Bug 1414: The Problem

- XQuery has two kinds of comparison operators
  - General comparisons: `=, !=, <, <=, >, >=`
  - Value comparisons: `eq, ne, lt, le, gt, ge`
- Assume a schemaless document (untyped data)
- `age > 21` returns true or false
- `age gt 21` is a type error (!)

How did this happen?

- The general comparison operators were inherited from XPath, and are existentially quantified
  - `author = "Gray"` is true if any author is Gray
- Therefore general comparison operators are not transitive
  - (1, 2) = (2, 3) and (2, 3) = (3, 4) but not (1, 2) = (3, 4)
- Desire to have transitive comparison operators to serve as the basis for grouping, ordering, etc.

Casting behavior

- General comparison operators cast untyped data to the type of the other operand
  - `age > 21` casts age to an integer
  - `city = "San Jose"` casts city to a string
  - Reasons: usability, XPath 1.0 compatibility
- Value comparison operators *cannot* have this casting behavior if they are to be transitive! Example:
  - `untyped("7") lt 9` would be true
  - `9 lt untyped("11")` would be true
  - `untyped("7") lt untyped("11")` would be false
- Therefore the value comparison operators always cast untyped data to string.

Are value-comparisons really transitive?

- Well, no (because of loss of precision on promotion)
- Suppose decimal has greater precision than double
  - `VBDec eq VBDbl` and `VBDbl eq VBDec + 1`
    but not `VBDec eq VBDec + 1`
- Value comparisons are transitive except for precision
- Call this property "mostly transitive"
- Question: Are "mostly transitive" operators useful in query rewrites, etc.?
  - SQL has had such operators for a long time
  - In implementations that use a common implementation for decimal and double, the operators are really transitive

The Microsoft Proposal

- See w3c-xsl-query/2005Jun/0027.html
- Change value comparisons to have the same casting behavior as general comparisons
- `age gt 21` would behave like `age > 21` if `age` returns an untyped value.
- But value and general comparisons would still treat empty sequences differently:
  - `age gt 21` returns ( ) if `age` is ( )
  - `age > 21` returns false if `age` is ( )
- Value comparisons would no longer be "mostly transitive"

The Consistency Argument

- It's good for `=` to be more consistent with `eq`, etc. (though some inconsistencies would remain)
- `order by` and `distinct-values` are defined to cast untyped data to strings before comparing.
  - Currently consistent with `gt` but not with `>`
  - Under MS proposal, would be consistent with neither
  - Example:

```
for $p in /item/price
where $p gt 25
order by $p
return $p
```

  - Can implementations use an index on `/item/price` ?
  - Will users be surprised to see `101, 29, 370, 55` ?

Consistency Argument (cont'd.)

- `max` and `min` are defined to cast untyped data to double before comparing
  - Currently inconsistent with both `gt` and `>`.
  - MS proposal would not make this any worse.
- Summary:
  - The MS proposal pushes the inconsistency from one place (`gt` vs. `>`) to another (`gt` vs. `order by`)
  - Whether this is an improvement is a matter of taste

The Optimization Argument

- Mostly-transitive operators are useful in optimization
- Query rewrites:
  - Suppose A eq B and B eq 5. Can you infer that A eq 5?
  - Should not rely on static analysis
- Access methods:
  - Sort-merge joins, hash-joins, etc.
  - Where does untyped data go in the sort or hash, if it can behave like either string or numeric?
- Indexing:
  - Where does untyped data go in an index, if it can behave like either string or numeric?
- Arguably, the MS proposal would make optimization more difficult.

The Transitivity Argument

- The value comparisons were created to be mostly-transitive.
- Mostly-transitive operators may have uses that we have not yet anticipated.
- If the value-comparison operators are not mostly-transitive, the case for these operators is weakened.
- If you want an operator that behaves like `=` but requires singleton operands, why not use `fn:exactly-one()` on one or both operands?

The Bottom Line

- None of the arguments for or against this proposal are absolutely compelling.
- IBM votes to preserve the status quo.