Item 173
git_comments:

1. * * Class to validate that an element exists in another element structure. *
2. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
3. nothing to do
4. * * An element visitor that does an in-place modification of the elements to * fix union-of-one and similar issues. *
5. no changes
6. **comment:** * cleanup unoin-of-one and other similar issues.
   label: code-design
7. noting to do
8. no change
9. result is now set properly

git_commits:

1. **summary:** fix for JENA-1365
   message: fix for JENA-1365

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
2. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)

3. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
4. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
5. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
6. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
7. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
8. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it

generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)

9. **summary:** QueryBuilder can build an invalid union-of-one query.
   description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
10. **summary:** QueryBuilder can build an invalid union-of-one query.
    description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
11. **summary:** QueryBuilder can build an invalid union-of-one query.
    description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
12. **summary:** QueryBuilder can build an invalid union-of-one query.
    description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)
    label: code-design
13. **summary:** QueryBuilder can build an invalid union-of-one query.
    description: From email chain betwen Claude and Andy is the union-of-one still legal during the construction of the query? Its illegal in a query - UNION is {..} UNION {..} UNION {..} The expected usage in the builder is: {noformat} builder.addUnion(<select-type-statement>).addUnion(<select-type-statement>) {noformat} Currently the code builds a union-of-one and adds the next union to it. check to ensure that a union-of-one is not generated in the final result. At a minimum it an error to generate a union of one. So either, throw an expection or generate "{ pattern }" (including inside {}) which is the moral equivalent. IMO The second is nicer. Tat is what I changed the formatter to do so at least it generated something even if it is changing the query a bit (round trip checking will fail but then it fails currently as it can't be parsed at all)

jira_issues_comments:

1. The related formatter change is: https://github.com/apache/jena/commit/f19d8b267e1976f655e2c7509e6c7ce67c0c9496#diff-9aaed90671f0c690e61dc6dcf937cefeL240 when the formatter for UNION should look like: {noformat}

@Override public void visit(ElementUnion el) { if (el.getElements().size() == 1) { // If this is an element of just one, just do it in-place return ; } . . . {noformat}
2. Commit b1408fff588a252e794e5e055f8ebeb9034364d7 in jena's branch refs/heads/master from [~claude@xenei.org] [https://git-wip-us.apache.org/repos/asf?p=jena.git;h=b1408ff] fix for JENA-1365
3. Commit 9093863bff4b436adeb9e502c3b3437af17f7e07 in jena's branch refs/heads/master from [~claude@xenei.org] [https://git-wip-us.apache.org/repos/asf?p=jena.git;h=9093863] fix for JENA-1365
4. Andy mentions that the FormatterElement method for ElementUnion shoudl read. @Override public void visit(ElementUnion el) { if (el.getElements().size() == 1) { // If this is an element of just one, just do it in-place return ; } .... I wonder if this is wise. QueryBuilder managed to build union-of-one queries and nothing generates an error when one is created. If the formatter simply removes it from the display but it still exists in the Element tree that is executed how does a developer/user figure out that something is wrong or where that something is. It is unfortunate that there are no comment lines in SPARQL as then the formatter could output the a message. :(
5. It is not in the contract for {{FormatterElement}} to perform checking. It does not do checking at the moment. {noformat} if (el.getElements().size() == 1) { visitAsGroup(el.getElements().get(0)); return; } {noformat} So it generates a query which will have the correct semantics. A "union of one" is the same as a nested {}-group. {{FormatterElement}} can only guarantee to generate the same AST for legal queries because it is dependent on how the parser generated the query in the first place.
6. Do the jena-querybuilder tests pass if this is changed? The change went in, and then had to be removed just for jena-querybuilder.
7. The tests work with the change to FormatterElement and I will check in the change soon, however I have a question. Should the test not be el.getElements().size() <= 1 to account for union-of-zero? Claude
8. Can add that - in which case the action is do nothing.
9. Commit f3a0f3c566388b6710083ae40a7470f614f10938 in jena's branch refs/heads/master from [~claude@xenei.org] [https://git-wip-us.apache.org/repos/asf?p=jena.git;h=f3a0f3c] update for JENA-369 after fix for JENA-1365
10. Updates completed and tested
11. Closed on Jena 3.4.0 release.