIL
       PERFORM FILL-ORD-DETAIL UNTIL END-DETAIL = 1
    END-IF
```

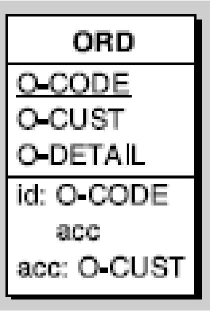
Statement rewriting

- **Replace “standard” OPEN, CLOSE, READ, WRITE with explicit SQL operations**

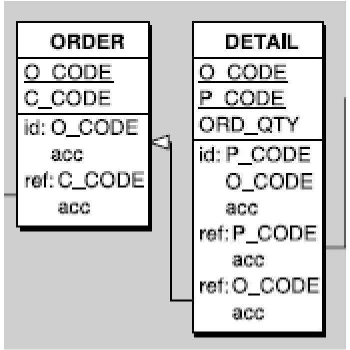
```
DELETE-CUS-ORD.

   READ ORDERS KEY IS O-CUST
```

- **We need additional cursor and procedure to read the order details:**



Legacy DB



New DB

```
DELETE-CUS-ORD.

     EXEC SQL OPEN ORD_GE_K1 END-EXEC

     EXEC SQL
       FETCH ORD_GE_K1
       INTO :O-CODE, :O-CUST
     END-EXEC
     IF SQLCODE NOT = 0
        MOVE 1 TO END-FILE
     ELSE
        EXEC SQL OPEN ORD_DETAIL END-EXEC
        SET IND-DET TO 1
        MOVE 0 TO END-DETAIL
        PERFORM FILL-ORD-DETAIL UNTIL END-DETAIL = 1
     END-IF
```

Statement rewriting: Pro and Contra

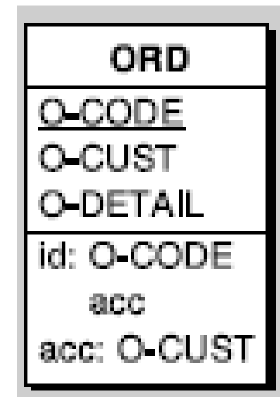
- Statement rewriting
  - Preserves logic of the legacy system
  - Intertwines legacy code with new access techniques
  - Detrimental for maintainability
- Physical + statement
  - Inexpensive and popular
  - Blows up the program: from 390 to ~1000 LOC
  - Worst strategy possible
- Conceptual + statement
  - Good quality DB, unreadable code: “First schema, then – code”
  - Meaningful if the application will be rewritten on the short term

Alternative 3: Logic Rewriting

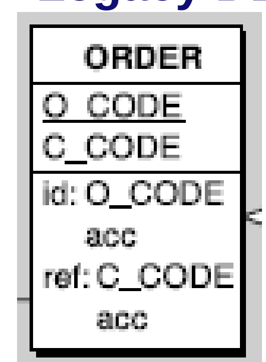
- **Akin to conceptual conversion**
  - **e.g., COBOL loop ⇒ SQL join**
  - **And meaningful only in combination with it**
    - **Otherwise: high effort with poor results**

```
DELETE-CUS-ORD.
   MOVE C-CODE TO O-CUST.
   MOVE 0 TO END-FILE.
   READ ORDERS KEY IS O-CUST
     INVALID KEY MOVE 1 TO END-FILE.
   PERFORM DELETE-ORDER UNTIL END-FILE = 1.
```

```
DELETE-CUS-ORD.
   EXEC SQL
     DELETE FROM ORDERS
     WHERE CUS_CODE = :C-CODE
   END-EXEC.
   IF SQLCODE NOT = 0 THEN GO TO ERR-DEL-ORD.
```



Legacy DB



New DB

Alternative 3: Logic Rewriting

- **Manual transformation with automatic support**
  - **Identify file access statements**
  - **Identify and understand data and statements that depend on these statements**
  - **Rewrite these statements and redefine the objects**

```
DELETE-CUS-ORD.
    MOVE C-CODE TO O-CUST.
    MOVE 0 TO END-FILE.
    READ ORDERS KEY IS O-CUST
       INVALID KEY MOVE 1 TO END-FILE.
    PERFORM DELETE-ORDER UNTIL END-FILE = 1.
```

```
DELETE-CUS-ORD.
    EXEC SQL
       DELETE FROM ORDERS
       WHERE CUS_CODE = :C-CODE
    END-EXEC.
    IF SQLCODE NOT = 0 THEN GO TO ERR-DEL-ORD.
```

Logic rewriting: Pro and Contra

- **Logic rewriting + physical**
  - **Low quality DB**
  - **High costs due to logic rewriting**
  - **Unfeasible**
- **Logic rewriting + conceptual**
  - **High quality**
  - **Highest costs**

Putting it all together

- All combinations are possible
- Not all are desirable

Scheme \ Code	Wrapping	Statement	Logic
Conc.	Conceptual, Wrapping	Conceptual, Statement	Conceptual, Logic
Phys.	Physical, Wrapping	Physical, Statement	Physical, Logic

Putting it all together

- **All combinations are possible**
- **Not all are desirable**

Scheme \ Code	Wrapping	Statement	Logic
Conc.	Better DB, performance penalty	Better DB, the application is rewritten later	Best but also $$$
Phys.	Zero time	Popular, $, Bad	Very bad, $$

Tools

- DB-MAIN CASE tool (University of Namur, ReVeR)
  - DDL extraction
  - Schema storage, analysis and manipulation
  - Implicit constraint validation
  - Schema mapping management
  - Data analysis & migration
  - Wrapper generation (COBOL-to-SQL, CODASYL-to-SQL)
- Transformations
  - Eclipse Modelling Framework: ATL
  - ASF+SDF Meta-Environment (CWI, Amsterdam)

People ((ex)-FUNDP)

- **Anthony Cleve**

- **Jean-Luc Hainaut**

- **Philippe Thiran**

Conclusions

- 3 levels of DB migration: schema, data, code

- Schema: physical/conceptual
- Data: determined by schema
- Code: wrapper/statement rewriting/logical rewriting

- Popular but bad: physical + statement
- Expensive but good: conceptual + logic
- Alternatives to consider:
  - conceptual + wrapping/statement
  - physical + wrapping (zero time)