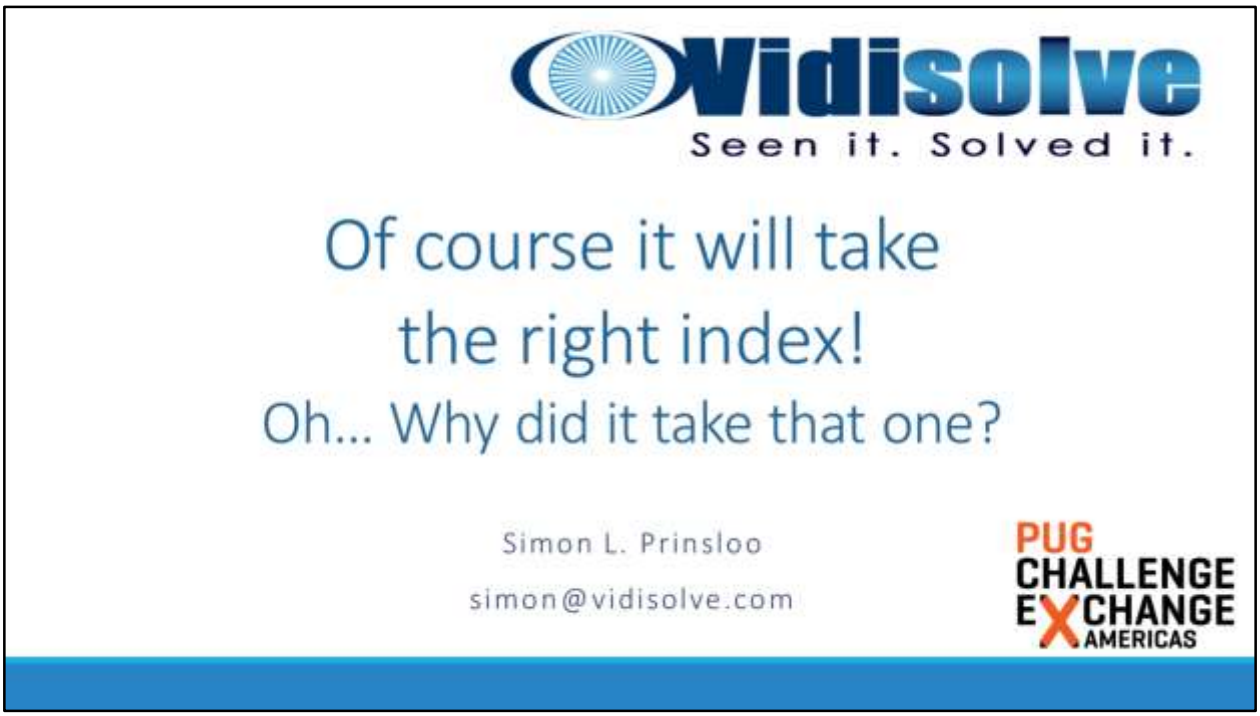


Simon has been working with the Progress 4GL and the database for the last 18 years. Keeping up with the developments in the 4GL and the database has always been his passion.

Simon has been involved in CASE tools since his first introduction to the 4GL and worked on various other systems, including the Namibian Tax System, large ERP systems in Southern Africa and a number of smaller systems.

Progress Software South Africa used Simon at various times to present Progress training courses for version 9 and early version 10. He also presented sessions at three regional Progress conferences in South Africa.

He created Vidisolve in 2011 and started to focus much more actively on the modernisation efforts of his clients. He is currently also busy with his Masters degree in Information Technology at the University of Pretoria.

Introduction

Simon Prinsloo

Vidisolve in Pretoria

Working with Progress since v.7 in 1996

Worked on various commercial systems

Mostly focused on CASE tools and implementing new functionality in legacy projects

Why do we care?

- Hardware is getting bigger and faster
- The compiler selects the index
- It uses a fantastic set of rules
- It will combine indexes

PUG CHALLENGE EXCHANGE AMERICAS

Why do we care?

Knowing the importance of creating indexes to support common data access patterns is a big step toward efficient design. However, to make your query code and indexes work together effectively, you must understand how ABL chooses indexes to satisfy a particular query.

OpenEdge Web Paper: ABL Database Triggers and Indexes

PUG CHALLENGE EXCHANGE AMERICAS

The index selection is rule based. That enables it to be done at compile time and it results in predictable, reproducible index selection. But applying good rules on an abnormal data distribution can have unwanted consequences.

For example, a pipeline of records that needs to be processed can have a logical field indicating whether the record was processed or not. As long as we read the unprocessed records, the set will be very small, but if we read processed records, the set will very quickly be very large.

Terminology

WHERE

WHERE *searchExpr* [BY *field*]

WHERE *searchExpr* AND *searchExpr* [BY *field*]

WHERE *searchExpr* OR *searchExpr* [BY *field*]

Bracketing

WHOLE-INDEX

Sort

SORT-ACCESS

PUG CHALLENGE EXCHANGE AMERICAS

Using multiple index brackets

Multi-bracket queries can be used by

- FOR EACH
- PRESELECT
- QUERY objects

Different brackets can be for the same index or for other indexes.

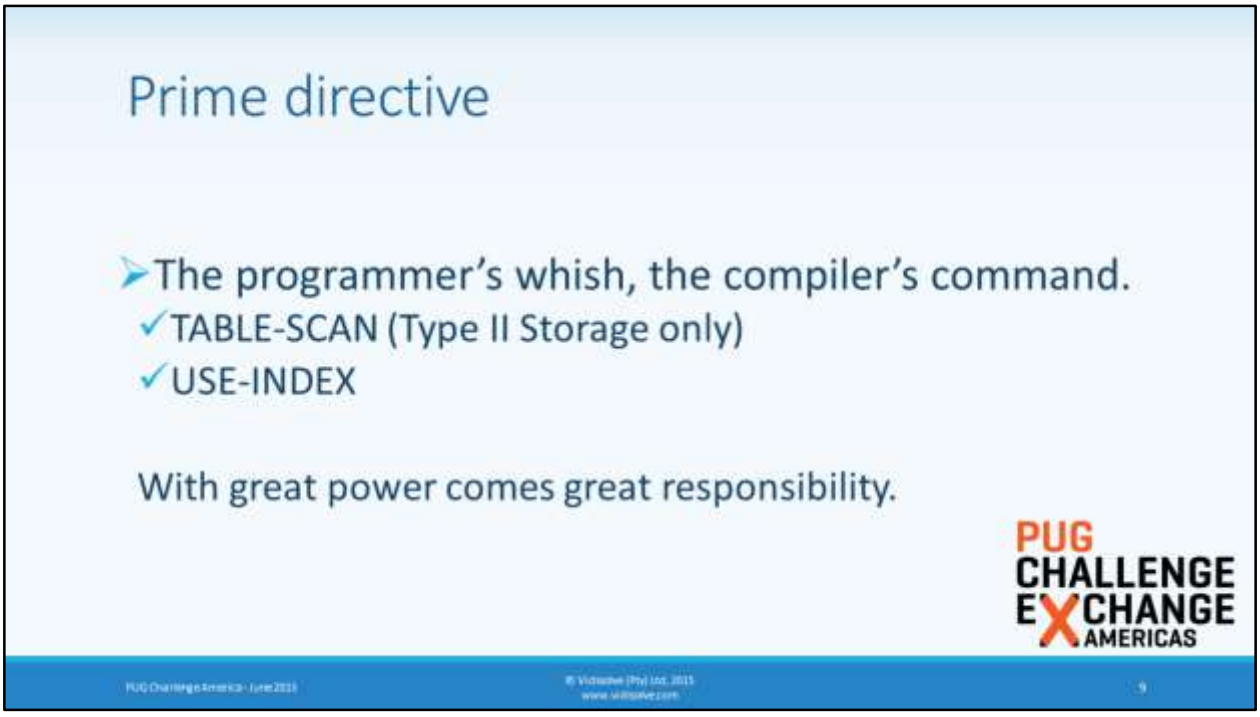
PUG CHALLENGE EXCHANGE AMERICAS

Using single index bracket

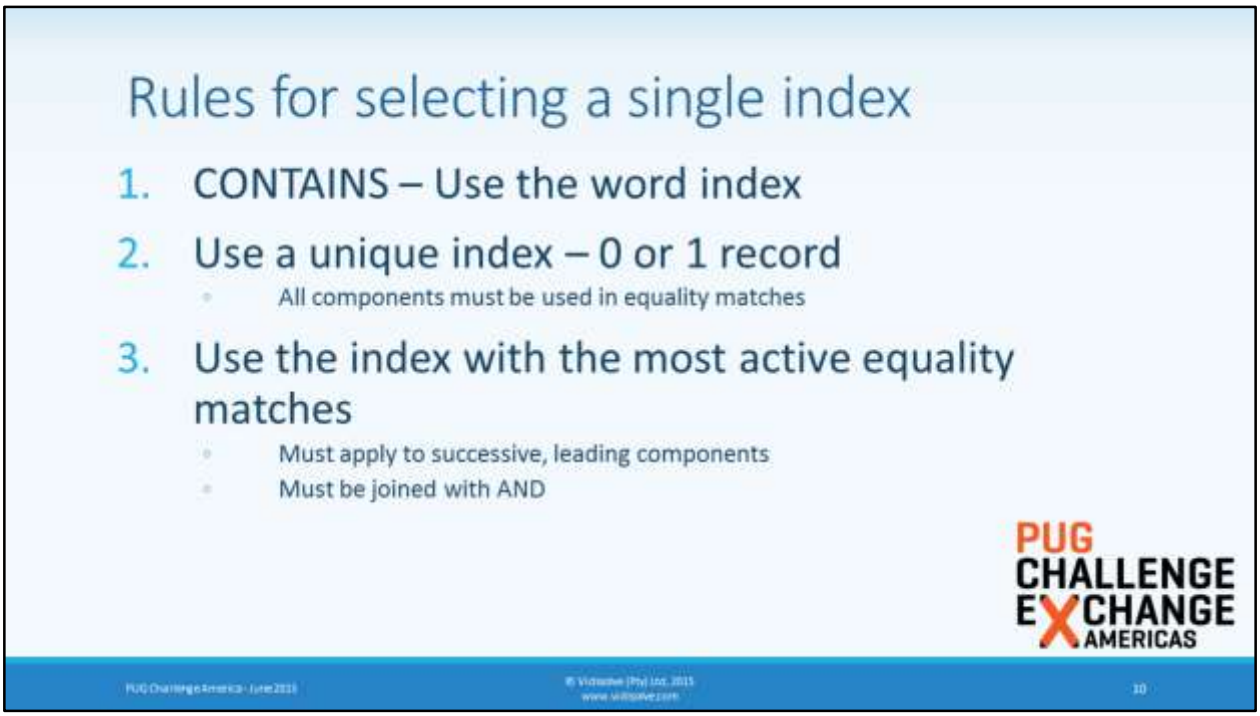
A single index bracket is selected when

- FIND is used
- USE-INDEX is specified
- A condition is encountered that forces a WHOLE-INDEX bracket.

PUG CHALLENGE EXCHANGE AMERICAS



If you do not have a Type II storage area and you use TABLE-SCAN, the DBMS will simply fall back to a full index scan of your primary index.



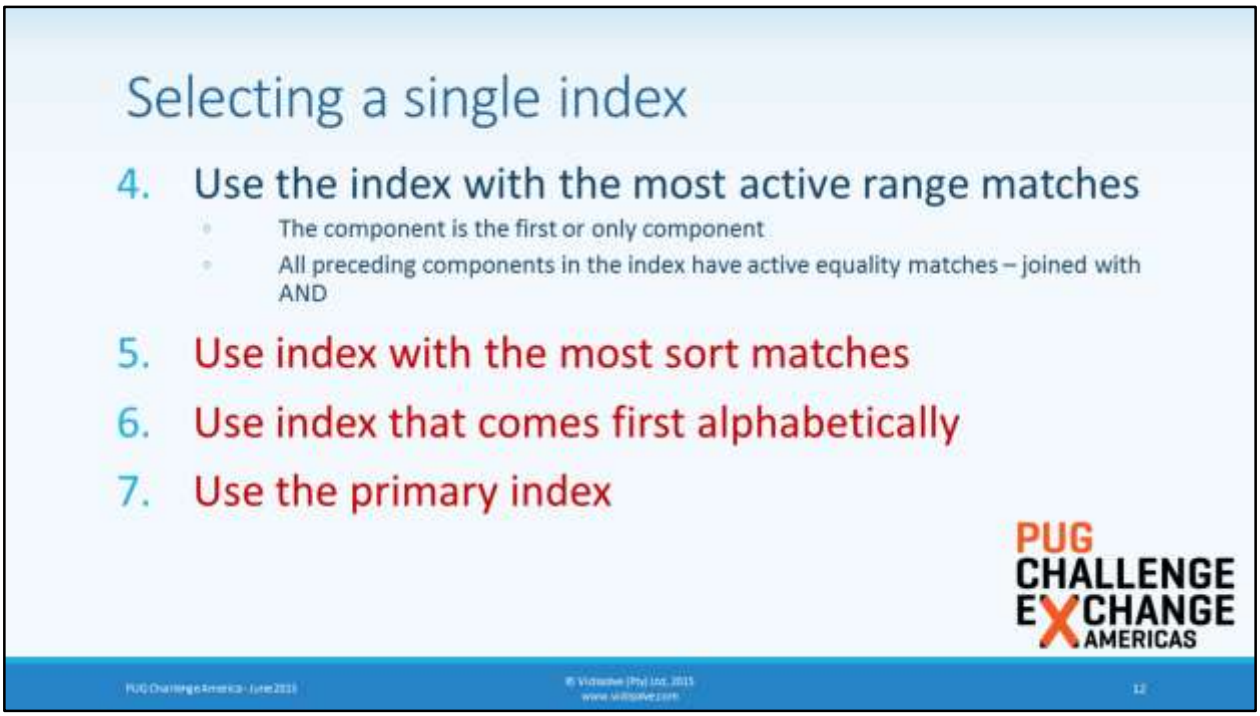
Of course, if there is a USE-INDEX or a TABLE-SCAN, the compiler will not select an index, it will do what it is told.

These rules will also apply top to bottom until only one index is left. In essence, each rule eliminates all indexes that does not fit the rule, but if it would eliminate all indexes, the rule is skipped. If more than one index is left after a rule was applied, we move on to the next rule.

Selecting a single index

4. Use the index with the most active range matches
   - The component is the first or only component
   - All preceding components in the index have active equality matches – joined with AND
5. Use index with the most sort matches
6. Use index that comes first alphabetically
7. Use the primary index

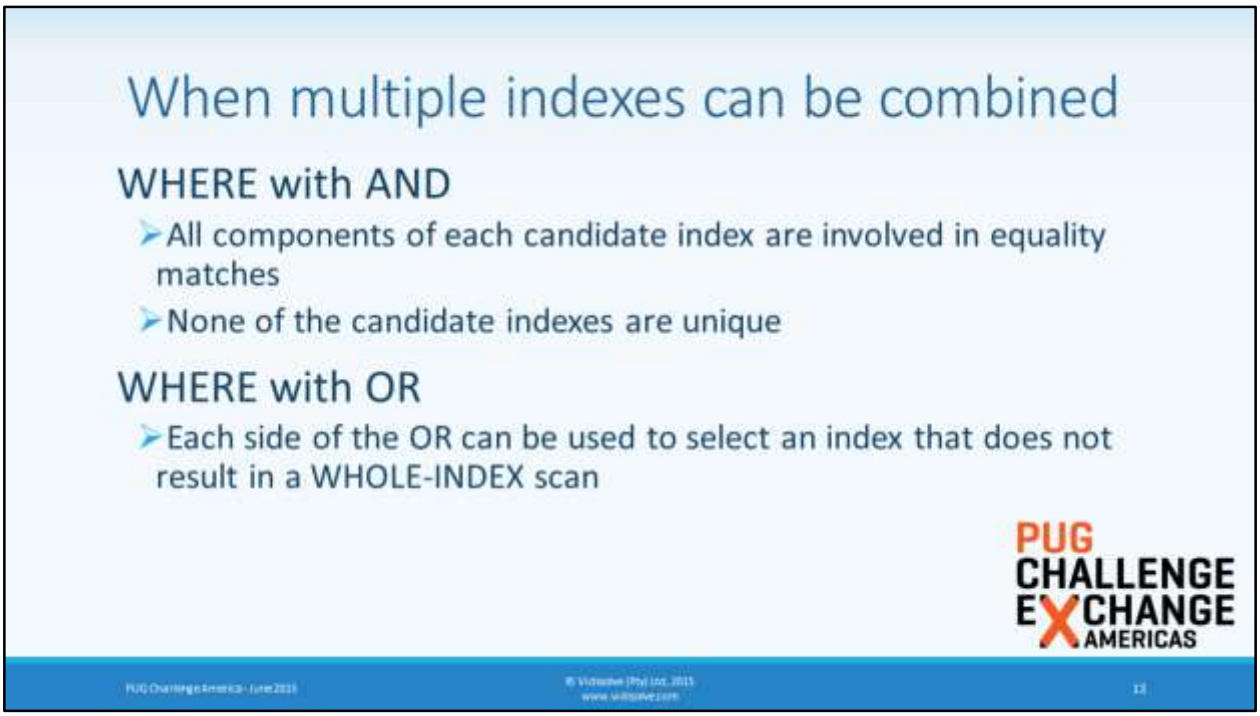
PUG CHALLENGE EXCHANGE AMERICAS



If none of the first four rules are applied to reduce the index set, you are most likely in trouble.

Rule 5 can cause a selection or act as a tie breaker between two or more indexes, in which case it is beneficial. But if it is the sole reason for index selection, it means the whole table will be scanned.

Rule 6 and 7 are essentially mutually exclusive. If two or more suitable indexes remains even after applying rule 5, rule 6 will apply. But if none of the indexes are in the list after rule 5, rule 7 will apply. It is however possible that rule 7 may outrank rule 6 in a case where the Primary index are one of the tied indexes. I did not test this (yet).



The index selection is rule based. That enables it to be done at compile time and it results in predictable, reproducible index selection. But applying good rules on an abnormal data distribution can have unwanted consequences.

For example, a pipeline of records that needs to be processed can have a logical field indicating whether the record was processed or not. As long as we read the unprocessed records, the set will be very small, but if we read processed records, the set will very quickly be very large.

How to find out which index is used

- COMPILE … XREF
  - SEARCH
  - SORT-ACCESS
  - WHOLE-INDEX
- QUERY-HANDLE:INDEX-INFORMATION (n)
  - Comma separated list of index names, one for each bracket
  - First entry can be WHOLE-INDEX

PUG CHALLENGE EXCHANGE AMERICAS

Single-field indexes

glTransaction
- transactionID
- transactionDate
- transactionType

transactionDate + transactionDate

* transactionID + transactionID

transactionType + transactionType

1

glTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

transactionType
+ transactionType

```
FOR EACH glTransaction NO-LOCK
    WHERE transactionType  = "INV"
      AND transactionDate >= 05/01/2015
      AND transactionDate <= 05/31/2015:
    …
END.
```

Which Index?

PUG CHALLENGE EXCHANGE AMERICAS

1

glTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

transactionType
+ transactionType

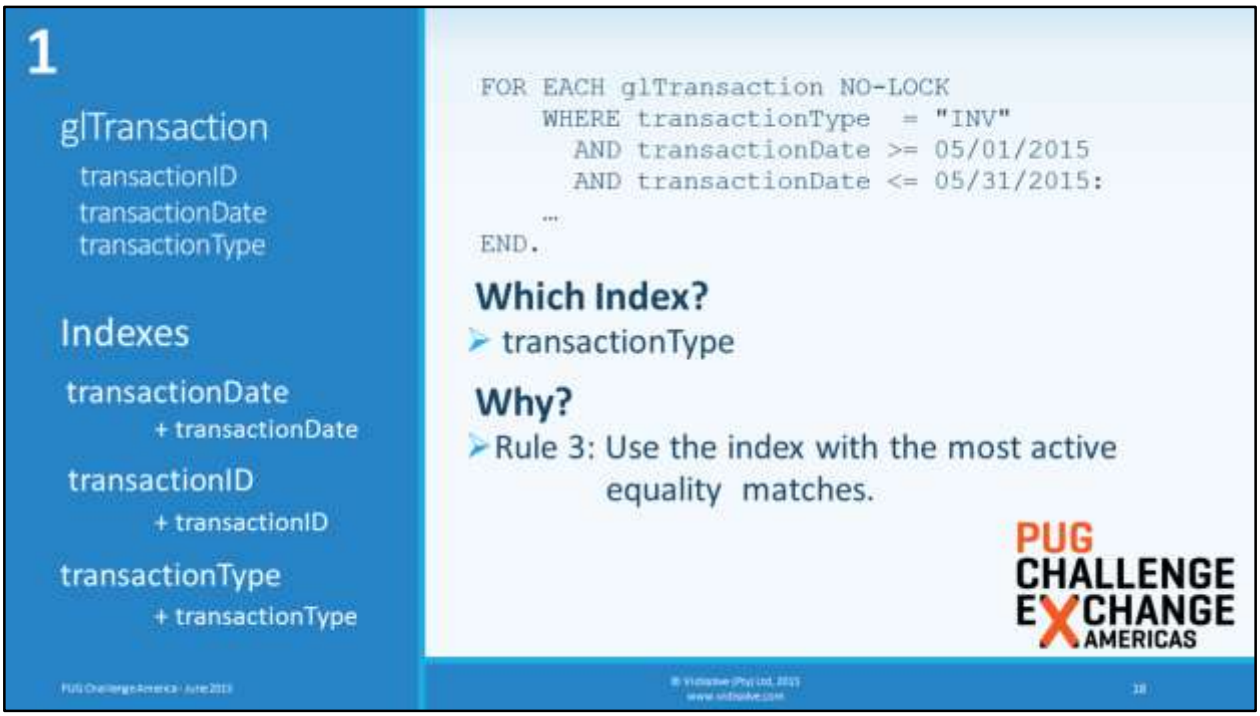
```
FOR EACH glTransaction NO-LOCK
    WHERE transactionType  = "INV"
      AND transactionDate >= 05/01/2015
      AND transactionDate <= 05/31/2015:
    …
END.
```

Which Index?

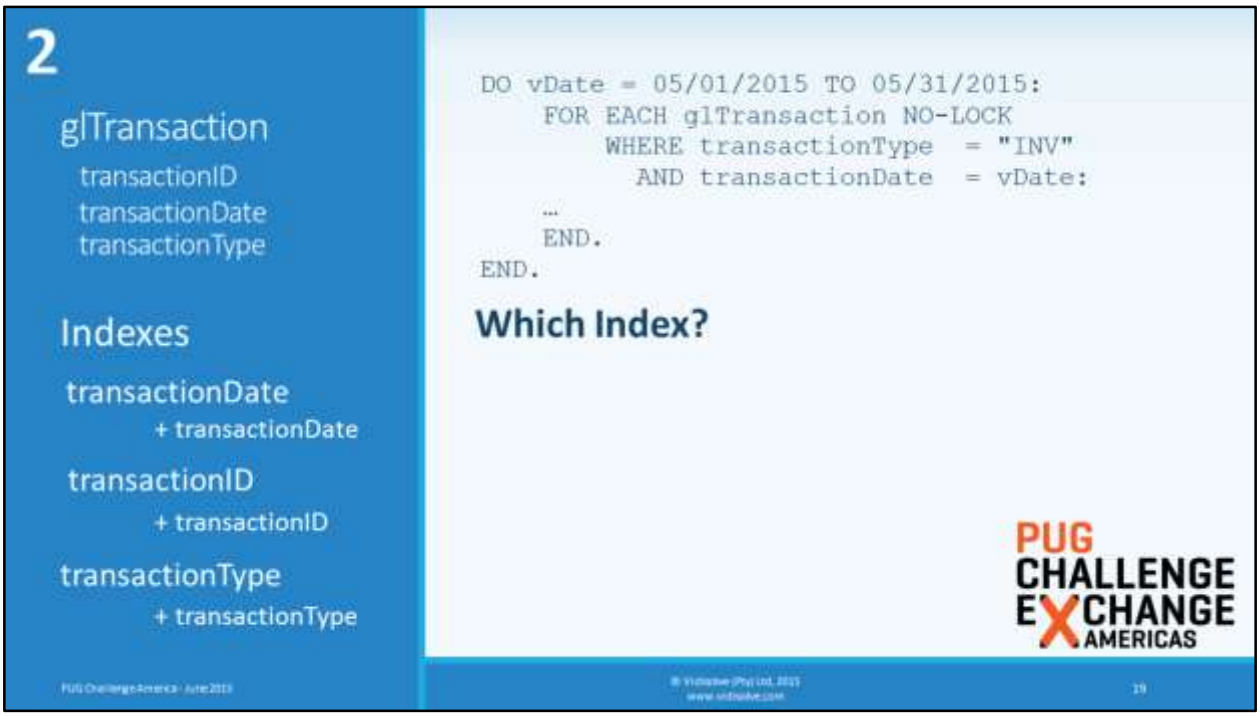
- transactionType

Why?

PUG CHALLENGE EXCHANGE AMERICAS



transactionDate has range matches and will not be considered for the combination of indexes when used as part of an AND.



This is a rather awkward structure, but will it give us some better performance if we need it?

2

glTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

transactionType
+ transactionType

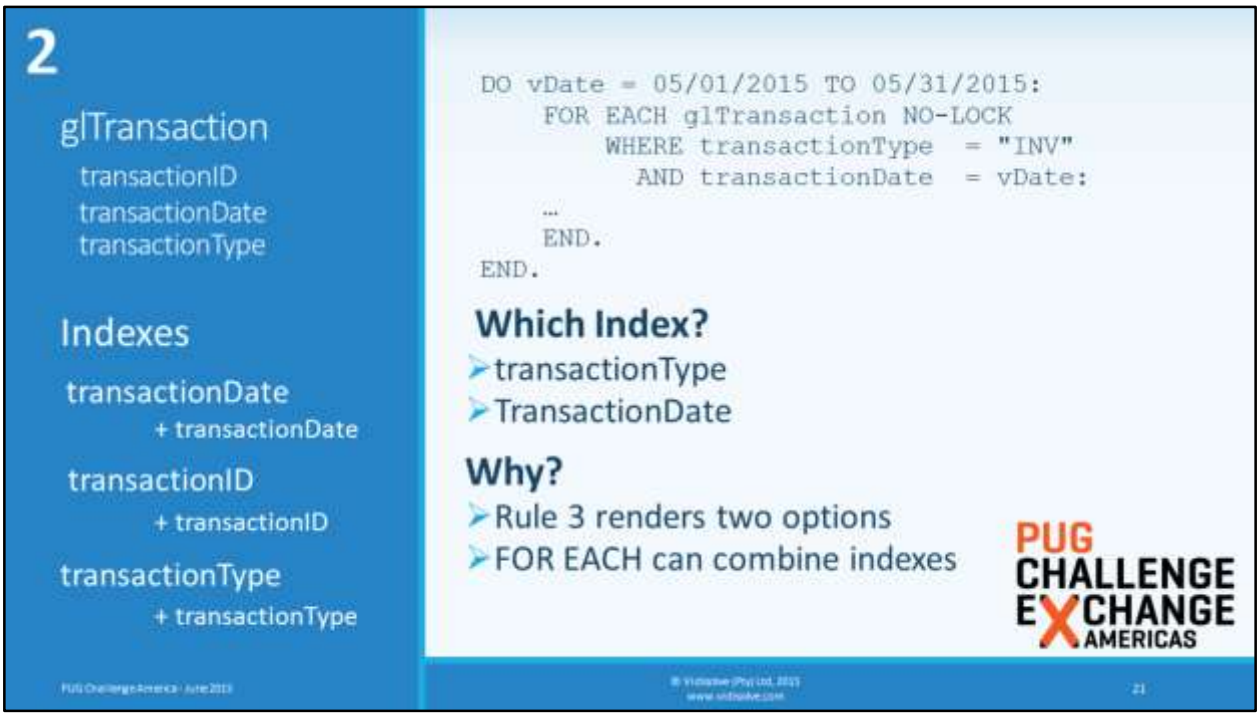
```
DO vDate = 05/01/2015 TO 05/31/2015:
    FOR EACH glTransaction NO-LOCK
        WHERE transactionType  = "INV"
          AND transactionDate  = vDate:
    …
    END.
END.
```

Which Index?

- transactionType
- TransactionDate

Why?

PUG CHALLENGE EXCHANGE AMERICAS



All index components of the two candidate indexes are used in equality matches. For that reason, they can be combined by the FOR EACH.

3

glTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

transactionType
+ transactionType

```
FIND LAST glTransaction NO-LOCK
    WHERE transactionType  = "INV"
      AND transactionDate  = TODAY
    NO-ERROR.
```

Which Index?

PUG CHALLENGE EXCHANGE AMERICAS

3

glTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

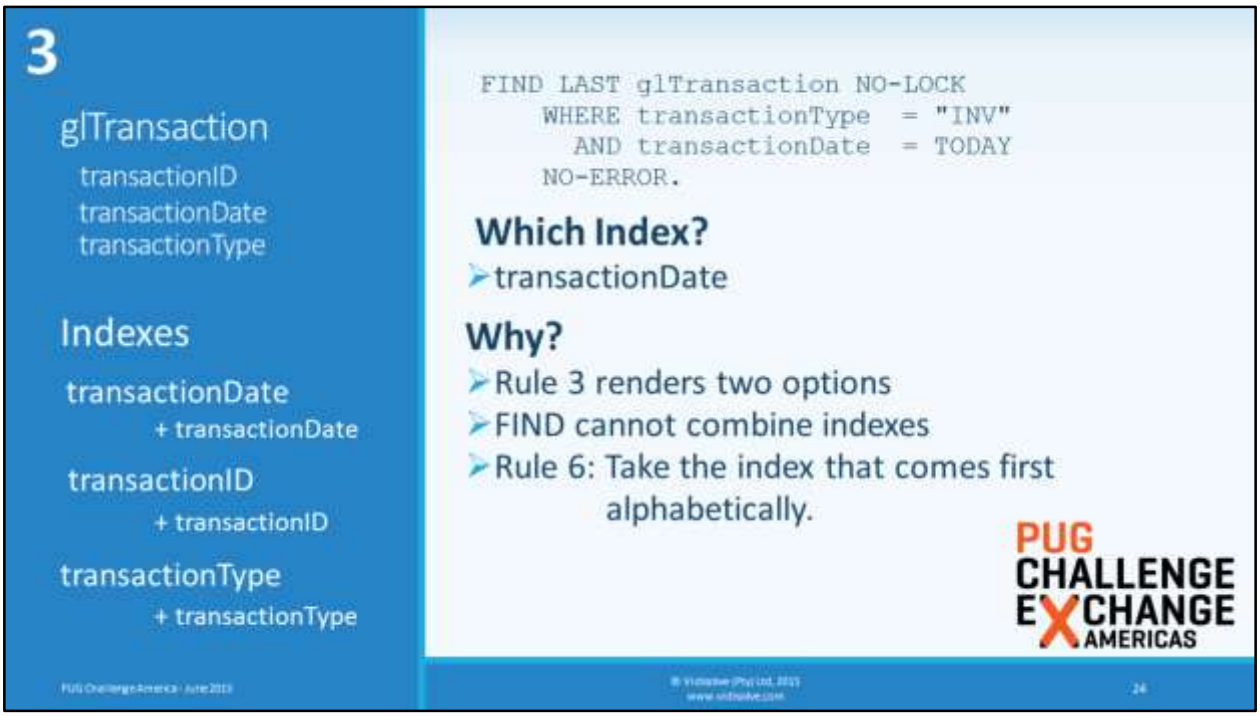
transactionType
+ transactionType

```
FIND LAST glTransaction NO-LOCK
    WHERE transactionType  = "INV"
      AND transactionDate  = TODAY
    NO-ERROR.
```

Which Index?

- transactionDate

Why?



In this case we are lucky. It is pure coincidence that the date index should work better. If it was the other way round, we could have been stuck with reading many more records and dropping most of it.

Compound indexes

glrTransaction

- transactionID
- transactionDate
- transactionType

transactionDate	+ transactionDate + transactionType
* transactionID	+ transactionID
transactionType	+ transactionType + transactionDate

4

glrTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate
+ transactionType

transactionID
+ transactionID

transactionType
+ transactionType
+ transactionDate

```
FOR EACH glrTransaction NO-LOCK
    WHERE transactionType  = "INV"
      AND transactionDate >= 05/01/2015
      AND transactionDate <= 05/31/2015:
    ...
END.
```

Which Index?

PUG CHALLENGE EXCHANGE AMERICAS

4

glrTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate
+ transactionType

transactionID
+ transactionID

transactionType
+ transactionType
+ transactionDate

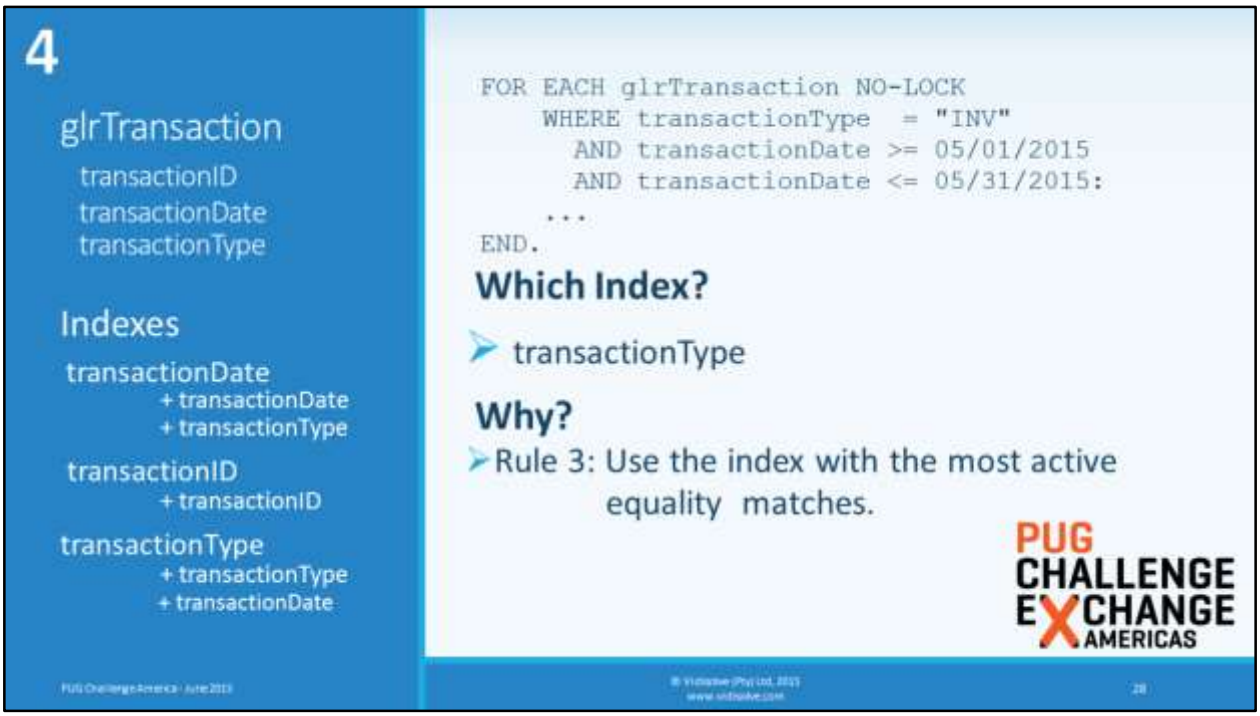
```
FOR EACH glrTransaction NO-LOCK
    WHERE transactionType  = "INV"
      AND transactionDate >= 05/01/2015
      AND transactionDate <= 05/31/2015:
    ...
END.
```

Which Index?

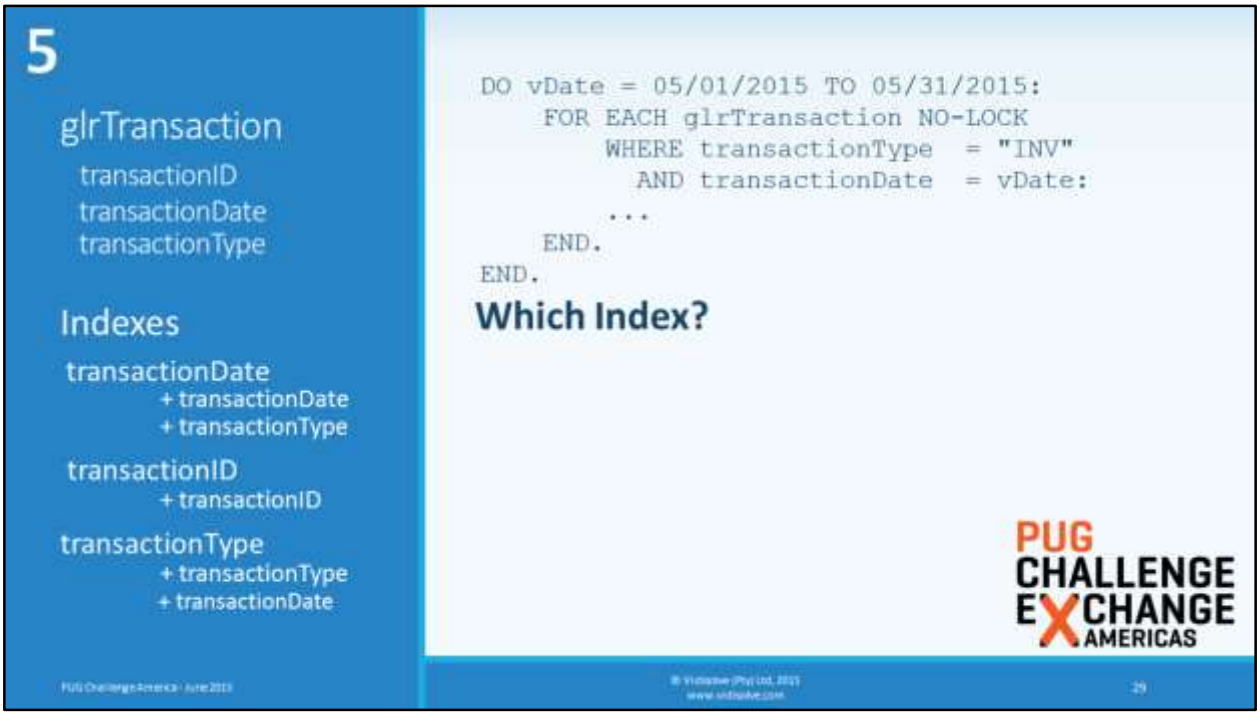
- transactionType

Why?

PUG CHALLENGE EXCHANGE AMERICAS



There is essentially no difference between selection rules for this example and example one, but we will get significantly better performance in this case, as we not only have an equality bracket on the first component of the index, but also have a range match on the second component.



This one is the same example as we had in example two, where both indexes was selected by rule 3 and then combined.

5

glrTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate
+ transactionType

transactionID
+ transactionID

transactionType
+ transactionType
+ transactionDate

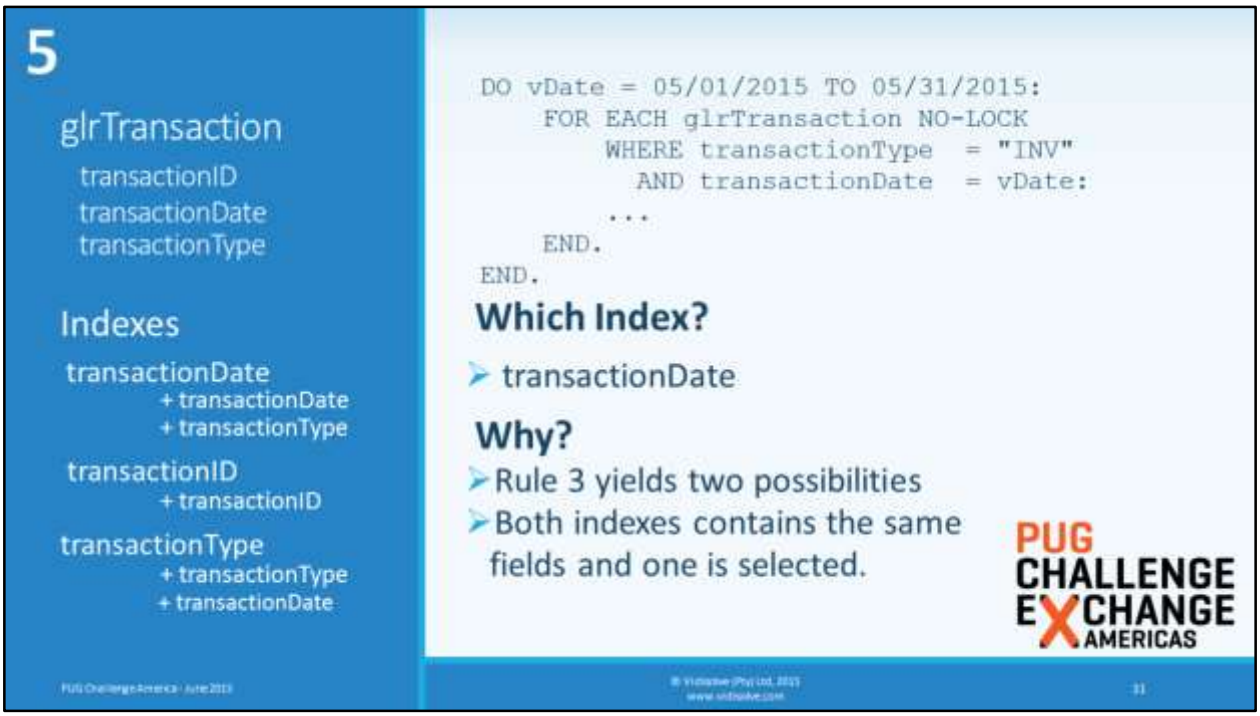
```
DO vDate = 05/01/2015 TO 05/31/2015:
    FOR EACH glrTransaction NO-LOCK
        WHERE transactionType  = "INV"
          AND transactionDate  = vDate:
        ...
    END.
END.
```

Which Index?

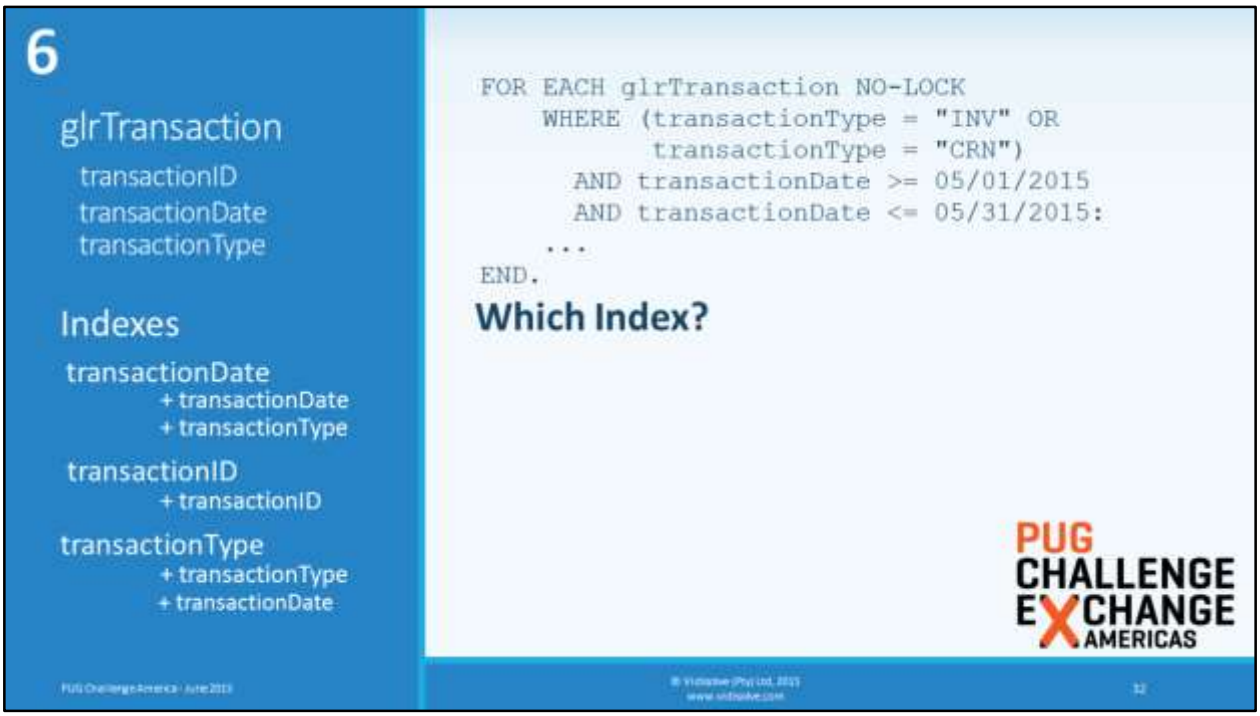
- transactionDate

Why?

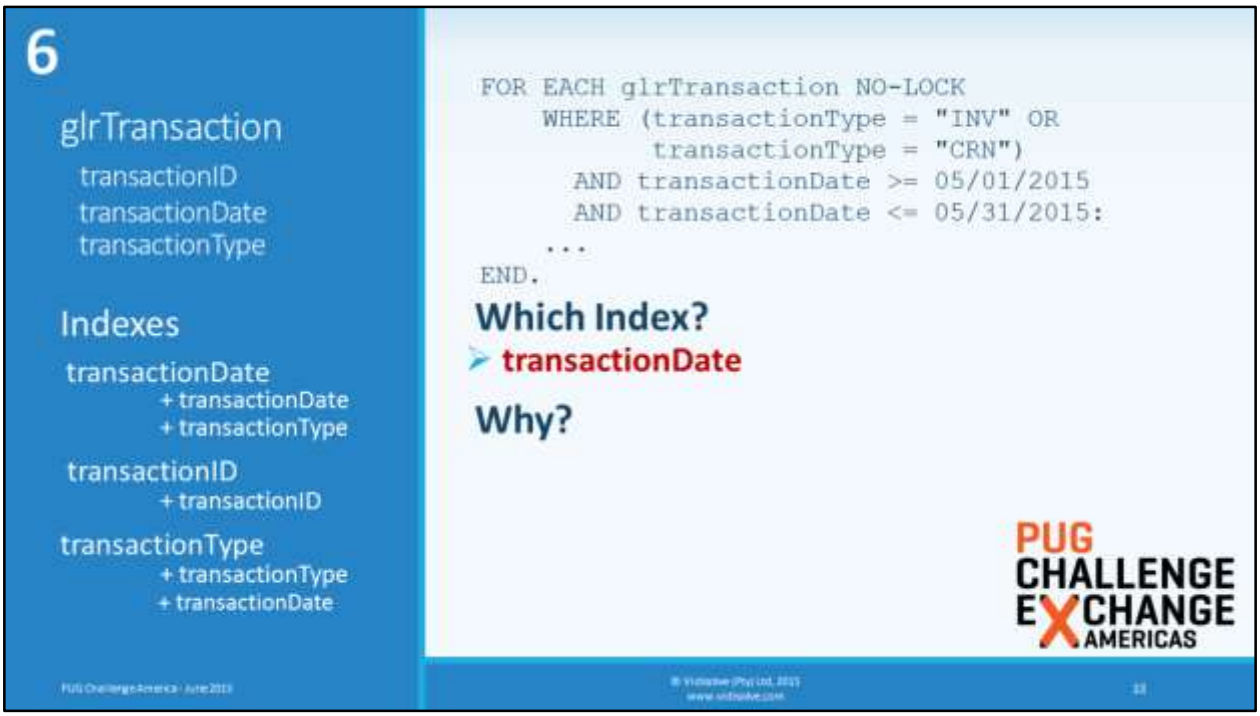
PUG CHALLENGE EXCHANGE AMERICAS



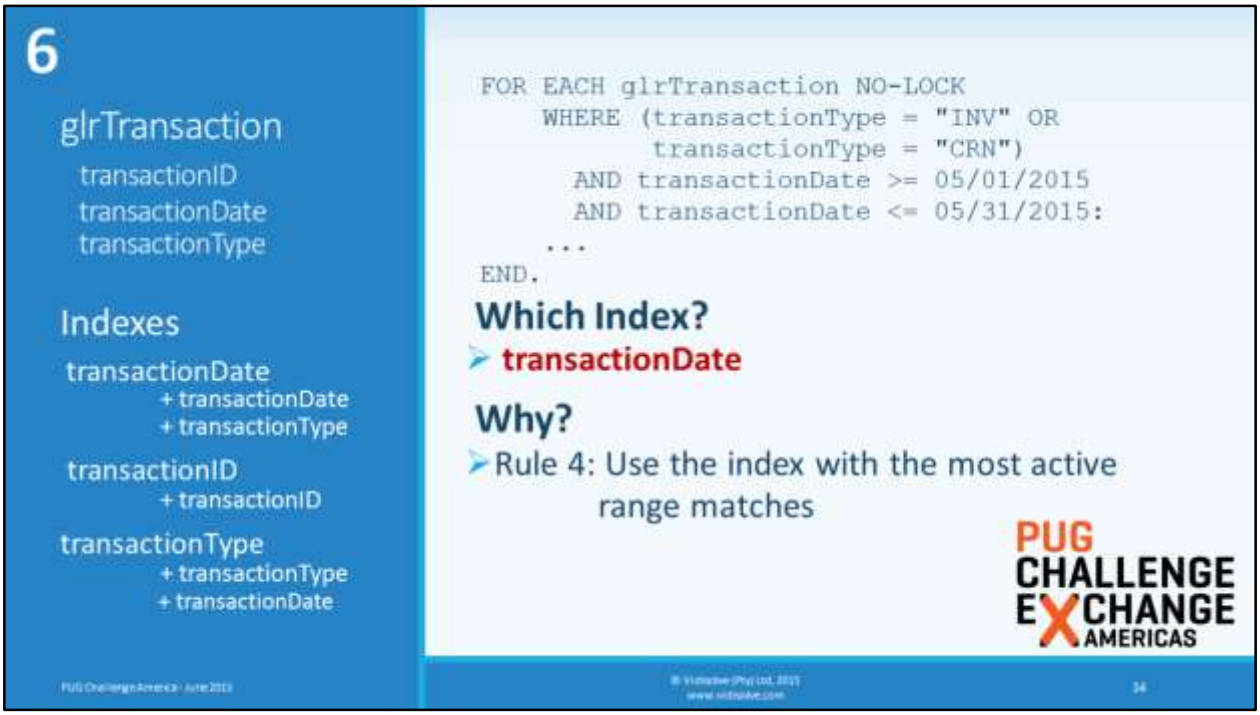
I cannot find a reference to this behaviour in the documentation, but it makes sense, as both indexes would render the exact same list of rowids, albeit in a different sequence. However, this construct is awkward and gains us nothing over example 4.



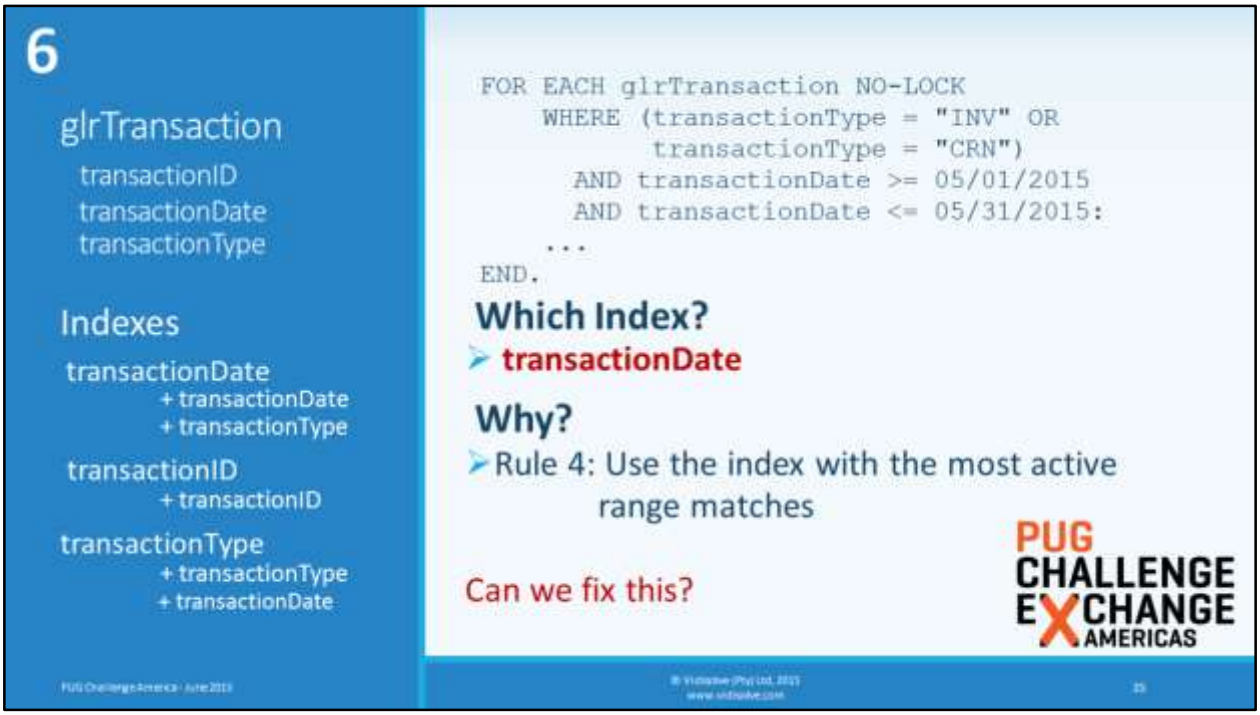
This is basically the same as example 4 – Equality on the type and range on the date, but here we have an OR



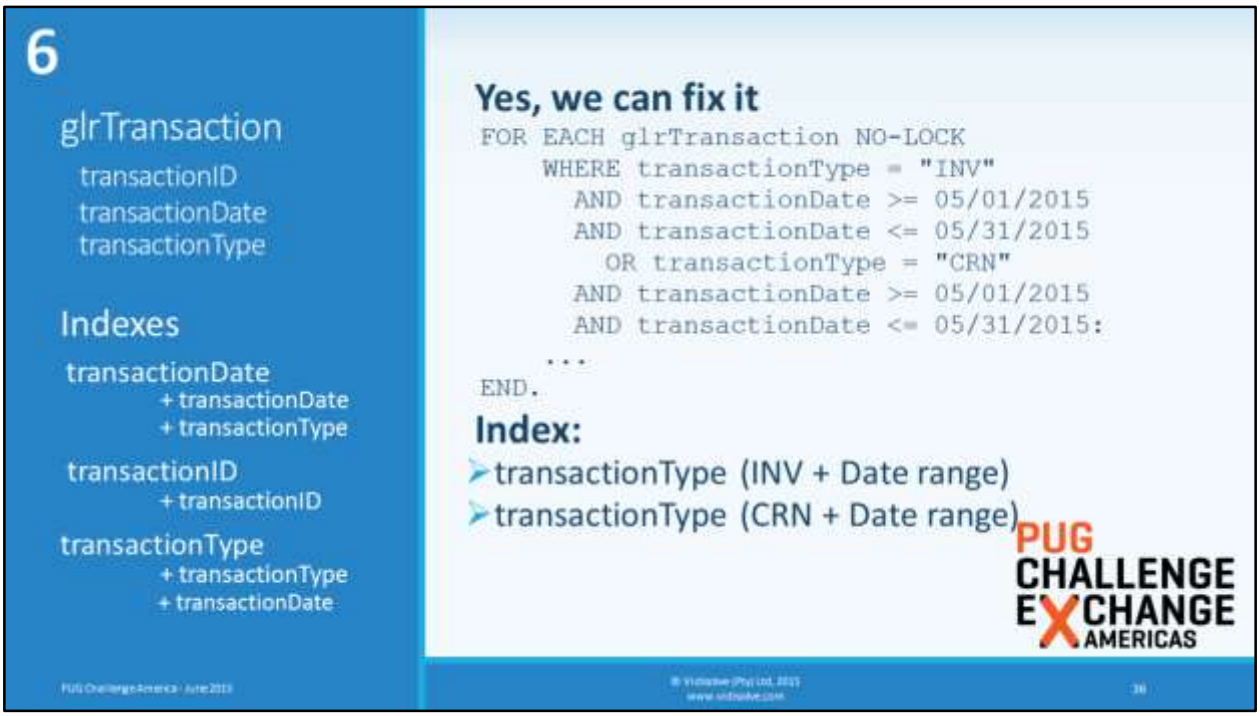
Not a good idea, as it will not leverage the second (transaction type) level of the index.



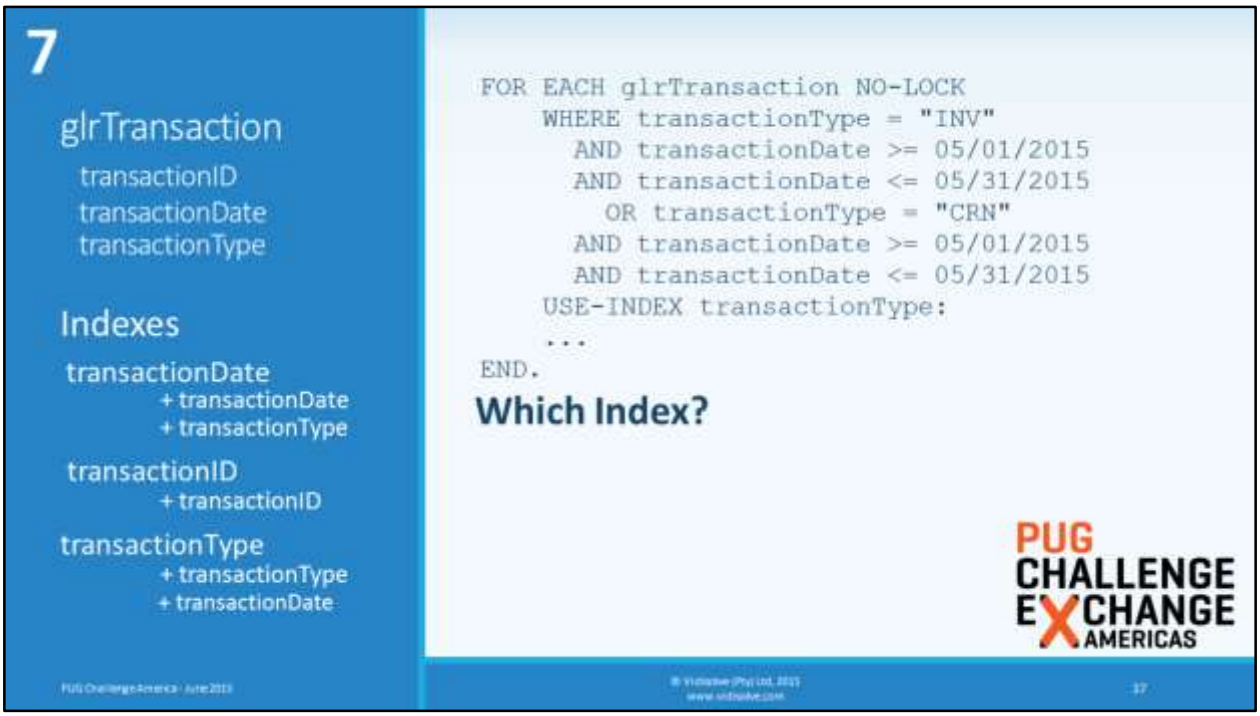
Because the expression inside the () must resolve first, we end up with WHERE clause that essentially boils down to a simple WHERE with AND, but the first one of the three expressions bound by AND is not a straight equality or range match.



With the expected data distribution this is not too terrible, but we do utilize the index in an optimal way.



We now have a WHERE with an OR and each OR fragment can be treated as an independent WHERE with AND.
This leads to the selection of the better index. Two brackets are made on the index and both levels of the index are leveraged.



This is the same as before, is it not? After all, the compiler would have selected the index that we indicate in any case.

7

glrTransaction

transactionID
transactionDate
transactionType

Indexes

transactionDate
+ transactionDate
+ transactionType

transactionID
+ transactionID

transactionType
+ transactionType
+ transactionDate

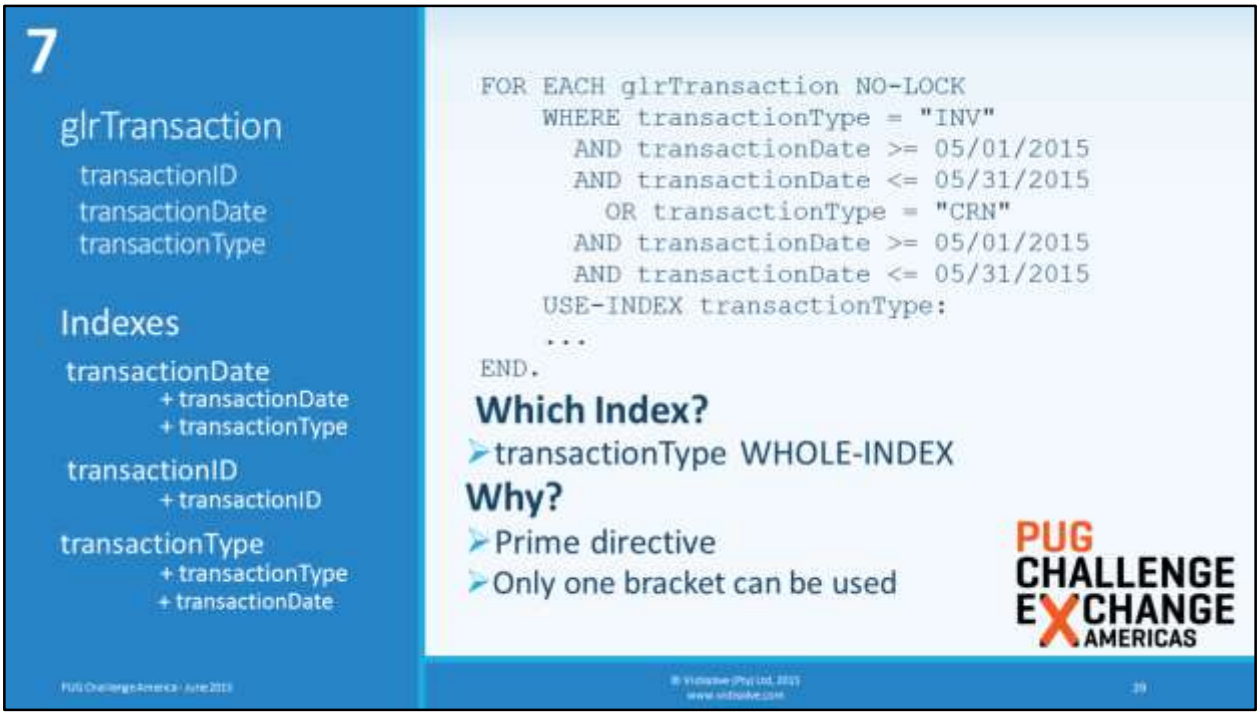
```
FOR EACH glrTransaction NO-LOCK
    WHERE transactionType = "INV"
      AND transactionDate >= 05/01/2015
      AND transactionDate <= 05/31/2015
        OR transactionType = "CRN"
      AND transactionDate >= 05/01/2015
      AND transactionDate <= 05/31/2015
    USE-INDEX transactionType:
    ...
END.
```

Which Index?

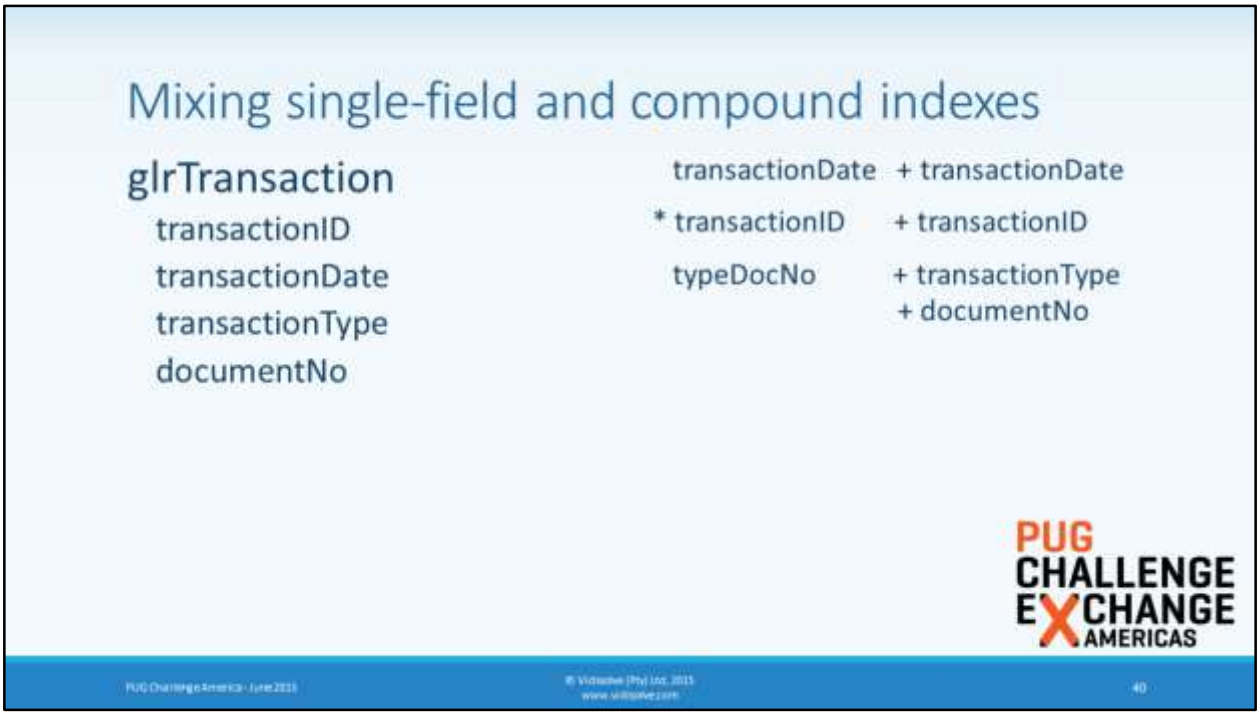
- transactionType WHOLE-INDEX

Why?

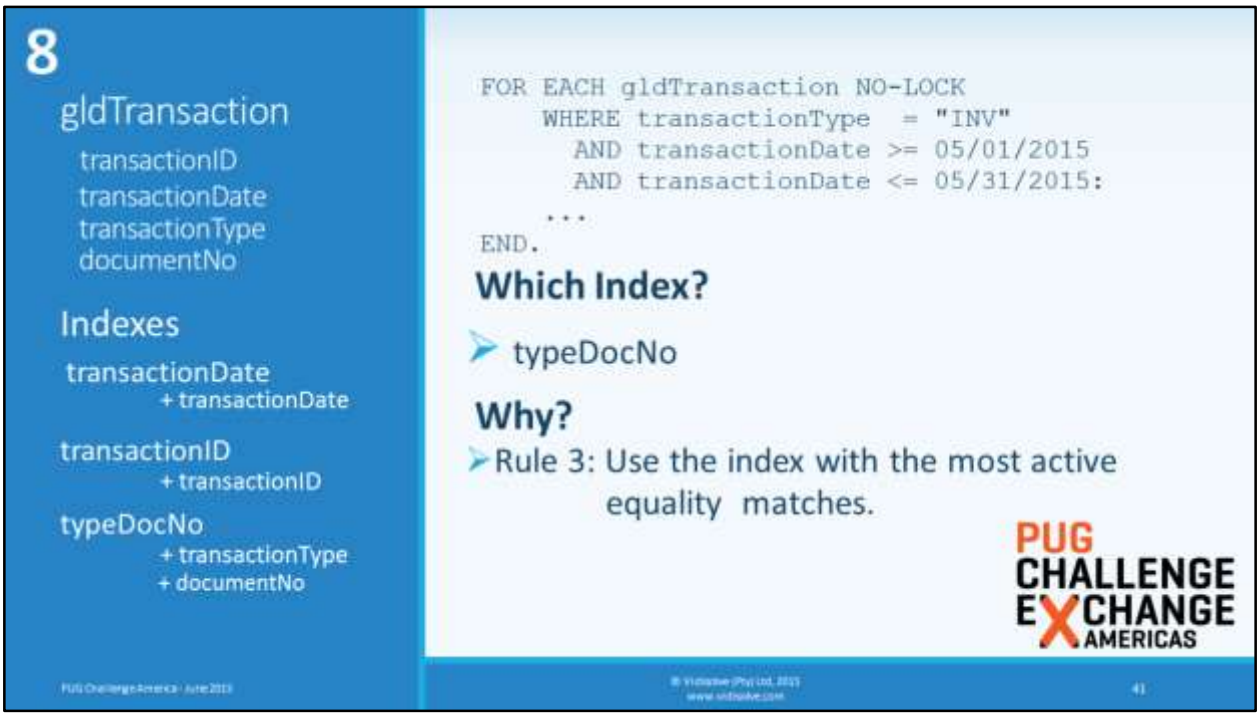
PUG CHALLENGE EXCHANGE AMERICAS



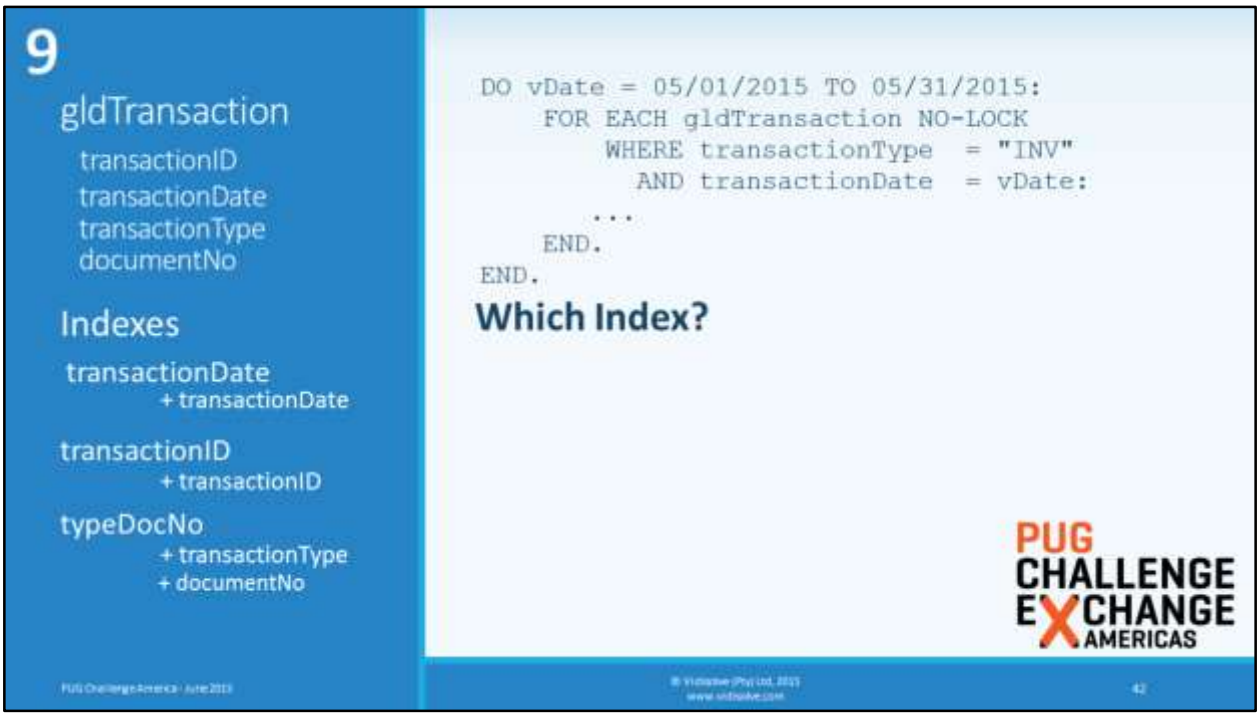
USE-INDEX allows for a single bracket. The only bracket that enclose all the options encompass the entire index. If we did not have the OR, i.e. if we had only the first or the last part, it would still bracket properly and the WHOLE-INDEX will go away.



This set is fairly similar than the first one, except that the index with transactionType has a second component. We will now see how that change some rules.



This is the same as in example 1.



This is essentially the same as the second case, where two single indexes was selected and combined.

9

gldTransaction

transactionID
transactionDate
transactionType
documentNo

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

typeDocNo
+ transactionType
+ documentNo

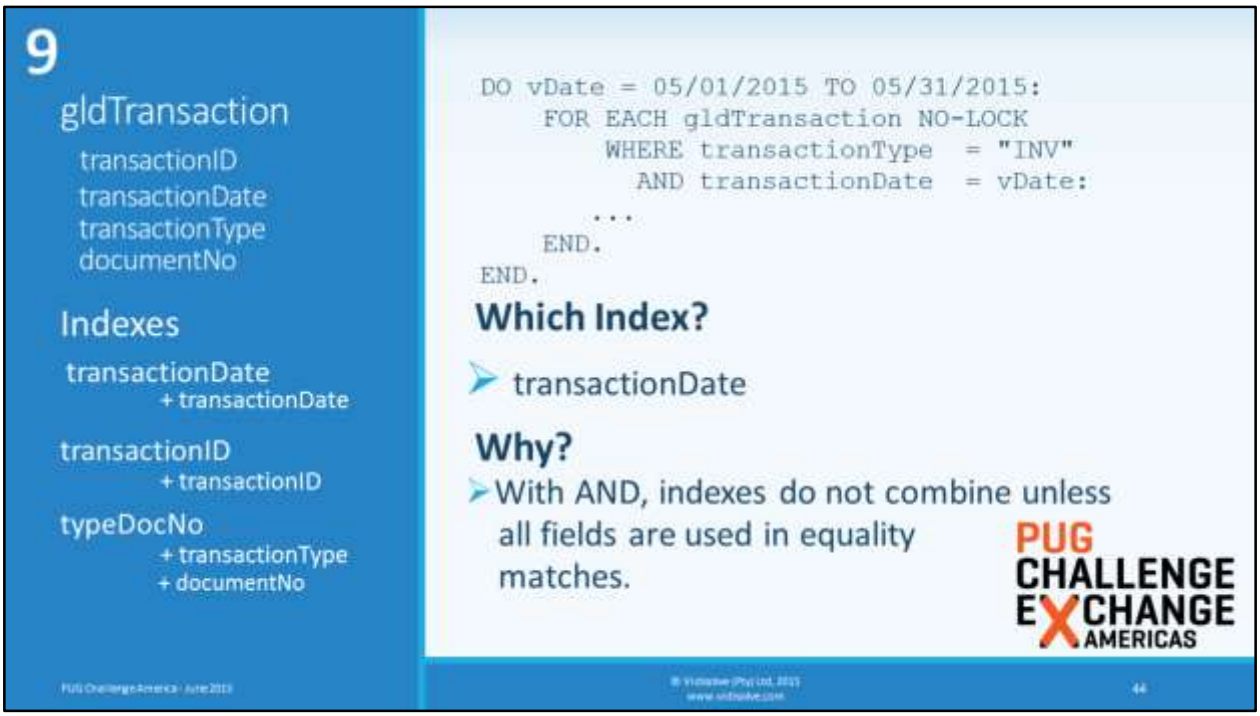
```
DO vDate = 05/01/2015 TO 05/31/2015:
    FOR EACH gldTransaction NO-LOCK
        WHERE transactionType  = "INV"
          AND transactionDate  = vDate:
        ...
    END.
END.
```

Which Index?

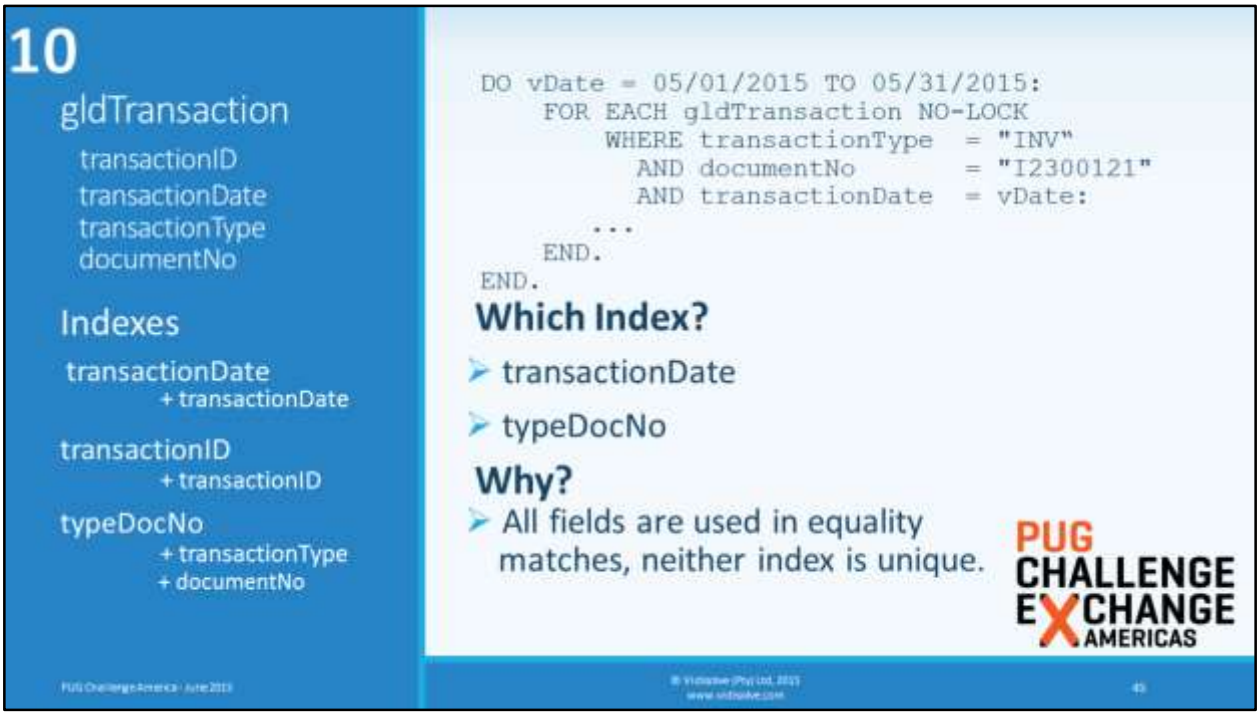
- transactionDate

Why?

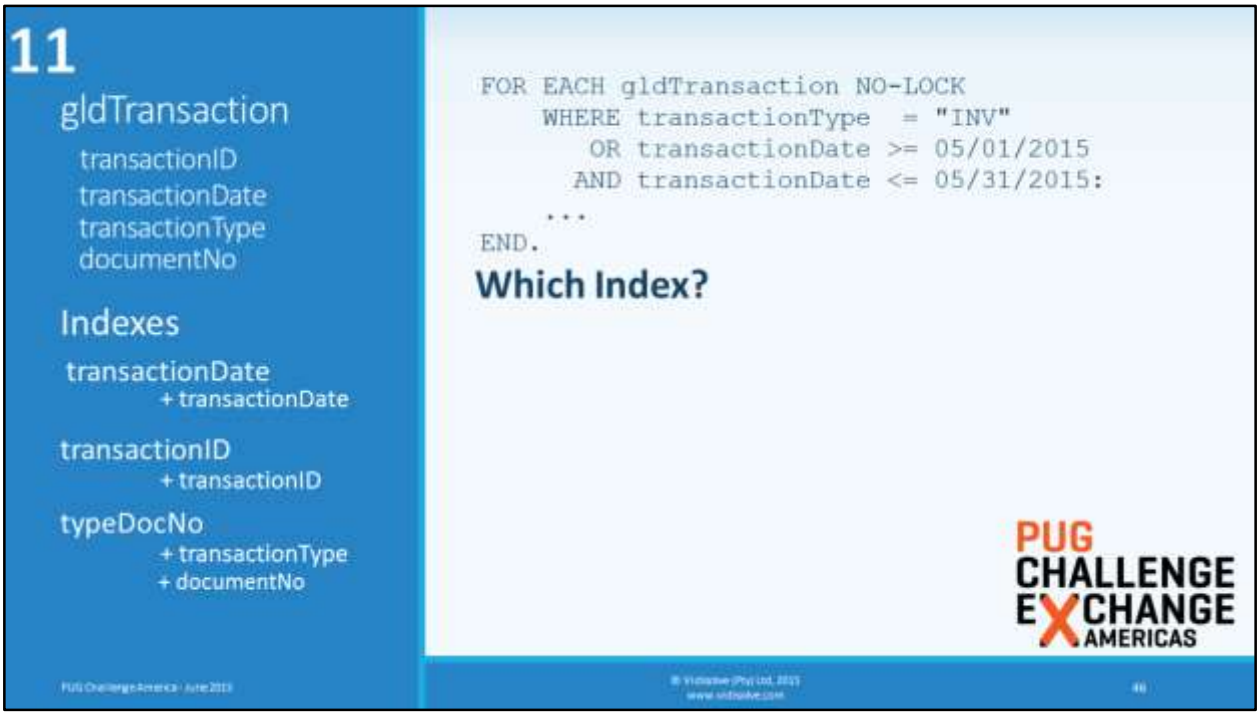
PUG CHALLENGE EXCHANGE AMERICAS



The compound index is not combined with the single index, because the whole index must be used in equality matches.



If the full index participated in the equality match, it would indeed combine with the other index, as happened in example 2.



Note that this (rather unlikely) query contains an OR.

11

gldTransaction

transactionID
transactionDate
transactionType
documentNo

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

typeDocNo
+ transactionType
+ documentNo

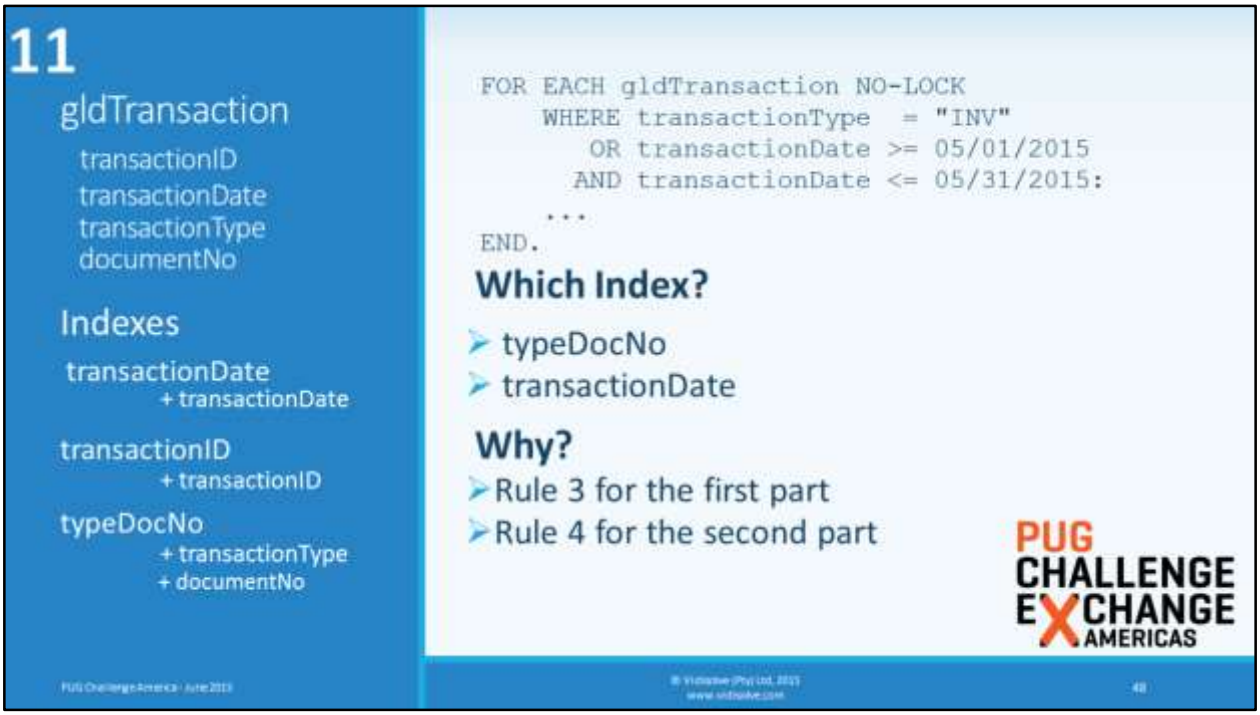
```
FOR EACH gldTransaction NO-LOCK
    WHERE transactionType  = "INV"
       OR transactionDate >= 05/01/2015
      AND transactionDate <= 05/31/2015:
    ...
END.
```

Which Index?

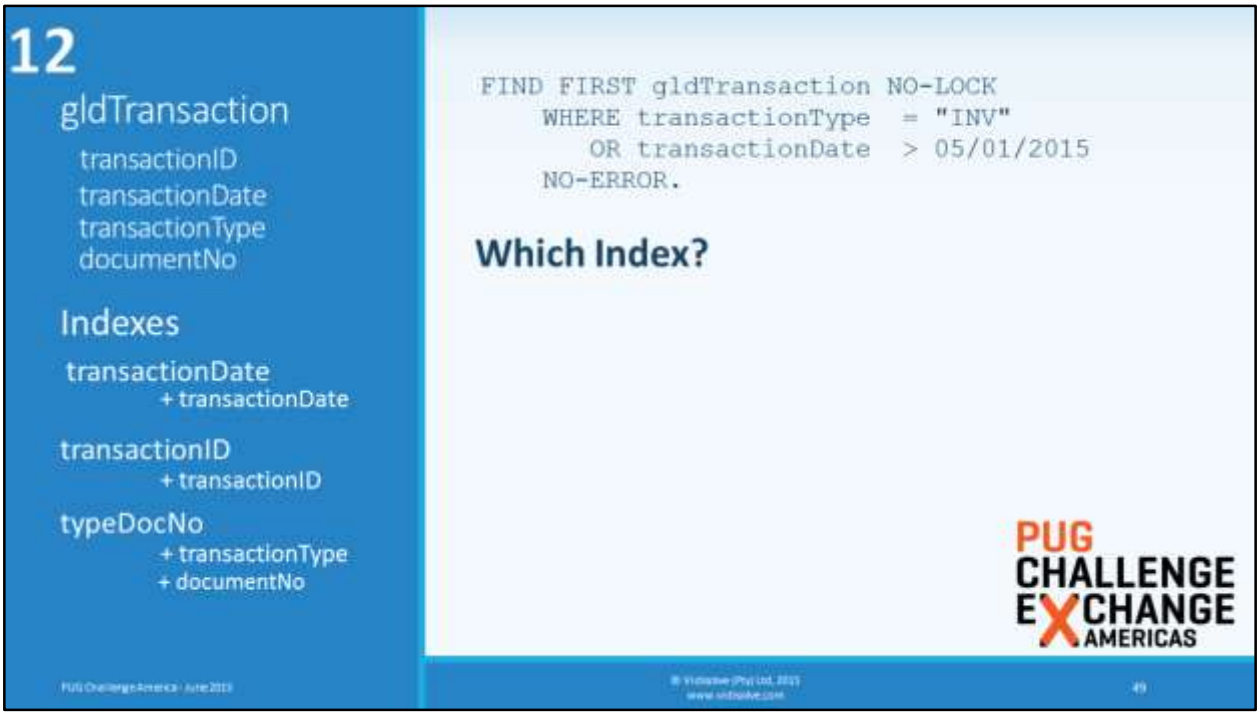
- typeDocNo
- transactionDate

Why?

PUG CHALLENGE EXCHANGE AMERICAS



Each side of the OR is evaluated as a separate query and gets its own index bracket. Note that if any one of these resulted in a full index scan of any index, only that index would be used, as all the records will be accessed in any case.



What will happen when we use a FIND?

12

gldTransaction

transactionID
transactionDate
transactionType
documentNo

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

typeDocNo
+ transactionType
+ documentNo

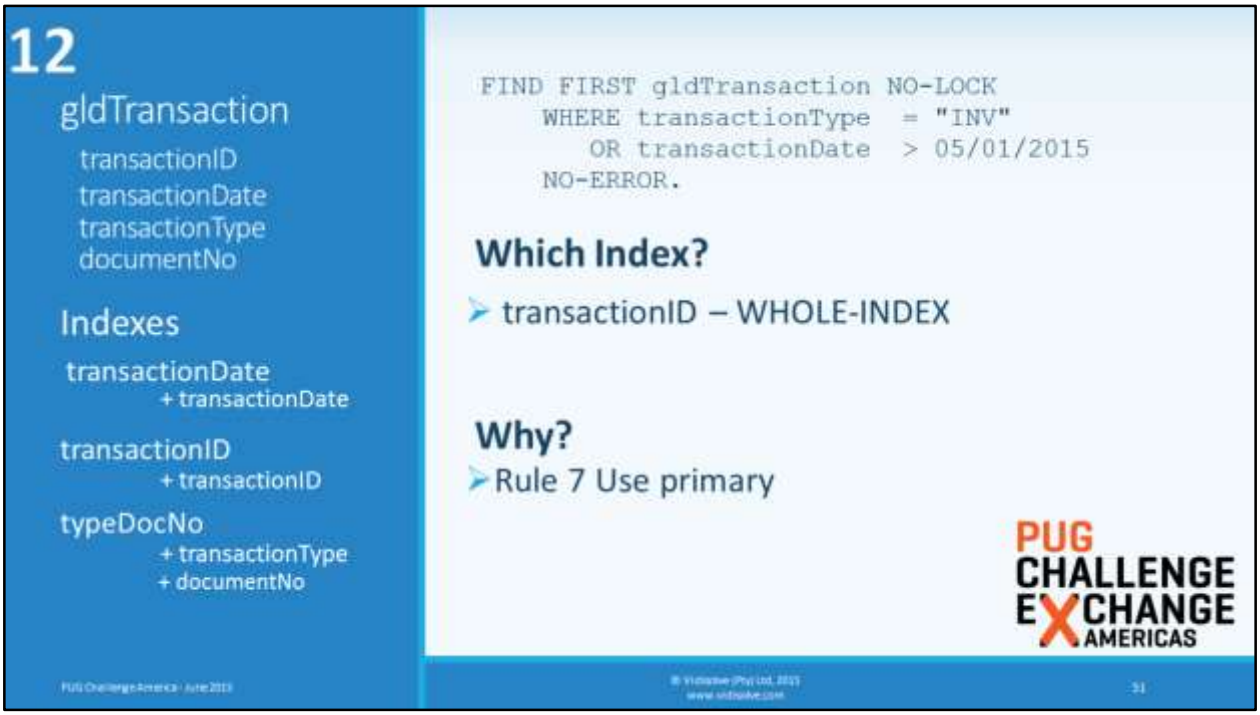
```
FIND FIRST gldTransaction NO-LOCK
    WHERE transactionType  = "INV"
       OR transactionDate  > 05/01/2015
    NO-ERROR.
```

Which Index?

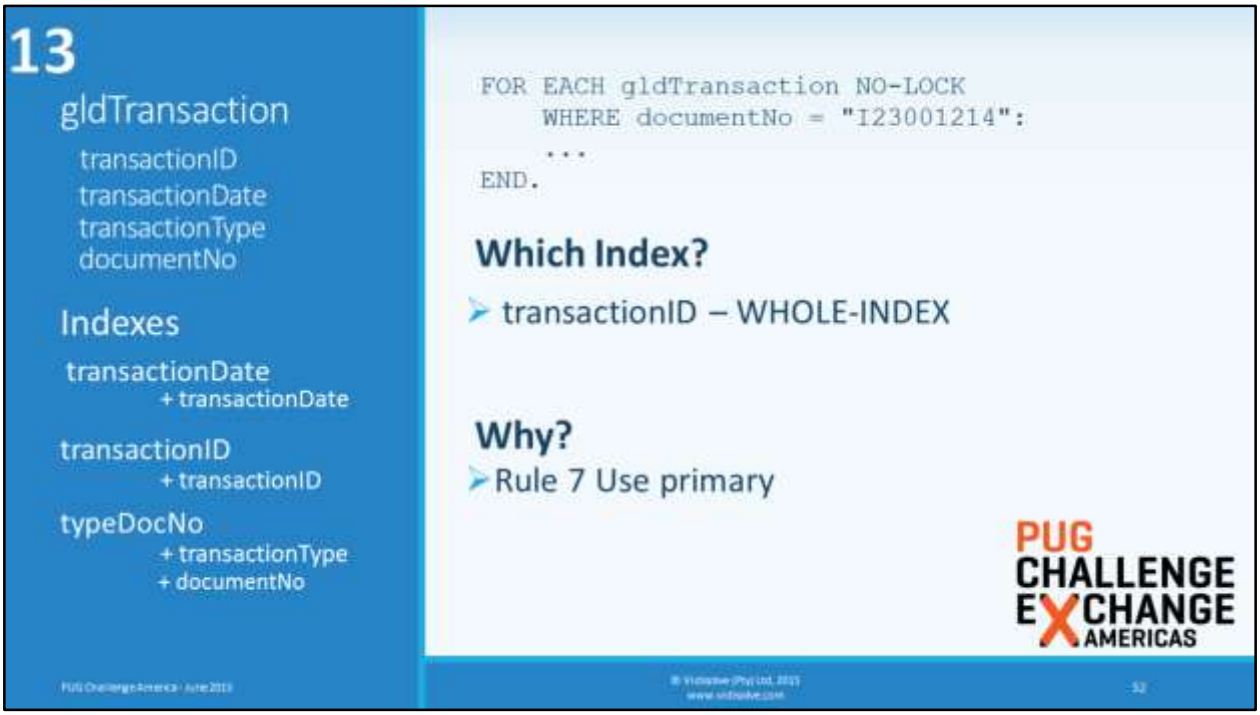
- transactionID – WHOLE-INDEX

Why?

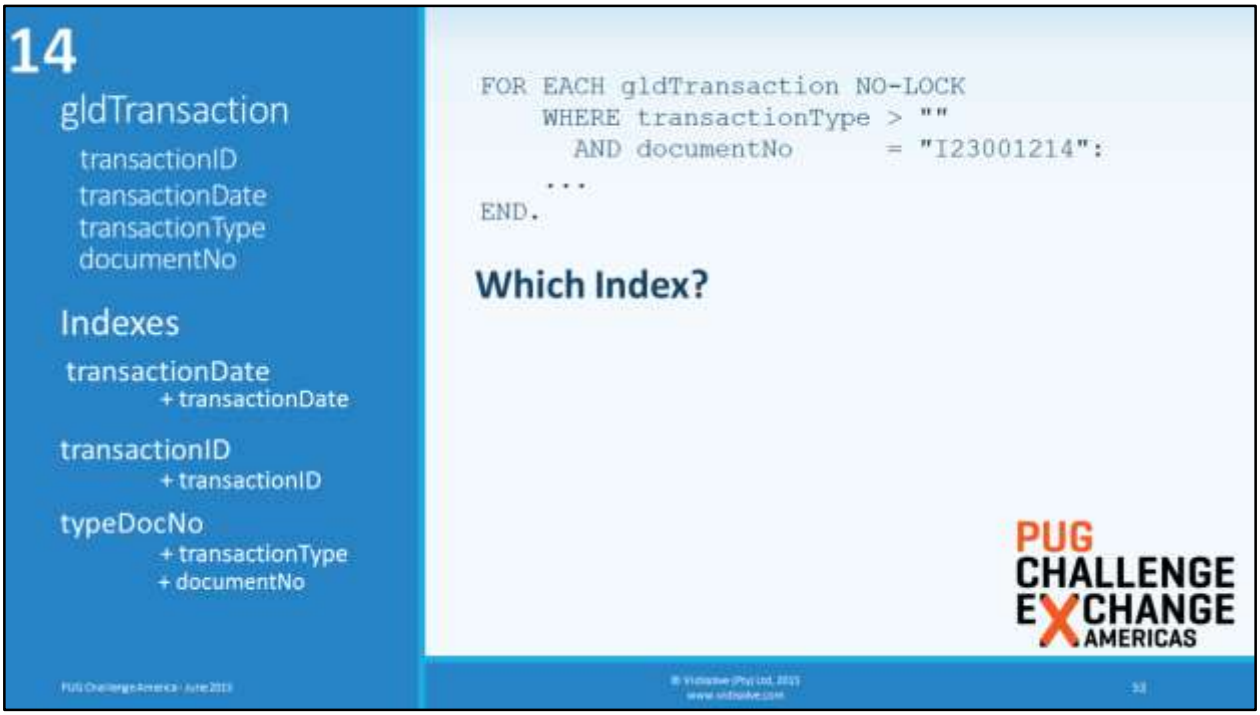
PUG CHALLENGE EXCHANGE AMERICAS



FIND allows for the use of a single index bracket only. Of the two candidate indexes that was used in example 11, neither one will satisfy the other side of the OR, hence both gets eliminated early on and the compiler falls back to rule 7.



Once again, all indexes are eliminated early on, and the compiler falls back to its default of scanning the primary index.



I often see this previous problem solved using code as above. Will it work?

14

gldTransaction

transactionID
transactionDate
transactionType
documentNo

Indexes

transactionDate
+ transactionDate

transactionID
+ transactionID

typeDocNo
+ transactionType
+ documentNo

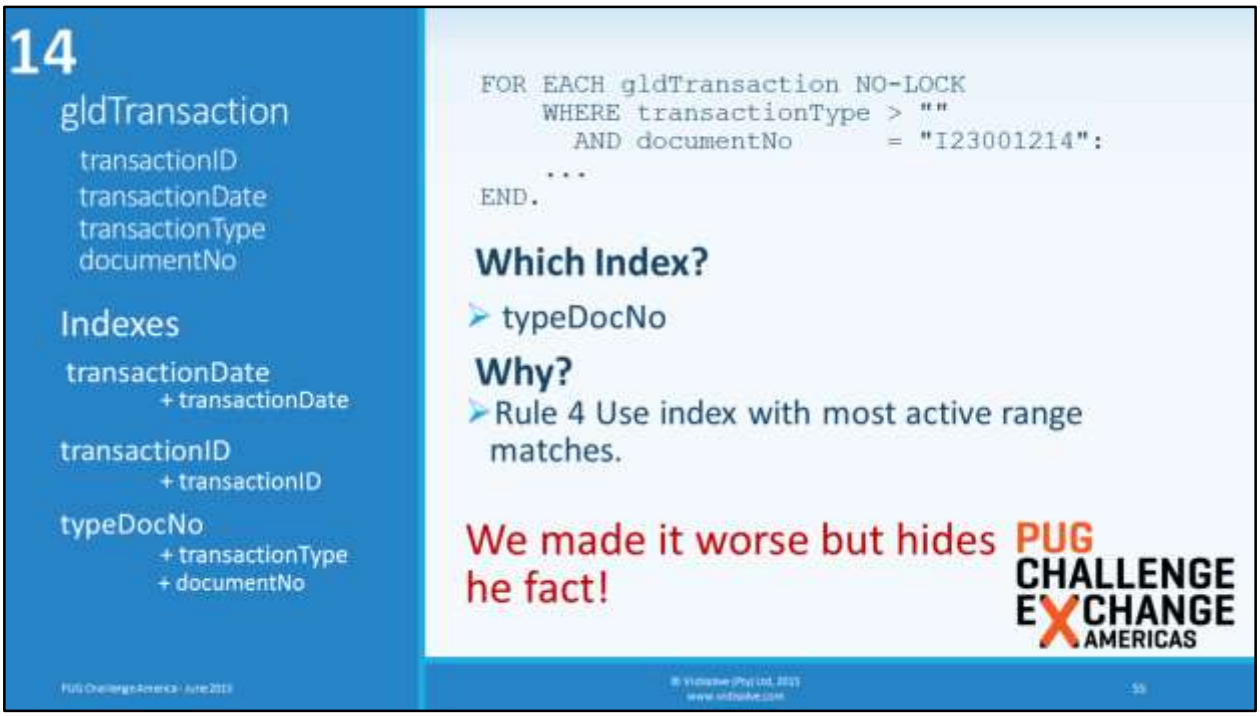
```
FOR EACH gldTransaction NO-LOCK
    WHERE transactionType > ""
      AND documentNo      = "I23001214":
    ...
END.
```

Which Index?

- typeDocNo

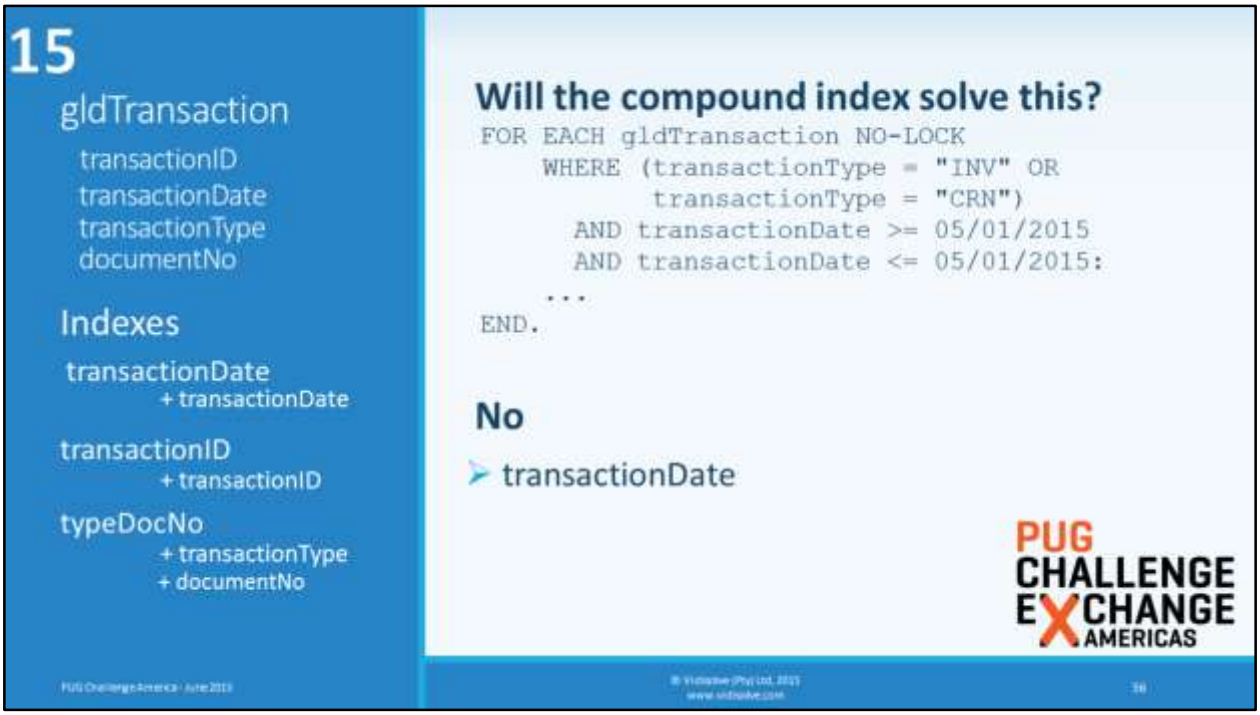
Why?

PUG CHALLENGE EXCHANGE AMERICAS



The index with the most range matches was selected. This means that components after the range matching component (transactionType) are ignored.

Since anything is > “”, we have a range here that spans the whole index, but the compiler does not realize this, at is has a “value” for the lower boundary of the bracket, so the WHOLE-INDEX is missing from the cross reference. But this index has two fields. Thus the index tree consists of all the rowids in the table plus all the transaction types plus all the document numbers, which could in total actually occupy more disk space than the primary index, which consists of all the rowids and all the transactionIDs, in which case we are now reading more blocks from disk and still access the entire table. Note however that in this design we expect the transactionID to be large, random values that will compress poorly, while the document type and number should lead to fairly good index compression, so this might still be the better option.



Once again, we have a range match bound with AND to an OR function, hence the index with the most range matches is selected.

Questions?

Feedback and more examples welcome

Simon Prinsloo

simon@vidisolve.com