Top 20 Data Quality Solutions for Data Science

Data Science & Business Analytics Meetup
Denver, CO    2015-01-21

Ken Farmer
DQ Problems for Data Science Loom Large & Frequently

Impacts Include:
- Strikingly visible defects
- Bad Decisions
- Loss of credibility
- Loss of revenue

Types of Problems Include:
- Requirement & Design Defects
- Misinterpretation Errors
- Source Data Defects
- Process Errors

![Graph showing the increase in widgets over time with a significant drop in months 19 to 20.](image-url)
Let's Talk about Solutions

But keep in mind...
Proportionality is Important
Solution #1: Quality Assurance (QA)

“If it's not tested it's broken” - Bruce Eckel

Tests of application and system behavior
- input assumptions
- performed after application changes

Programmer Testing:
- Unit Testing
- Functional Testing

QA Team Testing:
- Black Box Testing
- Regression Testing
- System and Integration Testing

Data Scientist Testing:
- All of the above
- Especially Programmer
Solution #2: Quality Control (QC)

Because the environment & inputs are ever-changing

Track & analyze record counts:
- identify partial data sets
- identify upstream changes

Track & analyze rule counts:
- identify upstream changes

Track & analyze replication & aggregation consistency:
- identify process failures
Solution #3: Data Profiling

Find out what data looks like BEFORE you model it

Save enormous amount of time:
- quickly get the type, size, cardinality, unk values
- share profiling results with users

Example Deliverables:
- What is the distribution of data for every field?
- How do partitions affect data distributions?
- What correlations exist between fields?

<table>
<thead>
<tr>
<th>Name</th>
<th>f250_499</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Number</td>
<td>59</td>
</tr>
<tr>
<td>Wrong Field Cnt</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Min</td>
<td>C</td>
</tr>
<tr>
<td>Max</td>
<td>M</td>
</tr>
<tr>
<td>Unique Values</td>
<td>12</td>
</tr>
<tr>
<td>Known Values</td>
<td>11</td>
</tr>
<tr>
<td>Case</td>
<td>mixed</td>
</tr>
<tr>
<td>Min Length</td>
<td>1</td>
</tr>
<tr>
<td>Max Length</td>
<td>1</td>
</tr>
<tr>
<td>Mean Length</td>
<td>1.0</td>
</tr>
<tr>
<td>Top Values</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>x 10056 occurrences</td>
</tr>
<tr>
<td>F</td>
<td>x 1299 occurrences</td>
</tr>
<tr>
<td>G</td>
<td>x 969 occurrences</td>
</tr>
<tr>
<td>C</td>
<td>x 358 occurrences</td>
</tr>
<tr>
<td>H</td>
<td>x 120 occurrences</td>
</tr>
<tr>
<td>I</td>
<td>x 89 occurrences</td>
</tr>
<tr>
<td>J</td>
<td>x 36 occurrences</td>
</tr>
<tr>
<td>M</td>
<td>x 18 occurrences</td>
</tr>
<tr>
<td>K</td>
<td>x 12 occurrences</td>
</tr>
<tr>
<td>e</td>
<td>x 11 occurrences</td>
</tr>
<tr>
<td>e</td>
<td>x 4 occurrences</td>
</tr>
</tbody>
</table>
Solution #4: Organization

So people don't use the wrong data
or the right data incorrectly

Manage Historical Data:
- Migrate old data
  - Schemas
  - Rules
- Curate adhoc files
  - Segregate
  - Name
  - Document
  - Eliminate

Simplify Data Models:
- Consistency in naming, types, defaults
- Simplicity in relationships and values
Solution #5: Process Auditing

Because with a lot of moving parts comes a lot of failures

Addresses:
- Slow processes
- Broken processes

Helps identify:
- Duplicate loads
- Missing files
- Pipeline status

Features:
- Tracks all processes: start, stop, times, return codes, batch_ids
- Alerting

Challenge with Streaming:
- Helps to create artificial batch concept from timestamp within the data

Example Products:
- Graphite - focus: resources
- Nagios - focus: process failure
- Or what's built-into every ETL tool

(the inspecting metaphor, not the searching metaphor)
Solution #6: Full Automation

Because manual processes account for a majority of problems

Addresses:
- Duplicate data
- Missing data

Top Causes:
- manual process restarts
- manual data recoveries
- manual process overrides

Solution Characteristics:
- Test catastrophic failures
- Automate failure recoveries
- Consider Netflix's Chaos Monkey
Solution #7: Changed Data Capture

Because identifying changes is harder than most people realize

Typical Alternatives:

Application Timestamp:
- pro: may already be there
- con: reliability challenges

Triggers:
- pro: simple for downstream
- con: reliability challenges
- con: requires changes to source

File-Image Delta:
- pro: implementation effort
- con: very accurate

File Image Delta Example

Source Data → Sort → Dedup → Same, Inserts, Deletes, Change New, Change Old → Target Data

File Delta & Transform
Solution #8: Master Data Management

Because sharing reference data eliminates many issues

Addresses:
- Data consistency between multiple systems

Features:
- Centralized storage of reference data
- Versioning of data
- Access via multiple protocols

Because sharing reference data eliminates many issues
Solution #9: Extrapolate for Missing Data

Because *if done well it can simplify queries*

Features:
- Requires sufficient data to identify pattern
- Identify generated data (see Quality Indicator & Dimension)
Solution #10: Crowd-sourced Data Cleansing

Because cleansing & enrichening data can benefit from consensus

Features:
- Collect consensus answers from workers
- Workers can be from external market
- Workers can be from your team

Example Product:
- CrowdFlower
- Mechanical Turk

Simple Data Scenario:
- Correct obvious problems
  - Spelling
  - Grammar
  - Missing descriptions
- Use public resources

Complex Data Scenario:
- Correct sophisticated problems
  - Provide scores
  - Provide descriptions
  - Correct complex data
- Use internal & external resources
- Leverage crowdsourcing services for coordination
Solution #11: Keep Original Values

Because your transformations will fail & you will change your mind

Options:

- keep archive copy of source data
  - Pro: can be very highly compressed
  - Pro: can be kept off-server
  - Con: cannot be easily queried

- Or keep with transformed data
  - Pro: can be easily queried
  - Con: may be used when it should not be
  - Con: may double volume of data to host

<table>
<thead>
<tr>
<th>os_orig</th>
<th>os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win2k</td>
<td>win2000</td>
</tr>
<tr>
<td>MS Win</td>
<td>win2000</td>
</tr>
<tr>
<td>Win 2000</td>
<td>win2000</td>
</tr>
<tr>
<td>Windoze 2k</td>
<td>win2000</td>
</tr>
<tr>
<td>Windows 2000 SP4</td>
<td>win2000</td>
</tr>
<tr>
<td>MS Win 2k SP2</td>
<td>win2000</td>
</tr>
<tr>
<td>ms win 2000 server ed</td>
<td>win2000</td>
</tr>
<tr>
<td>win2ksp3</td>
<td>win2000</td>
</tr>
<tr>
<td>Win 2k server sp9</td>
<td>win2000</td>
</tr>
</tbody>
</table>
Solution #12: Keep usage logs

Because knowing who got bad data can help you minimize impact

Solution Types:

- Log application queries
  - Pro: database audit tools can do this
  - Con: requires process audit logs to translate to data content

- Store data that was delivered
  - Pros: can precisely identify who got what when
  - Con: requires dev, only works with certain tools (ex: restful API)
  - Con: doesn't show what wasn't delivered
Solution #13: Cleanup at Source

Because it's cheaper to clean at the source than downstream

Always be prepared to:

- Clean & scrub data in-route to target database

But always try to:

- give clean-up tasks to source system
Solution #14: Static vs Dynamic, Strong vs Weak Type & Structure

Because this is debated endlessly

Static vs Dynamic Schemas:
- Dynamic Examples: MongoDB, JSON in Postgres, etc
- Dynamic Schemas – optimize for writer – at cost of reader

Static vs Dynamic Typing:
- static typing provide fewer defects*
  - but maybe not better data quality

Declarative Constraints:
- Ex: primary key, foreign key, Uniqueness, and check constraints
  - Code: “ALTER TABLE foo ADD CONSTRAINT ck1 CHECK(open_date <= close_date)”
Solution #15: Data Quality Indicator & Dimension

Example:

- Single id that represents status for multiple fields
- Bitmap example:
  - bitmap 16-bit integer
  - each bit represents a single field
  - bit value of 0 == good, value of 1 == bad
  - Translate integer to field status with UDF or table

<table>
<thead>
<tr>
<th>quality_id</th>
<th>result</th>
<th>col1</th>
<th>col2</th>
<th>col3</th>
<th>col4</th>
<th>col5</th>
<th>col6</th>
<th>col7</th>
<th>col8</th>
<th>colN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>0000000000000001</td>
<td>bad</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>0000000000000010</td>
<td>bad</td>
<td>good</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>0000000000000011</td>
<td>bad</td>
<td>bad</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>0000000000000100</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>0000000000000101</td>
<td>bad</td>
<td>bad</td>
<td>good</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>0000000000000111</td>
<td>bad</td>
<td>bad</td>
<td>bad</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
</tbody>
</table>

Because it allows users to know what is damaged
Solution #16: Generate Test Data

Because production data is of limited usefulness

Various Types:

- **Deterministic:**
  - Contains very consistent data
  - Great for benchmarking
  - cheap to build

- **Realistic:**
  - Produced through a simulation
  - Great for demos
  - Great for sanity-checking analysis
  - Hard to build

- **Extreme:**
  - Contains extreme values
  - Great for bounds-checking
  - cheap to build
Solution #17: Use the Data!

Data unused:
- Will decay over time
- Will lose pieces of data
- Will lose metadata

Data in a report:
- Looks great
- Can see obvious defects
- But doesn't drive, doesn't get tested

Data driving a business process:
- Looks great
- Gets tested & corrected – in order to run the business.
Solution #18: Push Transformations Upstream

Because you don't want to become source implementation experts

From Data Scientists to ETL Developers:
- Eliminates inconsistencies between runs

From ETL Developers to Source System:
- Eliminate unnecessary source system knowledge
- Decouples systems

MOVE

Source

DW / Hadoop / etc

Scientist

Scientist

Scientist

Source

DW / Hadoop / etc

Scientist

Scientist

Scientist
Solution #19: Documentation (Metadata)

Field Metadata:
- Name
- Description
- Type
- Length
- Unknown value
- Case
- Security

Extended Metadata:
- Lineage
- Data Profile
  - Common values/codes
  - Their meaning
  - Their frequency
- Validation rules & results
- Transformation rules

Source Code:
```python
if gender == 'm':
    return 'male'
else:
    return 'female'
```

Report & Tool Documentation:
- Description
- Filtering Rules
- Transformation Rules
Solution #20: Data Defect Tracking

*Because you won't remember why a set of data was bad in 6 months*

Like Bug-Tracking, but for sets of bad data:

- Will explain anomalies later
- Can be used for data annotation
- Is simple, just needs to be used
Bonus Solution #21: Change the Culture (ha)

Because you need support for priorities, resources, and time

Single Most Important thing to do:
- Establish policy of transparency = 90%
- Share data with customers, stakeholders, owners, users

Everything else results from transparency:
- Establish policy of automation
- Establish policy of measuring
- Plus everything we already covered

What doesn't work?
- Ask management to mandate quality
Resources & Thanks

International Association for Information & Data Quality (IAIDQ)
http://www.iqtrainwrecks.com/

Improving Data Warehouse and Business Information Quality, Larry English
The Data Warehouse Institute (TDWI)

1-A Large Scale Study of Programming Languages and Code Quality in Github
# Solution List

<table>
<thead>
<tr>
<th>Solution</th>
<th>Source</th>
<th>ETL</th>
<th>DEST/DW</th>
<th>Consume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. QA</td>
<td></td>
<td>HIGH</td>
<td>LOW</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>2. QC</td>
<td></td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>3. Data Profiling</td>
<td>HIGH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Organize</td>
<td></td>
<td></td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>5. Process Auditing</td>
<td>HIGH</td>
<td></td>
<td>MEDIUM</td>
<td></td>
</tr>
<tr>
<td>6. Full Automation</td>
<td>HIGH</td>
<td></td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>7. Changed Data Capture</td>
<td>HIGH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. MDM</td>
<td>HIGH</td>
<td>HIGH</td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>9. Extrapolate Missing Data</td>
<td>HIGH</td>
<td></td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>10. Crowdsource Cleansing</td>
<td>MEDIUM</td>
<td></td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>11. Keep original values</td>
<td>MEDIUM</td>
<td></td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>12. Keep usage logs</td>
<td></td>
<td>HIGH</td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>13. Cleanup at source</td>
<td>HIGH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. Static vs Dynamic</td>
<td>MEDIUM</td>
<td>LOW</td>
<td>MEDIUM</td>
<td></td>
</tr>
<tr>
<td>15. DQ Dimension</td>
<td></td>
<td></td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
<tr>
<td>16. Generate Test Data</td>
<td>HIGH</td>
<td></td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
<tr>
<td>17. Use the Data</td>
<td></td>
<td></td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
<tr>
<td>18. Push Transforms Upstream</td>
<td>HIGH</td>
<td></td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>19. Documentation</td>
<td>MEDIUM</td>
<td></td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
<tr>
<td>20. Data Defect Tracking</td>
<td></td>
<td></td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>21. Change the Culture</td>
<td>HIGH</td>
<td></td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
</tbody>
</table>