Detecting nulls codes

Théo, Helena, Ioana

March 4, 2021

1 Background

Nulls are an important construct used to represent missing, unknown or unavailable information.

Nulls in relational databases The main need for a special *null* code comes from relational databases, where data is regular, e.g., any tuple $R(a, b)$ needs to have a value for a and one for b; an unknown or unavailable b value is encoded by *null*.

When querying relational databases with SQL, nulls lead to a special semantics [4]:

- Any unary predicate evaluated over a null value leads to a special "third" logical value *unknown*, which is different from *true* and *false*; thus, if $r(5, null) \in R(a, b)$, the predicate $r.b > 0$ evaluates to *unknown*;
- Any binary predicate evaluated on two values, at least one of which is a null, evaluates to *unknown*;
- The answers to an SQL query are only based on tuples for which the query predicates evaluate to *true*. Thus, tuples with nulls in attributes on which the query performs a selection or a join will not contribute to the query result.
- Whether a relational attribute is null can only be determined using the special predicate *is null*.

Relational databases allow to specify the constraint *not null* which enforces that nulls are not allowed in a specific column of a specific table. Similarly, a *key* constraint, which specifies that a certain set of attributes is a key for a relation, implies that those attributes are never null. As long as a table is in a relational database on which such constraints have been specified, the database will prevent the insertion of a tuple that violates this constraint.

Pseudo-nulls in relational databases Despite the availability of the special *null* constant, in some relational tables, one encounters non-null values that play the same role. This is typically due to:

- data entry forms (user interfaces) that perform insufficient checks, and allow, e.g., users to enter "none" in a field where a numeric revenue value was expected;
- user's erroneous use of the interfaces, e.g., when a value needs to be chosen from a predefined set, such as a state of the U.S., users may forget to set it from the menu, leaving the default value which just happens to be the first, e.g., "Alabama" among the US states.

This has lead to the term *disguised nulls* in the literature **Ioana:** Helena mentioned this; find reference: a value that is syntactically not null, but which in fact corresponds to unavailable or unknown information, thus, logically a null.

Nulls in other tabular data Any tabular formats (in particular CSV and spreadsheets) tend to lead to nulls by the same mechanism as the relational databases: if a certain record (tuple) has no known value for a certain attribute (column), a null value is needed. However, given that a CSV file or a spreadsheet are not stored within a database, they do not benefit from a predefined, well-formalized value such as the SQL database null. Thus, in these formats, any logical null value will be translated into a null code.

Nulls in semistructured data A main praised advantage of a semistructured data model, the first of which was **OEM** (the Object Exchange Model), was to eliminate the need for nulls: when something is not known, just do not include the respective field/node/attribute [2]. However, in practice, null codes keep popping up.

- **JSON** explicitly supports nulls, with the help of a special value *null*[1].
- **RDF** appears to have explicitly avoided the headaches that come with nulls. There is no modeling construct for representing nulls, and the recommendation of "how to encode missing information" is just: do not have a triple if you do not know the value (object).
- Schema languages were introduced together with **XML**, which allow to state, e.g., that a certain element must have a certain attribute. This automatically brings the need for nulls, and some syntactic conventions have been introduced for this[2].

2 Problem statement

The problem we consider here is: **automatically finding disguised nulls (or null codes) across any data model accepted in ConnectionLens.**

3 Toward null code detection

We can identify some dimensions along which to classify null codes.

Well-typed vs. wrong-typed null codes In some cases, a null code is a value that has the same type as the non-null ones. This is the case for instance when -1 is used to encode a count (number), whose "normal" (non-null) values are positive. **Ioana:** I'm wondering if we should consider a 0 in e.g., a number of items, or a sum of sales etc. as a null? Probably not? It is also the case of a default value left unchanged by mistake ("Alabama" example above). These are *well-typed* null codes.

In other cases, the type of the null code is different than the expected type: in such cases, the null code is usually a string, such as "None", "N/A" etc. while the expected type can be a string, or a date, a number etc.

Importantly, *wrong-typed null codes can be recognized just from their (wrong) types.*

- If the data has a schema, the expected type is immediately clear. **Ioana:** To what extent does this apply to us?
- Otherwise, data profiling [1] is needed to infer the expected type.

Ioana: Does the data profiling book say something (also) about recognizing nulls?

Same-type nulls can probably be recognized at least by outlier detection techniques [3]. **Ioana:** What are the main techniques that could be used?

References

[1] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. *Data Profiling.* Morgan and Claypool, 2020. Available at `https://team.inria.fr/cedar/files/2021/01/s01057ed1v01y202009hlt047.pdf`.

[1] See `https://www.w3schools.com/js/js_json_datatypes.asp`

[2] See xsi:nil="true" discussed, e.g., here: `https://www.ibm.com/support/knowledgecenter/fr/SS7J8H/com.ibm.odm.dserver.rules.designer.author/designing_bom_topics/tpc_bom_schema_markup_xml.html`.

[2] Joachim Hammer, Hector Garcia-Molina, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995*, page 483. ACM Press, 1995.

[3] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining Concepts and Techniques*. Morgan Kaufmann, 2011. Available at `https://bit.ly/3kJMO17`.

[4] Raghu Ramakhrishnan and Johannes Gehrke. *Database Management Systems (3rd edition)*. McGraw-Hill, 2003. Available at `https://bit.ly/3sQhuAW`. See Chapter 5.6 for null values.