Last week

- Assignment 8
- How is it going?
- Questions to Marcel: m.f.v.amstel@tue.nl

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  - Data is important, technology is outdated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

- 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

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:
  01 ORDERS.
  01 LIST-DETAIL.
  02 ORD.
  02 DETAIL occurs 26 TIMES
  02 ORD-CUSTOMER PIC X(12).
  02 DETAIL-PK PIC 9(5).
  02 DETAIL-SK PIC X(25).
  02 DETAIL-KEY PIC X(25).
- 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

Logical design: schema concepts $\Rightarrow$ DB tables

Physical design: e.g., naming conventions

Hainaut 2009: Before and After

Another case study (Ch. 6)

Recall...

Data migration

• Logical design: schema concepts $\Rightarrow$ DB tables
• Physical design: e.g., naming conventions

• Refined schema: decomposed attributes
  • Address = Street, Number, City, ZIP, State
• Schema refinement:
  • 89 foreign keys, 37 computer foreign keys, 60 redundancies
• Relational DB2
  • Entities: decomposition of arrays

• 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

• So far we have considered DB schemas only
• Next step: data migration
What should be cleaned? 1 source [Rahm, Do]

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

<table>
<thead>
<tr>
<th>Source</th>
<th>Detail</th>
<th>Related Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Smith</td>
<td>John</td>
</tr>
<tr>
<td>Address</td>
<td>Home</td>
<td>Office</td>
</tr>
</tbody>
</table>

What should be cleaned? Multiple sources

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

<table>
<thead>
<tr>
<th>Source</th>
<th>Detail</th>
<th>Related Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Smith</td>
<td>John</td>
</tr>
<tr>
<td>Address</td>
<td>Home</td>
<td>Office</td>
</tr>
</tbody>
</table>

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 will still 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 \implies B) = 99%
    - 1% of the cases might require cleaning

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) 
  RETURNS 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

- If multiple inconsistencies are detected
  - Which one has to be selected to be resolved?
    - Usually resolutions are not independent
    - [Wijsen 2006]:
      - Single database instead of multiple uprepairs
      - Which one is more important?

From data to programs

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

Wrappers

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

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

Actual implementation of "READ"

Start wrapping action "READ"
Wrappers

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

Cannot be expressed in the DB itself

Upper wrapper
Model wrapper
Instance wrapper

Manually written
Automatically generated

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>Migrated</th>
<th>Manually transformed</th>
</tr>
</thead>
<tbody>
<tr>
<td># programs</td>
<td>669</td>
</tr>
<tr>
<td># copybooks</td>
<td>3,917</td>
</tr>
<tr>
<td># IDMS verbs</td>
<td>5,314</td>
</tr>
</tbody>
</table>

• Wrappers
  • 159 wrappers
  • 450 KLOC

Cursor?..?

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

  Cursor declaration

  Why would you like to use such a cursor?

  CUS_CODE CODE
  J11 12
  J12 11
  J13 14
  K01 15

What will this cursor return? O_CUST = J12

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

Statement rewriting

Legacy code

Using cursors

Legacy code

New data representation

Legacy data representation
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!

Prepare the cursor for READING
- READ the data

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

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

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)