Item 135
git_comments:

git_commits:

1. **summary:** [SPARK-22036][SQL] Decimal multiplication with high precision/scale often returns NULL
   message: [SPARK-22036][SQL] Decimal multiplication with high precision/scale often returns NULL ## What changes were proposed in this pull request? When there is an operation between Decimals and the result is a number which is not representable exactly with the result's precision and scale, Spark is returning `NULL`. This was done to reflect Hive's behavior, but it is against SQL ANSI 2011, which states that "If the result cannot be represented exactly in the result type, then whether it is rounded or truncated is implementation-defined". Moreover, Hive now changed its behavior in order to respect the standard, thanks to HIVE-15331. Therefore, the PR propose to: - update the rules to determine the result precision and scale according to the new Hive's ones introduces in HIVE-15331; - round the result of the operations, when it is not representable exactly with the result's precision and scale, instead of returning `NULL` - introduce a new config `spark.sql.decimalOperations.allowPrecisionLoss` which default to `true` (ie. the new behavior) in order to allow users to switch back to the previous one. Hive behavior reflects SQLServer's one. The only difference is that the precision and scale are adjusted for all the arithmetic operations in Hive, while SQL Server is said to do so only for multiplications and divisions in the documentation. This PR follows Hive's behavior. A more detailed explanation is available here: https://mail-archives.apache.org/mod_mbox/spark-dev/201712.mbox/%3CCAEorWNAJ4TxJR9NBcgSFMD_VxTg8qVxusjP%2BAJP-x%2BJV9zH-yA%40mail.gmail.com%3E. ## How was this patch tested? modified and added UTs. Comparisons with results of Hive and SQLServer. Author: Marco Gaido <marcogaido91@gmail.com> Closes #20023 from mgaido91/SPARK-22036. (cherry picked from commit e28eb431146bcdcaf02a6f6c406ca30920592a6a) Signed-off-by: Wenchen Fan <wenchen@databricks.com>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [SPARK-22036][SQL] Decimal multiplication with high precision/scale often returns NULL
   body: ## What changes were proposed in this pull request? When there is an operation between Decimals and the result is a number which is not representable exactly with the result's precision and scale, Spark is returning `NULL`. This was done to reflect Hive's behavior, but it is against SQL ANSI 2011, which states that "If the result cannot be represented exactly in the result type, then whether it is rounded or truncated is implementation-defined". Moreover, Hive now changed its behavior in order to respect the standard, thanks to HIVE-15331. Therefore, the PR propose to: - update the rules to determine the result precision and scale according to the new Hive's ones introduces in HIVE-15331; - round the result of the operations, when it is not representable exactly with the result's precision and scale, instead of returning `NULL` - introduce a new config `spark.sql.decimalOperations.allowPrecisionLoss` which default to `true` (ie. the new behavior) in order to allow users to switch back to the previous one. Hive behavior reflects SQLServer's one. The only difference is that the precision and scale are adjusted for all the arithmetic operations in Hive, while SQL Server is said to do so only for multiplications and divisions in the documentation. This PR follows Hive's behavior. A more detailed explanation is available here: https://mail-archives.apache.org/mod_mbox/spark-dev/201712.mbox/%3CCAEorWNAJ4TxJR9NBcgSFMD_VxTg8qVxusjP%2BAJP-x%2BJV9zH-yA%40mail.gmail.com%3E. ## How was this patch tested? modified and added UTs. Comparisons with results of Hive and SQLServer.

github_pulls_comments:

1. **[Test build #85116 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/85116/testReport)** for PR 20023 at commit [`3037d4a`](https://github.com/apache/spark/commit/3037d4aa6afc4d7630d86d29b8dd7d7d724cc990). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
2. @cloud-fan @dongjoon-hyun @gatorsmile @rxin @viirya I saw you worked on this files. Maybe you can help reviewing the PR. For further details about the reasons of this PR, please refer to the e-mail I sent on the dev mail list. Thank you.
3. Ideally we should not change behaviors as possible as we can, but since this behavior is from Hive and Hive also changed it, might be OK to follow Hive and also change it? cc @hvanhovell too
4. @cloud-fan yes, Hive changed and most important at the moment we are not compliant with SQL standard. So currently Spark is returning results which are different from Hive and not compliant with SQL standard. This is why I proposed this change.

5. In am generally in favor of following the SQL standard. How about we do this. Let's make the standard behavior the default, and add a flag to revert to the old behavior. This allows us to ease users into the new behavior, and for us it can provide some data points on when we can remove the old behavior. I hope we can remove this for Spark 2.4 or later. At the end of the day it will be a bit more work, as I'd definitely would make an effort to isolate the the two behaviors as much as possible.
6. thanks for looking at this @hvanhovell. The reasons why I didn't introduce a configuration variable for this behavior are: 1. As far as I know, currently there is no way to read reliably a configuration in catalyst; 2. Also in Hive, the behavior was changed without introducing any configuration to switch back to the previous behavior; 3. Many people are complaining about the current Spark behavior in the JIRA and therefore it seems that the previous behavior is neither desired nor useful to users. Let me know if you don't agree with these arguments. Thanks.
7. I don't fully agree :)... 1. You can use `SQLConf.get` for this. Or you can wire up the rules using the `SessionStateBuilder`. 2. I am reluctant to change this for a minor version. I don't think the Hive approach is very good for UX. 3. People are also going to complain if do change it.
8. Thank you for pining me, @mgaido91 . The approach of PR looks good to me. BTW, do we need to borrow more Hive 2.2 test cases?
9. @hvanhovell, as far as 1 is regarded, I was referring to [this comment](https://github.com/apache/spark/pull/19449#pullrequestreview-67789784) and [this PR](https://github.com/apache/spark/pull/18568) where it is explicitly stated that using `SQLConf` here is not safe and it shouldn't be done. Let me know if I am missing something. I am sorry, but I think I haven't fully understood what you meant by > Or you can wire up the rules using the SessionStateBuilder may I kindly ask you if you could elaborate this sentence a bit more? Thank you very much.
10. Thank you for your review @dongjoon-hyun. I think what we can do is add more test to the whitelist in `HiveCompatibilitySuite`, updating them according to HIVE-15331. Were you thinking to this or something different? Thanks.
11. I thought adding more cases into `decimals.sql`. :) But, for now, never mind about that~
12. I think that we should be careful on suddenly changing behavior.
13. Thanks for your efforts! This change needs more careful review and investigation. Could you post the outputs of Oracle, DB2, SQL server and Hive? Are their results consistent?
14. @gatorsmile, please refer to the [e-mail to the dev mail list](https://mail-archives.apache.org/mod_mbox/spark-dev/201712.mbox/%3CCAEorWNAJ4TxJR9NBcgSFMD_VxTg8qVxusjP%2BAJP-x%2BJV9zH-yA%40mail.gmail.com%3E) for further details. I run the script I added to the tests in this PR, the results are: - Hive behaves exactly as Spark after this PR; - SQLServer the same, even though on additions and subtractions it seems to maintain one more precision digit in some cases (I am running SQLServer 2017, since Hive implementation, and therefore this too, are inspired to SQLServer2005, there might have been a small behavior change in this case). Anyway, differently from Hive and Spark it throws an exception in case 3 described in the email (it is compliant to SQL standard, point 3 of the email is out of scope of this PR, I will create another PR for it once we agree on how to handle that case); - Oracle and Postgres have nearly infinite precision. Thus it is nearly impossible to provoke a rounding on them. If we force a precision loss on them (point 3 of the email, out of scope of this PR) they throw an exception (compliant to SQL standard and SQLServer); Here you are the outputs of the queries. **Hive 2.3.0 (same as Spark after PR)** ``` 0: jdbc:hive2://localhost:10000> create table decimals_test(id int, a decimal(38,18), b decimal(38,18)); No rows affected (2.085 seconds) 0: jdbc:hive2://localhost:10000> insert into decimals_test values (1, 100.0, 999.0), (2, 12345.123, 12345.123), (3, 0.1234567891011, 1234.1), (4, 123456789123456789.0, 1.123456789123456789); No rows affected (14.054 seconds) 0: jdbc:hive2://localhost:10000> select id, a+b, a-b, a*b, a/b from decimals_test order by id; +-----+---------------------------------------+---------------------------------------+----------------------------+----------------------------+ | id | _c1 | _c2 | _c3 | _c4 | +-----+---------------------------------------+---------------------------------------+----------------------------+----------------------------+ | 1 | 1099.000000000000000000 | -899.000000000000000000 | 99900.000000 | 0.100100 | | 2 | 24690.246000000000000000 | 0E-17 | 152402061.885129 | 1.000000 | | 3 | 1234.223456789101100000 | -1233.976543210898900000 | 152.358023 | 0.000100 | | 4 | 123456789123456790.123456789123456789 | 123456789123456787.876543210876543211 | 138698367904130467.515623 | 109890109097814272.043109 | +-----+---------------------------------------+---------------------------------------+----------------------------+----------------------------+ ``` **SQLServer 2017** ``` 1> create table decimals_test(id int, a decimal(38,18), b decimal(38,18)); 2> insert into decimals_test values (1, 100.0, 999.0), (2, 12345.123, 12345.123), (3, 0.1234567891011, 1234.1), (4, 123456789123456789.0, 1.123456789123456789); 3> select id, a+b, a-b, a*b, a/b from decimals_test order by id; 4> GO (4 rows affected) id ----------- -- -- --------------------------------------- -- 1 1099.000000000000000000 -899.000000000000000000 99900.000000 .100100 2 24690.246000000000000000 .000000000000000000 152402061.885129 1.000000 3 1234.223456789101100000 -1233.976543210898900000 152.358023 .000100 4 123456789123456790.123456789123456789 123456789123456787.876543210876543211 138698367904130467.515623 109890109097814272.043109 ``` **Postgres and Oracle** ``` postgres=# create table decimals_test(id int, a decimal(38,18), b decimal(38,18)); CREATE TABLE postgres=# insert into decimals_test values (1, 100.0, 999.0), (2, 12345.123, 12345.123), (3, 0.1234567891011, 1234.1), (4, 123456789123456789.0, 1.123456789123456789); INSERT 0 4 postgres=# select id, a+b, a-b, a*b, a/b from decimals_test order by id; id | ?column? | ?column? | ?column? | ?column? ----+---------------------------------------+---

------------------------------------+---+--- 1 | 1099.000000000000000000 | -899.000000000000000000 | 99900.000000000000000000000000000000000000 | 0.100100100100100100100 2 | 24690.246000000000000000 | 0.000000000000000000 | 152402061.885129000000000000000000000000000000 | 1.00000000000000000000 3 | 1234.223456789101100000 | -1233.976543210898900000 | 152.358023429667510000000000000000000000 | 0.000100037913541123085649 4 | 123456789123456790.123456789123456789 | 123456789123456787.876543210876543211 | 138698367904130467.515622620750190521000000000000000000 | 109890109097814272.043109406191131436 (4 rows) ``` **Spark before the PR** ``` scala> sql("create table decimals_test(id int, a decimal(38,18), b decimal(38,18)) using parquet") res0: org.apache.spark.sql.DataFrame = [] scala> sql("insert into decimals_test values (1, 100.0, 999.0), (2, 12345.123, 12345.123), (3, 0.1234567891011, 1234.1), (4, 123456789123456789.0, 1.123456789123456789)") res1: org.apache.spark.sql.DataFrame = [] scala> sql("select id, a+b, a-b, a*b, a/b from decimals_test order by id").show(truncate = false) +---+--+-------------------------------------+-------+-------------------------------------+ |id |(a + b) |(a - b) |(a * b)|(a / b) | +---+-------------------------------------+-------------------------------------+-------+-------------------------------------+ |1 |1099.000000000000000000 |-899.000000000000000000 |null |0.100100100100100100 | |2 |24690.246000000000000000 |0E-18 |null |1.000000000000000000 | |3 |1234.223456789101100000 |-1233.976543210898900000 |null |0.000100037913541123 | |4 |123456789123456790.123456789123456789|123456789123456787.876543210876543211|null |109890109097814272.043109406191131436| +---+-------------------------------------+-------------------------------------+-------+-------------------------------------+ ```

15. In DB2, `a * b` will just stop with an error message: SQL0802N Arithmetic overflow or other arithmetic exception occurred. Thus, it might be more straightforward for us to follow what DB2 does.
16. Regarding the rules for deciding precision and scales, DB2 z/OS also has its own rules: https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/sqlref/src/tpc/db2z_witharithmeticoperators.html Could you compare it with MS SQL Server?
17. it looks like other databases are very careful about precision lose. However, following DB2 and throwing exception is pretty bad for big data applications, but returning null is also bad as it violates SQL standard. A new proposal: can we increase the max decimal precision to 76 and keep max scale as 38? Then we can avoid precision lose IIUC.
18. @gatorsmile I answered to your comments about DB2 in [the e-mail](http://mail-archives.apache.org/mod_mbox/spark-dev/201712.mbox/%3CCAEorWNANs2WXGQEtAiJ9yaKAvtvfPt44HvaF-V9%2BnJ5Ye%3D6XzQ%40mail.gmail.com%3E). @cloud-fan that would help, but not solve the problem. It would just make the problem being generated by bigger numbers. As you can see from [the e-mail](http://mail-archives.apache.org/mod_mbox/spark-dev/201712.mbox/%3CCAEorWNANs2WXGQEtAiJ9yaKAvtvfPt44HvaF-V9%2BnJ5Ye%3D6XzQ%40mail.gmail.com%3E), DB2 behavior is actually in accordance to SQL standards and the other DBs, it just have a smaller maximum precision. And the case of throwing an exception is point 3 of my e-mails and it is out of scope of this PR, because I think we best discuss before which is the right approach in that case and then I can eventually create a PR. Therefore, also DB2 behavior is aligned to the other SQL engines, the SQL standard and Spark is the only one which is currently behaving differently.
19. ``` db2 => create table decimals_test(id int, a decimal(31,18), b decimal(31,18)) DB20000I The SQL command completed successfully. db2 => insert into decimals_test values (1, 2.33, 1.12) DB20000I The SQL command completed successfully. db2 => select a * b from decimals_test 1 --------------------------------- SQL0802N Arithmetic overflow or other arithmetic exception occurred. SQLSTATE=22003 ``` I might not get your point. Above is the result I got. This is your scenario 3 or 2?
20. @gatorsmile that is scenario 3. I will explain you why and after I will do and errata corrige of the summary I did in my last e-mail, because I made a mistake about how DB2 computes the result precision and scale, sorry for that. Anyway, what you showed is an example of point 3 because DB2 computes the result type as `DECIMAL( MIN(31, p1 + p2), MIN(31, s1 + s2) )`. Therefore, in your case the result type was `DECIMAL(31, 31)`. Since your result had more than 0 significant digits, it was out out of the range of the representable values and an overflow exception was thrown. You can reproduce case 2 as follows: ``` db2 => create table decimals_test (id int, a decimal(31,31), b decimal(31,31)) DB20000I The SQL command completed successfully. db2 => insert into decimals_test values(1, 0.12345678912345678912345689, 0.12345678912345678912345) DB20000I The SQL command completed successfully. db2 => select a*b from dd 1 -------------------------------- .0152415787806736785461049526020 ``` As you can see a truncation occurred. Now, let me amend my table to summarize the behavior of the many DBs: 1. **Rules to determine precision and scale** - *Hive, SQLServer (and Spark after the PR)*: I won't include the exact formulas, anyway the relevant part is that in case of precision higher that the maximum value, we use the maximum available value (38) as precision and the maximum between the needed scale (computing according the relevant formula) and a minimum value guaranteed for the scale which is 6. - *DB2*: computes the result type as `DECIMAL( MIN(31, p1 + p2), MIN(31, s1 + s2) )`. - *Postgres and Oracle*: NA - *SQL ANSI 2011*: no indication - *Spark now*: if the precision needed is more than 38, use 38 as precision; use the needed scale without any adjustment. 2. **Behavior in case of precision loss but result in the range of the representable values** - *Oracle, Hive, SQLServer (and Spark after the PR)*: round the result. - *DB2*: truncates the result (and sets a warning flag). - *Postgres*: NA, it has infinite precision... - *SQL ANSI 2011*: either truncate or round the value. - *Spark now*: returns NULL. 3. **Behavior in case of result out of the range of the representable values (i.e overflow)** - *Oracle, DB2, SQLServer*: throw an exception. - *Postgres*: NA, they

have nearly infinite precision... - *SQL ANSI 2011*: an exception should be raised - *Spark now, Hive*: return NULL (for Hive, there is a open ticket to make it compliant to the SQL standard).

21. Thanks for your detailed summary! We do not have a SQLCA. Thus, it is hard for us to send a warning message back like [what DB2 does.] (https://www.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/sqlref/src/tpc/db2z_decimalmultiplication.html). Silently losing the precision looks scary to me. Oracle sounds like following the rule, [`If a value exceeds the precision, then Oracle returns an error. If a value exceeds the scale, then Oracle rounds it.`] (https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/Data-Types.html#GUID-9401BC04-81C4-4CD5-99E7-C5E25C83F608) SQL ANSI 2011 does not document many details. For example, [the result type of DB2's division] (https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/sqlref/src/tpc/db2z_decimaldivision.html) is different from either our existing rule or the rule you changed. The rule you mentioned above about DB2 is just for multiplification. I am not sure whether we can finalize our default type cocersion rule `DecimalPrecision` now. However, for Hive compliance, we can add a new rule after we introduce the new conf `spark.sql.typeCoercion.mode`. See the PR https://github.com/apache/spark/pull/18853 for details. The new behavior will be added if and only if `spark.sql.typeCoercion.mode` is set to `hive`. Could you first help us improve the test cases added in https://github.com/apache/spark/pull/20008 ? Thanks!
22. Thanks for your analysis @gatorsmile. Actually the rule you specified for Oracle is what it uses when casting, rather then when doing arithmetic operations. Yes DB2 has rather different rules to define the output type of operations. Anyway, we can have a behavior practically identical to DB2 by changing the value of `MINIMUM_ADJUSTED_SCALE` to 31. Therefore, I'd propose, instead of using the configuration you pointed out, to use a configuration for the `MINIMUM_ADJUSTED_SCALE`, changing which we can have both the behavior of Hive and SQLServer and the one of DB2. What do you think? The reason why I am suggesting this is that my first concern is not Hive compliance, but SQL standard compliance. Indeed, as you con see from the summary, on point 1 there is not a uniform behavior (but this is OK to SQL standard since it gives freedom). But on point 2 we are the only ones who are not compliant to SQL standard. And having this behavior by default doesn't look the right thing to do IMHO. On point 3, only we and Hive are not compliant. Thus I think also that should be changed. But in that case, we can't use the same flag, because it would be inconsistent. What do you think? I can understand and agree that loosing precision looks scary. But to me returning `NULL` is even more scary if possible: indeed, `NULL` is what should be returned if either if the two operands are `NULL`. Thus queries running on other DBs which relies on this might return very bad result. For instance, let's think to a report where we join a prices table and a sold_product table per country. In this use case, we can assume that if the result is `NULL`, it means that there was no sold product in that country and then coalesce the output of the multiplication to 0. This would work well on any DB but Spark. With my proposal of tuning the `MINIMUM_ADJUSTED_SCALE`, each customer can decide (query by query) how much precision loss they can tolerate. And if we agree to change point 3 behavior to the SQL standard, in case of it is not possible to meet their desires we throw an exception, giving them the choice about what to do: allow more precision loss, change their input data type, etc. etc. This is the safer way IMHO. I would ne happy to help improving test cases. May I just kindly ask you how you meant to do that? What would you like to be tested more? Would you like me to add more test cases in scope of this PR or to open a new one for that? Thank you for your time reading my long messages. I just want to take the best choice and give you all the elements I have to decide for the best all together. Thank you.
23. Following ANSI SQL compliance sounds good to me. However, many details are vendor-specific. That means, the query results still varies even if we can be 100% ANSI SQL compliant. To avoid frequently introducing behavior breaking changes, we can also introduce a new mode `strict` for `spark.sql.typeCoercion.mode`. (Hive is also not 100% ANSI SQL compliant) Instead of inventing a completely new one, we can try to follow one of the mainstream open-source databases. For example, Postgres. Before introducing the new mode, we first need to understand the difference between Spark SQL and the other. That is the reason why we need to write the test cases first. Then, we can run them against different systems. This PR clearly shows the current test cases do not cover the scenarios of 2 and 3.
24. Thanks @gatorsmile. Then should I create a follow up PR for #20008 in order to cover the cases 2 and 3 before going on with this PR or can we go on with this PR and the test cases added in this PR?
25. The problem found by this PR is just one of the issues that returns NULL when SPARK SQL is unable to process it. Below is another example. I believe we can find more. ```SQL SELECT CAST('a' AS TIMESTAMP) ``` Before we deciding how to fix these issues (one by one or as a whole), we need to do more investigation and identify all of them. We also need to clearly document our current behaviors and then our users can know what is the result they can expect. Yeah! Please go ahead to create a new PR for adding more tests. Thanks!
26. @gatorsmile created #20084 for adding tests. Thanks.
27. I think for big data, throwing exception during runtime is pretty bad. I think it's ok to be incompatible with SQL standard for some cases, and return null. Personally I feel it's ok to follow SQL standard and truncate if precision lose, but we should not follow SQL standard to throw exception when overflow, at least this should not be the default behavior.
28. @cloud-fan thanks for your feedback. Honestly I don't have a string opinion on how we should behave in that case. That's why I wanted some feedbacks from the community on that point, while this PR covers only the other case, ie. rounding in case of precision loss in accordance to Hive and SQLServer's behavior. Anyway, IIUC now this PR is

blocked by #18853 to introduce `spark.sql.typeCoercion.mode` and use that property to decide whether to keep the previous behavior or the one proposed here.

29. hmmm, even we can add many modes like hive mode, postgres mode, etc. we should still make the default/native mode reasonable. It seems more reasonable to truncate the values instead of returning null, in the case of precision lose. @gatorsmile WDYT? Overflow is another story and we should still return null, but introduce a strict mode to throw exception for this case.
30. Sounds OK to me. Add a conf for this behavior change and also document the behavior change in migration section?
31. thanks @gatorsmile, do you have any suggestion for the conf name?
32. **[Test build #85864 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/85864/testReport)** for PR 20023 at commit [`20616fd`](https://github.com/apache/spark/commit/20616fdfc1a75ea9ae0ec531ce72d8c722facb31). * This patch **fails Spark unit tests**. * This patch merges cleanly. * This patch adds no public classes.
33. **[Test build #85863 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/85863/testReport)** for PR 20023 at commit [`6701a54`](https://github.com/apache/spark/commit/6701a54068145994e10b8dd38d9a38a1be1f3674). * This patch **fails Spark unit tests**. * This patch **does not merge cleanly**. * This patch adds no public classes.
34. **[Test build #85914 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/85914/testReport)** for PR 20023 at commit [`ecd6a2a`](https://github.com/apache/spark/commit/ecd6a2a0d01283b0d3a9582c93618ff1c2a772f7). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
35. any more comments @gatorsmile @cloud-fan @dongjoon-hyun @viirya @hvanhovell ?
36. The proposal of this PR is to follow Hive instead of MS SQL Server and DB2. I read the comments in HIVE-15331. Nobody pointed out why Hive decided to introduce new behaviors which are actually inconsistent with MS SQL Server. Looks weird to me. Anybody knew the reason?
37. @gatorsmile as far as I know, Hive implementation is consistent with MS SQL Server 2006 and the implementation is taken by the description of its behavior in MSDN blogs. DB2 implementation I think is not suitable for any big data system, since it is very likely to fail at runtime wasting a lot of computational time in big data applications.
38. @mgaido91 At least it is different from the doc https://docs.microsoft.com/en-us/sql/t-sql/data-types/precision-scale-and-length-transact-sql
39. sorry @gatorsmile, but I can't follow you. Why do you say it is different from the doc? I can't see any difference, sorry. May you explain me please? Thanks.
40. **[Test build #86137 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86137/testReport)** for PR 20023 at commit [`519571d`](https://github.com/apache/spark/commit/519571d5df6cecc9095bb2029c370fb52c9f6b16). * This patch **fails to build**. * This patch merges cleanly. * This patch adds no public classes.
41. **[Test build #86138 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86138/testReport)** for PR 20023 at commit [`1f36cf6`](https://github.com/apache/spark/commit/1f36cf6bcf784a04d77b2b5153cd504e6155c875). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
42. Since this introduces a behavior change, please update the [migration guide of Spark SQL](https://spark.apache.org/docs/latest/sql-programming-guide.html#migration-guide)
43. Update the PR description to explain the related difference between Hive and MS SQL Server and also document the solution this PR uses.
44. @gatorsmile in the migration guide should I put this change in the section Spark 2.2 -> Spark 2.3? Will this change be in Spark 2.3?
45. **[Test build #86182 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86182/testReport)** for PR 20023 at commit [`090659f`](https://github.com/apache/spark/commit/090659fe5f2471462ada0d54c0c855d9fe4aba7e). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
46. **[Test build #86260 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86260/testReport)** for PR 20023 at commit [`cf3b372`](https://github.com/apache/spark/commit/cf3b372ef4d2d4c798bd448f9cf2338269b6ce43). * This patch **fails Spark unit tests**. * This patch merges cleanly. * This patch adds no public classes.
47. the test failure is unrelated. A proof is that I just updated the migration section in the last commit and the previous one was passing all the tests. @cloud-fan @gatorsmile any more comments?
48. Jenkins, retest this please
49. LGTM. One thing we can improve is the golden file test framework. I found we sometimes repeat the test cases with a config on and off. We should write the test cases once and list the configs we wanna try, and ask the test framework to do it. This can be a follow-up. @mgaido91 thanks for your great work!
50. **[Test build #86272 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86272/testReport)** for PR 20023 at commit [`03644fe`](https://github.com/apache/spark/commit/03644fe117f56b745c344beb6ea08af5045c2dbd). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.

51. **[Test build #86276 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86276/testReport)** for PR 20023 at commit [`7653e6d`](https://github.com/apache/spark/commit/7653e6d5c427c80f925a5992b01c80a8f2d58969). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
52. **[Test build #86277 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86277/testReport)** for PR 20023 at commit [`2b66098`](https://github.com/apache/spark/commit/2b6609876aa76da5a9169b11142782613ea39ab7). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
53. **[Test build #86271 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86271/testReport)** for PR 20023 at commit [`cf3b372`](https://github.com/apache/spark/commit/cf3b372ef4d2d4c798bd448f9cf2338269b6ce43). * This patch **fails from timeout after a configured wait of \`300m\`**. * This patch merges cleanly. * This patch adds no public classes.
54. LGTM except a few comments about the doc
55. LGTM, pending jenkins
56. **[Test build #86330 has finished](https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/86330/testReport)** for PR 20023 at commit [`b4b0350`](https://github.com/apache/spark/commit/b4b0350dea09db897b70485ef1fad41a742eae30). * This patch passes all tests. * This patch merges cleanly. * This patch adds no public classes.
57. thanks, merging to master/2.3!

github_pulls_reviews:

1. nit. How about making into one SQL statement? ```sql insert into decimals_test values (1, 100.0, 999.0), (2, 12345.123, 12345.123), (3, 0.1234567891011, 1234.1), (4, 123456789123456789.0, 1.123456789123456789) ```
2. The hyperlinks in the PR came from Microsoft, and the first purpose is SQL compliant. Can we remove this line?
3. Two cases (2 and 3) were mentioned in the email. If this is the only `NULL`-return test case from previous behavior, can we have another test case? ``` Currently, Spark behaves like follows: 1. It follows some rules taken from intial Hive implementation; 2. it returns NULL; 3. it returns NULL. ```
4. The third case is never checked in the current codebase, ie. when we go out of the representable range of values. I haven't added a test for it, because I was waiting for feedbacks by the community about how to handle the 3rd case and I focused this PR only on points 1 and 2. But I can add a test case for it and eventually change it in a future PR to address the 3rd point in the e-mail. Thanks.
5. Use `assert` to make sure assumptions?
6. Is this rule newly introduced?
7. Can't we just use `ShortDecimal`, `IntDecimal`...?
8. Is this different than `forType` if applied on `Literal.dataType`?
9. Shouldn't we also prevent `scale` > `MAX_SCALE`?
10. Sounds like `MAXIMUM_ADJUSTED_SCALE` instead of `MINIMUM_ADJUSTED_SCALE`.
11. Sounds like `Math.min`?
12. Not sure if this blog link can be available for long time.
13. I can add it even though it is not needed... there is no way we can violate those constraints. If you believe it is better to use assert, I will do that.
14. Yes, it is. If we don't introduce this, we have a failure in Hive compatibility tests, because Hive use the exact precision and scale needed by the literals, while we, before this change, were using conservative values for each type. For instance, if we have a `select 123.12345 * 3`, before this change `3` would have been interpreted as `Decimal(10, 0)`, which is the type for integers. After the change, `3` would become `Decimal(1, 0)`, as Hive does. This prevents from needing more precision that what is actually needed.
15. yes, please see my comment above for an example. Thanks.
16. No, please see my comments above.
17. this is prevented outside this function.
18. It is the `MINIMUM_ADJUSTED_SCALE`. We can't have a scale lower that that, even though we would need not to loose precision. Please see the comments above.
19. It is `max` because we take either the scale which would prevent a loss of "space" for `intDigits`, ie. the part on the left of the dot, or the `minScaleValue`, which is the scale we are ensuring to provide at least.
20. > We can't have a scale lower that that... Don't you get a scale lower than `MINIMUM_ADJUSTED_SCALE` by `Math.min(scale, MINIMUM_ADJUSTED_SCALE)`?
21. Yes, sorry, my answer was very poor, I will rephrase. `scale` contains the scale which we need to represent the values without any precision loss. What we are doing here is saying that the lower bound for the scale is either the scale that we need to correctly represent the value or the `MINIMUM_ADJUSTED_SCALE`. After this, in the line below we state that the scale we will use is the max between the number of digits of the precision we don't need on the left of the dot and this `minScaleValue`: ie. even though in some cases we might need a scale higher than `MINIMUM_ADJUSTED_SCALE`, but the number of digits needed on the left on the dot would force us to have a scale lower than `MINIMUM_ADJUSTED_SCALE`, we enforce that we will maintain at least

`MINIMUM_ADJUSTED_SCALE`. We can't let the scale be lower that this threshold, even though it would be needed to enforce that we don't loose digits on the left of the dot. Please refer also to the blog post I linked in the comment above for further (hopefully better) explanation.

22. Please remove the web link to the commercial products.
23. Before naming a conf, I need to understand the rule you are following. https://docs.microsoft.com/en-us/sql/t-sql/data-types/precision-scale-and-length-transact-sql The SQL Server only applies `MINIMUM_ADJUSTED_SCALE` for multiplication and division. However, in your impl, you are using it for all the BinaryArithmetic operators?
24. Yes, I followed Hive's implementation which works like this and applies this 6 digits minimum to all operations. This means that SQLServer allows to round more digits than us in those cases, ie. we ensure at least 6 digits for the scale, while SQLServer doesn't.
25. @gatorsmile what about `spark.sql.decimalOperations.mode` which defaults to `native` and accepts also `hive` (and in future also `sql2011` for throwing exception instead of returning NULL)?
26. how about `spark.sql.decimalOperations.allowTruncat`? Let's leave the mode stuff to the type coercion mode.
27. We should make it an internal conf and remove it after some releases.
28. ok, I'll go with that, thanks @cloud-fan.
29. typo? `TRUNCAT` -> `TRUNCATE`?
30. `allowTruncat` -> `allowTruncate`?
31. thanks. This is what @cloud-fan suggested. Actually we are not truncating, but rounding. Personally, I'd prefer `allowPrecisionLoss`, but I am fine with any value. WDYT @cloud-fan @gatorsmile?
32. Sorry that was my typo... `allowPrecisionLoss` SGTM
33. it's better to push some reference about where we get this rule.
34. if scale needs to be less than 6, we would fail the analysis, right?
35. we can put this before the `if`
36. no, we would return the a type with scale 6 and in `CheckOverflow` all the numbers which don't fit will be translated to `NULL`, since this is current Spark behavior in such cases. May I kindly ask you to elaborate a but more what you mean by "push some reference"? Thanks.
37. sorry, typo, "put some reference"... We can put a link to a document of a mainstream RDBMS and say this rule follows xxx...
38. I did, but @gatorsmile told me to remove it: https://github.com/apache/spark/pull/20023#discussion_r159117817.
39. Did any open source RDBMS have this rule?
40. Actually we already referred a commercial RDBMS in L33...
41. Hive has it, but in the documentation it is not explained. And in the comments it just references the blog I referenced, but then removed according to @gatorsmile's comment.
42. This is an example. `adjustPrecisionScale` is also be applied for all the operations. However, the doc shows the adjustment is only applicable to multiplication and division.
43. The logics in this adjustment function is also different from the MS SQL Server docs. > In multiplication and division operations we need precision - scale places to store the integral part of the result. The scale might be reduced using the following rules: > The resulting scale is reduced to min(scale, 38 – (precision-scale)) if the integral part is less than 32, because it cannot be greater than 38 – (precision-scale). Result might be rounded in this case. > The scale will not be changed if it is less than 6 and if the integral part is greater than 32. In this case, overflow error might be raised if it cannot fit into decimal(38, scale) > The scale will be set to 6 if it is greater than 6 and if the integral part is greater than 32. In this case, both integral part and scale would be reduced and resulting type is decimal(38,6). Result might be rounded to 6 decimal places or overflow error will be thrown if integral part cannot fit into 32 digits.
44. sorry, but I think this is exactly the same which is described there. The implementation might seem doing different things but actually the result will be the same. They both take the min between 6 and the desired scale if the precision is not enough to represent the whole scale.
45. yes, that may be a difference indeed. But I think it is a minor one, since 99% of the cases the precision is exceeded only in multiplications and divisions.
46. What if we don't do this? Requiring more precision seems OK as now we allow precision lose.
47. So the rule in document is ``` val resultPrecision = 38 if (intDigits < 32) { // This means scale > 6, as iniDigits = precision - scale and precision > 38 val maxScale = 38 - intDigits val resultScale = min(scale, maxScale) } else { if (scale < 6) { // can't round as scale is already small val resultScale = scale } else { val resltScale = 6 } } ``` I think this is a little different from the current rule ``` val minScaleValue = Math.min(scale, 6) val resultScale = max(38 - intDigits, minScaleValue) ``` Think aboout the case `iniDigits < 32`, SQL server is `min(scale, 38 - intDigits)`, we are `38 - intDigits`
48. @cloud-fan yes, but you have to keep in mind that we are doing so only when precision is > 38. With some simple math (given `intDigits = precision - scale`), SQL server is `min(scale, scale + 38 - precision)`. Since we perform this operation only when precision is greater than 38, the second member is always the minimum. Which means that in such a case, SQL server behaves like us, ie. it takes always `38 - intDigits`. When precision is < than 38, instead we return the input precision and scale, as SQL server does. We are just using the precision instead of the intDigits for the if.

49. if we don't do this we have many test failure in spark-hive, because hive does so. Moreover, requiring more precision is not OK, since it leads to a useless loss of precision. Think of this example: you multiply a column which is DECIMAL(38, 18) by `2`. If you don't do this, `2` is considered a DECIMAL(10, 0). According to the rules, the result should be DECIMAL(38 + 10 + 1, 18), which is out of range: then according to the rules it becomes DECIMAL(38, 7), leading to potentially loosing 11 digits of the fractional part. With this change, instead, the result would be DECIMAL(38 + 1 + 1, 18), which becomes DECIMAL(38, 16), safely having a much lower precision loss.
50. ah i see, makes sense
51. makes sense
52. nit: I feel it's more readable to just put the new cases for literal before these 4 cases.
53. we should make it true by default as it's a more reasonable behavior and follows Hive/SQL standard.
54. remove the above 3 lines
55. use `assert` for assumptions, not comments.
56. This line needs some comments.
57. can we just move these tests to the new `decimal.sql`?
58. unfortunately this is not really feasible since we match on different thigs: here we match on `left.dataType` and `right.dataType`, while for literals we match on `left` and `right`
59. we can do ``` (left, right) match { case (l: Literal, r) => ... case (DecimalType.Expression(p, s), r @ IntergralType()) => ... } ```
60. Yeah, this part is consistent.
61. We need to make a decision. You know, we try our best to keep our rule as stable as possible.
62. Add this example as the code comment?
63. Hive is also doing this?
64. yes, Hive is doing so. That is the reason why I introduced the change (without it, we would have had test failures in spark hive). I will add this in the comment.
65. ``` private[sql] def forType(dataType: DataType): DecimalType = dataType match { case ByteType => ByteDecimal case ShortType => ShortDecimal case IntegerType => IntDecimal case LongType => LongDecimal case FloatType => FloatDecimal case DoubleType => DoubleDecimal } ``` This list is incomplete. Is that possible, the input literal is `Literal(null, NullType)`
66. this problem was present before this PR. Should we fix it here? Is this fix needed? I guess that if it would have been a problem, it would already have been reported.
67. `arithmetic operations between decimals`
68. I'm ok to do the adjustment for all operations, which is same as Hive.
69. nit: `b @ BinaryOperator `
70. We only need to deal with integers? how about float and double?
71. `fromLiteral`?
72. Why create a new test file instead of adding more cases in `decimalArithmeticOperations.sql`?
73. when float and double are involved, the decimal is converted to double
74. because that file was meant for the `typeCoercion` modes (eg. if we introduce a sql2011 mode which throws exception instead of returning NULL), while this is more generic about arithmetic operations' behavior.
75. that file is under `.../typeCoercion/native/`, which is meant for the default behavior(native mode of type coercion). If we introduce a sql2001 mode, we will put a same file under `.../typeCoercion/sql2001/`
76. since we have `forType` I used `forLiteral` to be coherent on the naming
77. I see. I'll merge this file into the other then. Thanks.
78. but we also have `fromDecimal`...
79. Also need to explain what is the previous behavior.
80. We need to explicitly document which arithmetic operations are affected.
81. At least, we need to say, NULL will be returned in this case.
82. Yeah. This is better.
83. This is the new behavior introduced in Hive 2.2. We have to emphasize it.

jira_issues:

1. **summary:** Decimal multiplication with high precision/scale often returns NULL
   description: {noformat} create temporary table dec (a decimal(38,18)); insert into dec values(100.0); hive> select a*a from dec; OK NULL Time taken: 0.165 seconds, Fetched: 1 row(s) {noformat} Looks like the reason is because the result of decimal(38,18) * decimal(38,18) only has 2 digits of precision for integers: {noformat} hive> set hive.explain.user=false; hive> explain select a*a from dec; OK STAGE DEPENDENCIES: Stage-0 is a root stage STAGE PLANS: Stage: Stage-0 Fetch Operator limit: -1 Processor Tree: TableScan alias: dec Select Operator expressions: (a * a) (type: decimal(38,36)) outputColumnNames: _col0 ListSink Time taken: 0.039 seconds, Fetched: 15 row(s) {noformat}

jira_issues_comments:

1. While Hive is using similar precision/scale rules as SQL Server for arithmetic operations (https://msdn.microsoft.com/en-us/library/ms190476.aspx), things seem to be breaking down once we hit the max precision of 38. Looks like this sheds a bit of light on how SQL Server handles things in that case: https://blogs.msdn.microsoft.com/sqlprogrammability/2006/03/29/multiplication-and-division-with-numerics - basically it gives preference to the integer portion rather than the scale (Hive does the opposite), and setting a minimum scale of 6. cc [~xuefuz]
2. There are more than a couple of JIRAs on this topic. I think it can be argued both ways regarding which (int part vs dec part) to favor. As pointed by the blog post, user needs to quality the type matching the actually data. When not possible, use double/float instead. What is missing in Hive is a mode in which error is thrown instead of null, but that's not decimal type specific.
3. Yes the user ideally should qualify the type, but they often don't. And with the way this works now decimal multiplication is pretty much unusable without casts. The precision/scale rules for division look like they have been changed because of similar issues (/HIVE-5866). We could try to match what was done there in multiplication, but I think it would be better to try to emulate the SQL Server behavior given that we were using their precision/scale rules
4. I dont' think this can be made bullet proof. I can always construct an example that shows a rule that fails to accommodate.
5. It cannot be bullet proof - even the Msft blogpost admits this. However I do think that we can do a little better here than the existing behavior, which punts multiplication of any value over 100.
6. Initial patch. Let's see what changes in the golden files.
7. Looks like precommit tests never ran - re-uploading patch.
8. Here are the results of testing the latest attachment: https://issues.apache.org/jira/secure/attachment/12842017/HIVE-15331.2.patch {color:green}SUCCESS:{color} +1 due to 2 test(s) being added or modified. {color:red}ERROR:{color} -1 due to 16 failed/errored test(s), 10775 tests executed *Failed tests:* {noformat} org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[auto_sortmerge_join_2] (batchId=44) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[decimal_precision] (batchId=47) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[decimal_udf] (batchId=8) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample2] (batchId=5) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample4] (batchId=15) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample6] (batchId=61) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample7] (batchId=60) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample9] (batchId=38) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[vector_decimal_expressions] (batchId=48) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[vector_decimal_precision] (batchId=45) org.apache.hadoop.hive.cli.TestMiniLlapCliDriver.testCliDriver[transform_ppr2] (batchId=134) org.apache.hadoop.hive.cli.TestMiniLlapLocalCliDriver.testCliDriver[stats_based_fetch_decision] (batchId=150) org.apache.hadoop.hive.cli.TestMiniLlapLocalCliDriver.testCliDriver[vector_decimal_expressions] (batchId=146) org.apache.hadoop.hive.cli.TestMiniLlapLocalCliDriver.testCliDriver[vector_decimal_precision] (batchId=146) org.apache.hadoop.hive.cli.TestMiniLlapLocalCliDriver.testCliDriver[vector_decimal_udf] (batchId=152) org.apache.hadoop.hive.cli.TestMiniTezCliDriver.testCliDriver[explainanalyze_2] (batchId=92) {noformat} Test results: https://builds.apache.org/job/PreCommit-HIVE-Build/2450/testReport Console output: https://builds.apache.org/job/PreCommit-HIVE-Build/2450/console Test logs: http://104.198.109.242/logs/PreCommit-HIVE-Build-2450/ Messages: {noformat} Executing org.apache.hive.ptest.execution.TestCheckPhase Executing org.apache.hive.ptest.execution.PrepPhase Executing org.apache.hive.ptest.execution.ExecutionPhase Executing org.apache.hive.ptest.execution.ReportingPhase Tests exited with: TestsFailedException: 16 tests failed {noformat} This message is automatically generated. ATTACHMENT ID: 12842017 - PreCommit-HIVE-Build
9. Updating golden files.
10. RB at https://reviews.apache.org/r/54505/
11. Here are the results of testing the latest attachment: https://issues.apache.org/jira/secure/attachment/12842239/HIVE-15331.3.patch {color:green}SUCCESS:{color} +1 due to 2 test(s) being added or modified. {color:red}ERROR:{color} -1 due to 9 failed/errored test(s), 10782 tests executed *Failed tests:* {noformat} org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample2] (batchId=5) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample4] (batchId=15) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample6] (batchId=61) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample7] (batchId=60) org.apache.hadoop.hive.cli.TestCliDriver.testCliDriver[sample9] (batchId=38) org.apache.hadoop.hive.cli.TestMiniLlapCliDriver.testCliDriver[orc_ppd_schema_evol_3a] (batchId=134) org.apache.hadoop.hive.cli.TestMiniLlapCliDriver.testCliDriver[transform_ppr2] (batchId=134) org.apache.hadoop.hive.cli.TestMiniLlapLocalCliDriver.testCliDriver[stats_based_fetch_decision] (batchId=150) org.apache.hadoop.hive.cli.TestMiniTezCliDriver.testCliDriver[explainanalyze_2] (batchId=92) {noformat} Test results: https://builds.apache.org/job/PreCommit-HIVE-Build/2481/testReport Console output: https://builds.apache.org/job/PreCommit-HIVE-Build/2481/console Test logs:

http://104.198.109.242/logs/PreCommit-HIVE-Build-2481/ Messages: {noformat} Executing org.apache.hive.ptest.execution.TestCheckPhase Executing org.apache.hive.ptest.execution.PrepPhase Executing org.apache.hive.ptest.execution.ExecutionPhase Executing org.apache.hive.ptest.execution.ReportingPhase Tests exited with: TestsFailedException: 9 tests failed {noformat} This message is automatically generated. ATTACHMENT ID: 12842239 - PreCommit-HIVE-Build

12. I think this is ready for review now.
13. [~sershe] can you review?
14. Looks good; it seems like it reduces precision even in cases where it's not needed though... is that intended? See out file changes, where last few (non-zero) digits of the result are rounded away
15. Which file in particular? These are all cases where the resulting precision/scale exceeds 38 - for example decimal_precision.q.out, the operation is decimal(20,10) * decimal(20,10), the precision/scale (assuming unlimited precision/scale) would be decimal(41,20), however we're already at the Hive limit of 38. So this is where the updated logic would kick in terms of making the precision/scale fit within 38 digits.
16. Decimal_precision: {noformat} -1234.5678901235 1524157.87532399036884525225 +1234.5678901235 1524157.87532399036884525 {noformat}
17. +1
18. Committed to master