Ranges, Ranges Everywhere!

Stew Ashton
Independent
Paris, France

Keywords:

Oracle Database 12c, SQL, ranges, date ranges, Temporal Validity

Introduction

The current data explosion requires querying, summarizing and managing more and more data related to time periods or other ranges. Range-based data present specific problems of coherence, query performance and scalability. This session will present typical problems and new, highly efficient SQL solutions - many using Row Pattern Matching, a feature of Oracle Database 12c.

The session will start by an analysis of ranges, which will focus on the fundamental difference between data that allows overlapping ranges and data that does not.

It will continue with queries and DML on one table: analysis of gaps and overlaps, and packing of data with contiguous ranges.

Finally, problems with multiple tables will be addressed: joining range-based tables such as temporal tables and range-based DML.

Techniques will be presented to greatly reduce CPU consumption by range-based queries, in particular by avoiding JOINs with inequality conditions.

This session is geared primarily to SQL practitioners who are willing to unleash the full power of SQL in the Oracle database. They will learn how to identify specific requirements and pitfalls of range-based data, and how to apply new techniques for decreasing CPU usage and improving performance by orders of magnitude.

What is a Range?

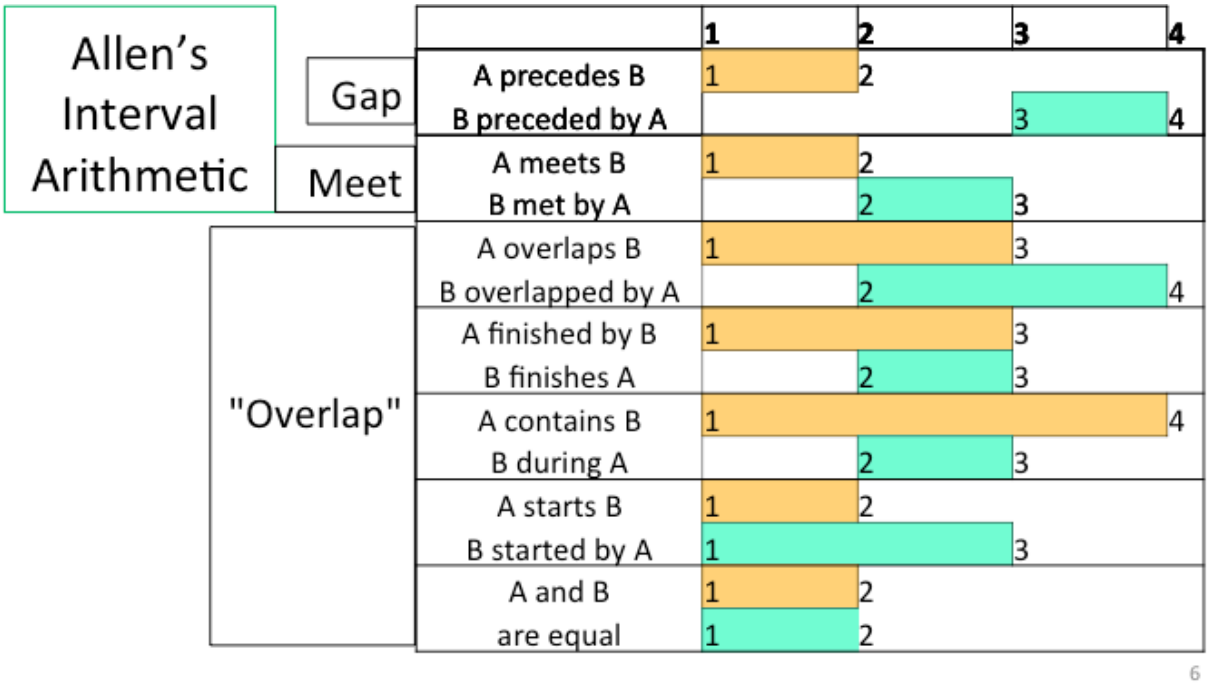
A range is composed of two values that can be compared, and that should always use the same datatype. Comparable datatypes are:

- Discrete: integer, date (without time)
- Continous: number, datetime, interval, (n)(var)char, raw
- rowid

When designing a range, two questions must always be addressed:

- Is the "end" value part of the range?
- Are NULLs allowed?

To help answer those questions, I'll introduce Allen's list of the possible relations between two ranges:



Ranges must be capable of "meeting". That means there can be no value that comes after the first range and before the second. For continuous quantities, the only way to assure that is to make the end value of the first range equal to the start value of the second range, and to say that the end value is **exclusive**. Ranges or intervals defined this way are called [Closed-Open) .

The [Closed-Open) technique is used by Temporal Validity (as defined in the SQL:2013 standard and implemented in Oracle Database 12c) and by the WIDTH_BUCKET() function that places values within equiwidth histograms.

Discrete quantities can use inclusive or exclusive end values, so everything works with exclusive. The only exception is ROWID ranges: the access method "BY ROWID RANGE" requires an inclusive end point.

Range-related DDL

Ranges are hard enough to manage with good data; if the data doesn't follow the rules, no SQL can function properly. Range values must have appropriate CHECK constraints:

- Start value < end value
- If integer, define as such
- If date without time component, CHECK(dte = trunc(dte))
- Define as NOT NULL or decide exactly what NULL means.

In Temporal Validity for example, a NULL end date means "until the end of time".

It is also critical to decide whether **overlaps** should be allowed. Overlaps are very hard to avoid if you don't want them, and hard to query efficiently if you do.

Avoiding overlaps

Referring to Allen's definitions, some overlaps cannot be avoided by classic UNIQUE constraints on one or both range columns: (partial) overlaps and "contains". These are "inter-row" constraints that can only be managed by:

1. Triggers (hard to do right)
2. ON COMMIT materialized views (not very scalable)
3. SQL assertions (not implemented yet)

I present an alternative I call the "virtual range". In fact, the row contains only the start date, and the end date is calculated to be the start date of the next row in chronological order. This solution avoids overlaps and gaps; I present two alternatives for managing gaps. In one solution, a gap is identified by an intervening row with a flag indicating that it starts a gap. The illustration below shows the physical table and the "virtual range" view that can be derived from it.

START_VALUE	D
16-11-15 08:30	
16-11-15 09:30	D
16-11-15 18:30	

START_VALUE	END_VALUE
16-11-15 08:30	16-11-15 09:30
16-11-15 18:30	(null)

Gaps

Finding gaps in ranges is certainly one of the most frequent actions in data processing, if only because of free time searches in calendars. Using analytics, the solution is very efficient and concise:

WHAT	FROM_TM	TO_TM
Ranges	09:00	10:50
Grids	10:00	10:45
SQL Magic	10:00	10:45
Plugins	12:00	12:45

FROM_GAP	TO_GAP
	09:00
10:50	12:00
12:45	
10:50	12:00
12:45	

```
select * from (
  select max(to_tm) over (
    order by from_tm nulls first
  ) as from_gap,
  lead(from_tm) over (
    order by from_tm nulls first
  ) as to_gap
  from (
    select * from t
    union all
    select null, null, null from dual
  )
) where lnnvl(from_gap >= to_gap);
```

Overlaps

Querying overlaps is a difficult problem, because the number of rows that might overlap each other is unknown. Also, self joins may be very inefficient and generate numerous rows.

I shall present two techniques. The first one uses UNPIVOT and analytics to split the ranges into components, and return only the components involved in an overlap; as a bonus, I return the number of rows that cover that component.

The second technique returns the original rows that overlap, grouped together by the overall range they cover. For example, if 1-3 overlaps 2-4 then I return 1-4 and 1-3, followed by 1-4 and 2-4. The overall range 1-4 identifies all the rows that overlap with those boundaries.

Packing

If two rows overlap or meet, and otherwise have the same data, it is not uncommon to condense them into one row. Chris Date calls this operation "packing". MATCH_RECOGNIZE makes it easy to identify and condense these rows, after which a MERGE statement can apply the changes.

JOIN: range and value

Using an example provided by Jonathan Lewis, I will demonstrate how MATCH_RECOGNIZE can replace an inequality join and provide the same answer up to 500 times faster.

JOIN: two ranges

The techniques used for identifying overlaps can be adjusted to return intersections between rows in two different tables.

Updating ranges in DW tables

Using an example from Dr. Friedrich Holger, I will demonstrate how a history table with ranges can be updated with new data, using MATCH_RECOGNIZE.

Contact address:

Stew Ashton
Independent
(no work address)
75017 Paris, France

Phone: (no work phone)
Email stew.ashton@gmail.com
Internet: stewashton.wordpress.com