SQL Server 2014 In-Memory Tables (Hekaton)

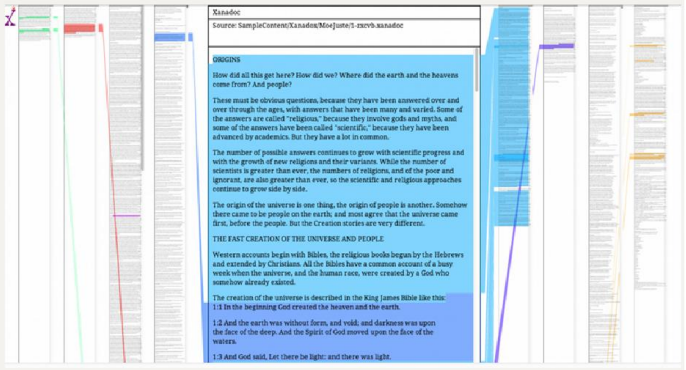
Andrew Novick
www.NovickSoftware.com

Andy Novick

- SQL Server Consultant
- SQL Server MVP

Did you ever work on a software project that was late?

Xanadu, The World's Most Delayed Software, Is Finally Released After 54 Years In The Making



What's a Heketon?

- 100 Times Faster!
- They didn't get there ;(

What makes it faster?

- No locks, latches, spinlocks, etc.
- Data is written sequentially, only!
- No indexes are stored or logged. They're in memory only.
- Non-durable tables. No I/O
  - Cache
  - Replacement for TempDB in some cases.
- Checkpoints are background sequential I/O
- Compiled Native stored procedures

What's not faster

- Chatty applications may not see any gain.
- Logging is still done (except for indexes)
  - So Low Log Latency is still important
- DML on many rows????

Compiled Sproc - Requirements

- Native Compilation
- SCHEMABINDING
- EXECUTE AS OWNER
- BEGIN Atomic – Begin a Tran or Savepoint
  - Isolation Level Must be declared
  - LANGUAGE – Fixed at compile time

Compiled SPROC Surface Area Limitations

- Memory optimized tables only
- INNER JOIN supported
- OUTER JOIN not supported

- Also no: MERGE, OUTPUT,

Estimating Memory Consumption

- Memory Size = Table Size + SUM(Index Size)

- TABLE Size = Row Size * Row Count
- Row Size = SUM (Column Size) + Header
- Header = 24 + 8 * Index Count

- Hash Index size = Bucket_Count * 8 bytes

- The multiply it all by 2

Can it Crash the system

- YES
- Integrated with Resource Governor

Durability

- Data file
- Delta file

Merge Process

- DBA job
- Processes the changes and produces a new starting data file
- sys.sp_xtp_merge_checkpoint_files
  @database_name
  , @transaction_lower_bound
  , @transaction_upper_bound

Indexes

- Inherently covering
  - NO INCLUDE needed
- HASH
- RANGE

Logging

- Uses the traditional Log
- Only logical changes is logged.
- All 3 recovery models are supported

Recovery

- Recovery at the speed of I/O
- This is a function of database size!

Isolation Levels

- Snapshot
  - Reads are consistent as of the start of the TXN
  - Writes are always consistent
- Repeatable Read
  - Reads give the same row version a commit time as earlier
- Serializable
  - As if there were no concurrent transactions
  - All actions happen at a single commit time

Optimistic Locking

- There are no traditional:
  - Locks
  - Latches
  - Spin-locks
- Multi-version Optimistic Locking
- You MUST code for failure.

Interop vs Compiled SProc

Interop

- Access both memory and disk tables
- Most of T-SQL

- Best for
  - Ad hoc queries
  - Reporting Queries
  - Migration

Memory with Compiled Sproc

- Only Memory tables
- Limited T-SQL

- Best for
  - OLTP
  - Optimized performance

Not Supported - Tables

- LOB – Varchar(MAX), NVARCHAR(MAX)
  - Text, ntext, image, varbinary(max)
- Foreign Key
- Trigger
- Replication
- Collations other than Bin2! ('A' != 'a')

Not Supported – Interpreted TSQL

- TRUNCATE TABLE
- MERGE (memory-optimized table as target)
- Dynamic and keyset cursors (these automatically degrade to static).
- Access from CLR modules, using the context connection.
- Referencing a memory-optimized table from an indexed view.
- Cross-database queries
- Cross-database transactions
- Linked servers

No Supported in Compiled Sprocs

- Merge
- OUTER JOINS
- Lots of other stuff
  - OUTPUT
  - CURSOR

DEMO

OLTP Demo

	OnDisk	InMem
1,000,000	133	14

EAV

- Entity-Attribute-Value
- Entity – a thing that has characteristics
  - House
  - Person
  - Stock
- Attribute – Characteristic of an entity
  - Number of bedrooms
  - Weight
  - Market price
- Value the value of the attribute for the entity

Point-in-Time Data

- Each row represents the value of the attribute for the entity for a range of time
- Ways to represent the time range
  - begin_datetime
  - begin_datetime - end_datetime

Typical EAV table – begin_datetime

```
CREATE TABLE eav_pit_only_begin_datetime (
          attribute_id int not null
        , entity_id int not null
        , begin_datetime datetime not null
        , value varchar(255) not null

        , constraint pk_eav_pit_only_begin_datetime
                primary key clustered
                (attribute_id, entity_id, begin_datetime)
 )
```

EAV Multi-table with View

- Separate into 3 tables
  - eav_current
    - begin_datetime
    - No end_datetime it is INFERRED as 9999-12-31
    - Partitioned by attribute_id
  - eav_recent
    - begin_datetime/end_datetime
    - Partitioned_by_attribute_id
  - eav_history
    - Begin_datetime/end_datetime
    - Partitioned by end_datetime
- ETL loads to eav_current saves history to eav_recent
- Archive moves rows from eav_recent to eav_history

Typical EAV table – begin/end datetime

```
CREATE TABLE eav_pit_only_begin_datetime (
          attribute_id int not null
        , entity_id int not null
        , begin_datetime datetime not null
        , end_datetime datetime default ('9999-12-31')
        , value varchar(255) not null

        , constraint pk_eav_pit_only_begin_datetime
                primary key clustered
                (attribute_id, entity_id, end_datetime)
)
```

Typical EAV table – begin/end + Types

```
CREATE TABLE eav_pit_begin_end_datetime_types (
          attribute_id int not null
        , entity_id int not null
        , begin_datetime datetime not null
        , end_datetime datetime default ('9999-12-31')
        , data_type char(1) not null -- C, D, F, I
        , value_c varchar(255) null
        , value_d (255) null
        , value_f (255) null
        , value_i (255) null
    , constrant pk_eav_pit_begin_end_datetime_types
                primary key clustered
                (attribute_id, entity_id, end_datetime)
)
```

References

- SQL Server 2014 In-Mem Sample
  http://msdn.microsoft.com
  /en-us/library/dn511655(v=sql.120).aspx?WT.mc_id=Blog_SQL_SQL2014_DI
- Why Hekaton is Truly Revolutionary
  http://sqlblog.com/blogs/merrill_aldrich/archive/2013/10/21/why-hekaton-in-memory-oltp-truly-is-revolutionary.aspx

Thank you for coming

Andy Novick

anovick@novicksoftware.com