Implementing evolution: Database migration

Alexander Serebrenik
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  • Data is important, technology is outdated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

Physical conversion / Conceptual conversion

Wrap / Statement rewriting / Logical rewriting
Schema conversion: Physical vs Conceptual

Physical

Conceptual
Schema conversion: Physical

- Easy to automate
  - Existing work: COBOL \(\Rightarrow\) relational, hierarchical \(\Rightarrow\) relational, relational \(\Rightarrow\) OO
  - “Migration as translation” vs “migration as improvement”
- Semantics is ignored
  - Limitations of COBOL \(\Rightarrow\) Design decisions in the legacy system \(\Rightarrow\) Automatic conversion \(\Rightarrow\) the same design decisions in the new system
    - Risk: compromised flexibility
• **Refinement:** Data and code may contain implicit constraints (field refinement, foreign key, cardinality) on the schema
Conceptualization

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
De-optimization

• **Recall:**

```cobol
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
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

CUSTOMER
- CODE
- DESCR
- NAME
- ADDRESS
- FUNCTION
- REC-DATE
  - id: CODE
  - 0-100

ORDER
- CODE
  - id: CODE
  - 0-20

PURCH
- TOT
  - id: in.STOCK
  - 1-1

STOCK
- CODE
- NAME
- LEVEL
  - id: CODE
  - 0-N

CUSTOMER
- Code
- Name
- Address
- Function
- Rec-date
  - id: Code
  - 0-N

purchase
  - Tot
  - 0-N

pass
  - 1-1

details
  - Ord-qty
  - 0-N

ORDER
  - Code
  - id: Code
  - 0-N

DETAILS
- ORD-QTY
  - id: from.STOCK
  - of.ORDER
  - 1-1

place
  - by
  - 1-1

of
  - 1-1
Logical design: schema concepts ⇒ DB tables
Physical design: e.g., naming conventions
Hainaut 2009: Before and After
Another case study (Ch. 6)

- 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

<table>
<thead>
<tr>
<th></th>
<th>Physical IDS/II</th>
<th>Refined IDS/II</th>
<th>Conceptual</th>
<th>Relational DB2</th>
</tr>
</thead>
<tbody>
<tr>
<td># entity types</td>
<td>159</td>
<td>159</td>
<td>156</td>
<td>171</td>
</tr>
<tr>
<td># relationship types</td>
<td>148</td>
<td>148</td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td># attributes</td>
<td>458</td>
<td>9,027</td>
<td>2,176</td>
<td>2,118</td>
</tr>
<tr>
<td>max # att./entity type</td>
<td>8</td>
<td>104</td>
<td>61</td>
<td>94</td>
</tr>
</tbody>
</table>
Recall...

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

<table>
<thead>
<tr>
<th>Scope/Problem</th>
<th>Dirty Data</th>
<th>Reasons/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Illegal values</td>
<td>values outside of domain range</td>
</tr>
<tr>
<td>bdate=30.13.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td>Violated attribute dependencies</td>
<td>age = (current date – birth date) should hold</td>
</tr>
<tr>
<td>age=22, bdate=12.02.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record type</td>
<td>Uniqueness violation</td>
<td>uniqueness for SSN (social security number) violated</td>
</tr>
<tr>
<td>emp₁=(name=”John Smith”, SSN=”123456”)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>emp₂=(name=”Peter Miller”, SSN=”123456”)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td>Referential integrity violation</td>
<td>referenced department (127) not defined</td>
</tr>
<tr>
<td>emp=(name=”John Smith”, deptno=127)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Instance-level**

<table>
<thead>
<tr>
<th>Scope/Problem</th>
<th>Dirty Data</th>
<th>Reasons/Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Missing values</td>
<td>unavailable values during data entry (dummy values or null)</td>
</tr>
<tr>
<td>phone=9999-999999</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Misspellings</td>
<td>city=”Liipzig”</td>
<td>usually typos, phonetic errors</td>
</tr>
<tr>
<td>Cryptic values, Abbreviations</td>
<td>experience=”B”; occupation=”DB Prog.”</td>
<td></td>
</tr>
<tr>
<td>Embedded values</td>
<td>name=”J. Smith 12.02.70 New York”</td>
<td>multiple values entered in one attribute (e.g. in a free-form field)</td>
</tr>
<tr>
<td>Misfielded values</td>
<td>city=”Germany”</td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td>Violated attribute dependencies</td>
<td>city and zip code should correspond</td>
</tr>
<tr>
<td>city=”Redmond”, zip=77777</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record type</td>
<td>Word transpositions</td>
<td>usually in a free-form field</td>
</tr>
<tr>
<td>name₁ = “J. Smith”, name₂=”Miller P.”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duplicated records</td>
<td>emp₁=(name=”John Smith”, bdate=12.02.70); emp₂=(name=”J. Smith”,...)</td>
<td>same employee represented twice due to some data entry errors</td>
</tr>
<tr>
<td>Contradicting records</td>
<td>emp₁=(name=”John Smith”, bdate=12.02.70), emp₂=(name=”John Smith”, bdate=12.12.70)</td>
<td>the same real world entity is described by different values</td>
</tr>
<tr>
<td>Source</td>
<td>Wrong references</td>
<td>referenced department (17) is defined but wrong</td>
</tr>
<tr>
<td>emp=(name=”John Smith”, deptno=17)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What should be cleaned? Multiple sources

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

<table>
<thead>
<tr>
<th>Customer (source 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Client (source 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cno</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>493</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customers (integrated target with cleaned data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

- 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 ⇒ 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):

```sql
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

Wrapper

Legacy code

New data representation
Wrappers

- Replace “standard” OPEN, CLOSE, READ, WRITE with wrapped operations

Start wrapping action “READ”

Actual implementation of “READ”

```
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.
  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

<table>
<thead>
<tr>
<th>Upper wrapper</th>
<th>Manually written</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model wrapper</td>
<td>Automatically generated</td>
</tr>
<tr>
<td>Instance wrapper</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Migrated</th>
<th>Manually transformed</th>
</tr>
</thead>
<tbody>
<tr>
<td># programs</td>
<td>669</td>
<td>17</td>
</tr>
<tr>
<td># copybooks</td>
<td>3,917</td>
<td>68</td>
</tr>
<tr>
<td># IDS/II verbs</td>
<td>5,314</td>
<td>420</td>
</tr>
</tbody>
</table>

- Wrappers
  - 159 wrappers
  - 450 KLOC
Statement rewriting

Legacy code

Legacy data representation

Using cursors

Legacy code

New data representation
Cursor?..

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

- Cursor declaration

- What will this cursor return? O_CUST = J12

<table>
<thead>
<tr>
<th>CUS_CODE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>J11</td>
<td>12</td>
</tr>
<tr>
<td>J12</td>
<td>11</td>
</tr>
<tr>
<td>J13</td>
<td>14</td>
</tr>
<tr>
<td>K01</td>
<td>15</td>
</tr>
</tbody>
</table>

Why would you like to use such a cursor?

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

- Opening a cursor

- Retrieving data

- Closing cursor

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.

EXEC SQL OPEN ORD_GE_K1 END-EXEC

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
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

- Files can have multiple keys and multiple READ commands

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

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

• Prepare the cursor for READING
• READ the data
Statement rewriting

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

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

Legacy DB

New DB
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.
```
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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Conc.</th>
<th>Phys.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual, Wrapping</td>
<td>Conceptual, Statement</td>
<td>Conceptual, Logic</td>
</tr>
<tr>
<td>Physical, Wrapping</td>
<td>Physical, Statement</td>
<td>Physical, Logic</td>
</tr>
</tbody>
</table>

Code
Putting it all together

- All combinations are possible
- Not all are desirable

<table>
<thead>
<tr>
<th>Code</th>
<th>Scheme</th>
<th>Conc.</th>
<th>Phys.</th>
<th>Wrapping</th>
<th>Statement</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero time</td>
<td>Better DB, performance penalty</td>
<td>Better DB, the application is rewritten later</td>
<td>Very bad, $$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popular, $, Bad</td>
<td>Best but also $$$</td>
<td>Zero time</td>
<td>Popular, $$, Bad</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very bad, $$</td>
<td>Best but also $$$</td>
<td>Very bad, $$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• All combinations are possible
• Not all are desirable
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)