NIWA DAS database upgrade

Brent Wood
NIWA

NIWA
Taihoro Nukurangi

National Institute of Water and Atmspheric Research

New Zealand Crown Research Institute (CRI)

Government owned commercially funded environmental research institute, with science centres focused on:

Aquaculture
Atmosphere
Climate
Coasts and Oceans
Environmental Information
Fisheries
Freshwater and Estuaries
Natural Hazards
Pacific Rim

http://niwa.co.nz

NIWA
Taihoro Nukurangi

Who am I?

Brent Wood

- Joined NIWA (or its predecessor) in 1975
- Fisheries field/seagoing technician
- Data manager/database manager/database designer
- Metadata manager (Geonetwork)
- Open source GIS user (QGIS, PostGIS, Spatialite, GMT, Mapserver, Geoserver, R, ...)

NIWA
Taihoro Nukurangi

This presentation describes a rebuild of the NIWA research vessel Data Acquisition System database

It now stores around 40 billion instrument/sensor readings captured since 1990

The 2021 rebuild addressed performance issues with the earlier design, and undertook to meet several user requirements (in addition to normal data management needs):

- rapid access to fine scale (1 min to 1 sec) temporal data (eg: diurnal analytics)
- rapid access to courser scale (1 hr to 1 day) data (eg: seasonal/annual analytics)
- near real time access to manage operational deployments (eg: underwater camera)

It describes: why we did it (as above)
what we did
how we did it
with some example SQL queries

(Gets a bit technical WRT to SQL queries!)

NIWA
Taihoro Nukurangi

DAS database:

NIWA Data Acquisition System database

Stores instrument/sensor readings, mostly from RV Tangaroa

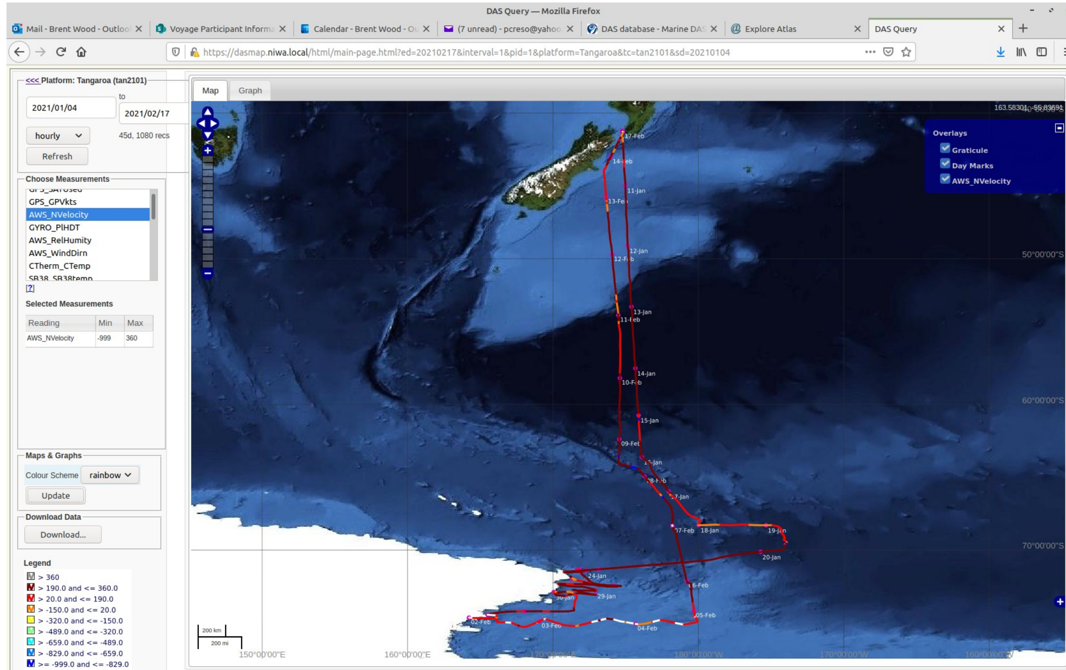
Historically at 1 min intervals

Captured by various DAS software applications since 1990:

- original OS/2 DAS developed by Datacom
- FRC in-house DAS (Python – storage optimised - Mb/Gb, not Tb)
- Techsas (IFREMER commercial application)

NIWA
Taihoro Nukurangi

User application: DASmap



Developed for NIWA in early/mid 2000’s by Integrated Mapping (John McCombs, Christchurch)

Written using Mapserver & Python Mapscript as web application

Became too slow as the db grew over a decade or so, but still meets general user needs for previewing & extracting reading data

Alter this to use new database:

- modify embedded SQL’s to query hstore table/column structure -
- new queries returning identical result as old queries on old tables

NIWA
Taihoro Nukurangi

PostgreSQL:

“World’s most advanced Open Source Database” (or so they claim!)

PostgreSQL is generally regarded as a reasonable alternative to Oracle as a robust, secure enterprise level RDBMS

PostgreSQL is an ORDBMS, not just an RDBMS (supports object based non-relational data)

PostgreSQL has more features than ANY other RDBMS, and is fully ACID compliant (Atomicity, Consistency, Isolation, Durability)

Already widely used in NIWA

NIWA
Taihoro Nukurangi

Postgres extensions – What are they:

Additions to core Postgres (often third party) to add focused and specific functionality

The extensions used in this case:

PostGIS: adds OGC SFS spatial datatypes and functions/operators
- probably the most complete and powerful spatial functionality of any RDBMS

hstore: adds key/value datatype and functions to enter, access & decompose these
- simpler than jsonb and meets requirements (core extension)

TimescaleDB: adds timeseries functionality and auto partitioning
- think of it as doing for timeseries data what PostGIS does for spatial data

NIWA
Taihoro Nukurangi

Postgis:

Postgis provides an enhanced spatial data capability for the Postgres ORDBMS

Exceeds the specification for OGC SFS specification for SQL

Meets the ANSI MM/SQL specifications for SQL databases

Open Geospatial Consortium Inc.

Date: 2010-08-04

Reference number of this document: OGC 06-104r4

Version: 1.2.1

Status: Corregendum

Category: OpenGIS® Implementation Standard

Editor: John R. Herring

OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option

Copyright © 2010 Open Geospatial Consortium, Inc.
To obtain additional rights of use, visit http://www.opengeospatial.org/legal/

Document type: OpenGIS® Implementation Standard
Document subtype: (none)
Document stage: Approved Corrigendum
Document language: English

hstore:

Core extension providing support for key/value storage in Postgres

What are key-value pairs?

A value stored in an hstore column is a list of keys, each with a value (so a pair)
1=>3.14159,2=>1.4142,3=>2.7183, ...

Why are they appropriate for this use case?

The “key” value is an identifier for a sensor on an instrument
(instrument metadata is managed in various other database tables)

The “value” is the sensor value for the timestamp (in another column) in the record

Thus a database column stores all sensor readings for a given timestamp using a structured but non-normalised No-SQL approach within an SQL database

NIWA
Taihoro Nukurangi

TimescaleDB:

Provides storage, query and analytical tools for timeseries data in a Postgres database

Fundamentally changes your Postgres database –

provides auto-partitioning of timeseries tables (hypertables) for highly optimised performance with never ending streams of accumulating timeseries data

adds tools to aggregate, count, analyse records (quickly!)

we use the free open source version in a local Postgres database, but it is also available on AWS, Azure & Google Cloud.

http://www.timescale.com

NIWA
Taihoro Nukurangi

Old database: Postgres + Postgis

- Readings normalised (robust, RDBMS “best practice” design),
  one timestamped reading per record
- 1 minute interval
- Manual one year partitioning
- 600,000,000 readings = 600,000,000 records
- DASmap gui took >3 minutes to load 1 voyage

New database: Postgres + Postgis/Timescaledb/hstore

- Readings as hstore key/value pairs (non-relational approach),
  one timestamp per record
  many readings/record, but individually accessible
- 1 second interval
- Automated 1 week chunks in Timescaledb hypertable
- ~40,000,000,000 readings = ~70,000,000 records
- DASmap gui takes <10 seconds to load 1 voyage

NIWA
Taihoro Nukurangi

Basic table structures:

Old reading table:

(*manually maintained yearly partition, 1 min readings*)

t_reading

timer	timestamp	primary key,
sensor	int	pkey + foreign key on t_sensor (key),
value	int	

New reading table:

(*Timescaledb hypertable – auto partition weekly, 1 sec readings*)

t_reading_hstore_sec

timer	timestamp	primary key
values_hstore	hstore	

NIWA
Taihoro Nukurangi

1 min LIFO extract SQL:

get 1 day of 1 minute LIFO style readings from 1 sec table:

```
SELECT DISTINCT date_trunc('minute',_timer)+interval '1 minute' as timer,
                key::int,
                first_value(value) as last_value
OVER (PARTITION BY key, date_trunc('minute',_timer) order by _timer desc)
FROM (SELECT timer as _timer,
              (each(values_sec)).*
      FROM t_reading_hstore_sec
      WHERE timer between '2013-04-15 00:00:00'
                      and '2013-04-15 23:59:59' ) as x
ORDER BY key::int, timer;
```

NIWA
Taihoro Nukurangi

```
SELECT DISTINCT date_trunc('minute',_timer)+interval '1 minute' as timer,
                key::int,
                first_value(value) as last_value
OVER (PARTITION BY key, date_trunc('minute',_timer) ORDER BY _timer desc)
FROM (SELECT timer as _timer,
              (each(values_sec)).*
      FROM t_reading_hstore_sec
      WHERE timer between '2013-04-15 00:00:00'
                      and '2013-04-15 23:59:59' ) as x
ORDER BY key::int, timer;
```

Any guesses how long it takes (from 40b readings)?

To get the values for 1 minute only (just change “between” value): ???

To get the values for 1 day (1440 minutes): ???

NIWA
Taihoro Nukurangi

```
SELECT DISTINCT date_trunc('minute',_timer)+interval '1 minute' as timer,
                key::int,
                first_value(value) as last_value
OVER (PARTITION BY key, date_trunc('minute',_timer) ORDER BY _timer desc)
FROM (SELECT timer as _timer,
              (each(values_sec)).*
      FROM t_reading_hstore_sec
      WHERE timer between '2013-04-15 00:00:00'
                      and '2013-04-15 23:59:59' ) as x
ORDER BY key::int, timer;
```

Any guesses how long it takes (from 40b readings)?

To get the values for 1 minute only (just change “between” value): <30 ms

To get the values for 1 day (1440 minutes): ~55 sec (~7hrs for a year)

NIWA
Taihoro Nukurangi

```
SELECT DISTINCT date_trunc('minute',_timer)+interval '1 minute' as timer,
                key::int,
                first_value(value) as last_value
OVER (PARTITION BY key, date_trunc('minute',_timer) ORDER BY _timer desc)
FROM (SELECT timer as _timer,
              (each(values_sec)).*
      FROM t_reading_hstore_sec
      WHERE timer between '2013-04-15 00:00:00'
                      and '2013-04-15 23:59:59' ) as x
ORDER BY key::int, timer;

        timer        | key |           last_value
---------------------+-----+--------------------------------
 2013-04-15 00:01:00 |  27 | 17.799999237060547
 2013-04-15 00:02:00 |  27 | 17.700000762939453
 …
 2013-04-15 01:20:00 |  63 | 1385.6600341796875
 2013-04-15 01:21:00 |  63 | 1394.30004882815
 …
```

NIWA
Taihoro Nukurangi

SQL to get 1 minute lats/longs for a trip (7 weeks) from hstore key/value pairs:

Use COALESCE to get first non-NULL value as there are several GPS units with different keys over the years...

```
SELECT timer,
       COALESCE (value -> '67', value -> '307', value -> '317', '') as lat,
       COALESCE (value -> '68', value -> '308', value -> '318', '') as lon
FROM t_reading_hstore_min
WHERE timer >= '20210101'::date
     AND timer <= '20210217'::date
ORDER BY timer;
```

Execution time: ???

NIWA
Taihoro Nukurangi

SQL to get one minute lats/longs for a trip (6 weeks) from hstore key/value pairs:

```
SELECT timer,
       COALESCE (value -> '67', value -> '307', value -> '317', '') as lat,
       COALESCE (value -> '68', value -> '308', value -> '318', '') as lon
FROM t_reading_hstore_min
WHERE timer >= '20210101'::date
     AND timer <= '20210217'::date
ORDER BY timer;
```

Execution time: ~1 sec

NIWA
Taihoro Nukurangi

SQL to get one minute lats/longs for a trip of 7 weeks) from hstore key/value pairs:

```
SELECT timer,
      COALESCE (value -> '67', value -> '307', value -> '317', '') as lat,
      COALESCE (value -> '68', value -> '308', value -> '318', '') as lon
FROM t_reading_hstore_min
WHERE timer >= '20210101'::date
     AND timer <= '20210217'::date
ORDER BY timer;
```

```
        timer          |    lat     |     lon
-----------------------+------------+-------------
 2021-01-01 00:00:00 | -41.311622 | 174.811812
 2021-01-01 00:01:00 | -41.311621 | 174.811812
 2021-01-01 00:02:00 | -41.311621 | 174.811812
...
```

NIWA
Taihoro Nukurangi

SQL to create a voyage trackline from lat & lon coordinates (yep - GIS at last!):

```
-- build linestring from raw hstore coords
-- use ST_RemoveRepeatedPoints() to clean repeated points
-- use ST_Simplify() to drop resolution to about 10m (remove redundant - nearby - vertices)
-- ST_Simplify @ 0.00001=34000 vertices, @ 0.0001=10000 vertices
SELECT ST_AsText(
         ST_Simplify(
           ST_RemoveRepeatedPoints(
             ST_Makeline(
               ST_SetSRID(
                 ST_MakePoint(
                   COALESCE (value -> '68', value -> '308', value -> '318')::real,
                   COALESCE (value -> '67', value -> '307', value -> '317')::real),
               4326)
             ORDER BY timer)
           ),0.0001)) as track
FROM t_reading_hstore_min
WHERE timer >= '20210101'::date
      AND timer <= '20210217'::date;
```

Execution time: ???

NIWA
Taihoro Nukurangi

SQL to create a voyage trackline from lat & lon coordinates:

```
-- build linestring from raw hstore coords
-- use ST_RemoveRepeatedPoints() to clean repeated points
-- use ST_Simplify() to drop resolution to about 10m (remove redundant - nearby - vertices)
-- ST_Simplify @ 0.00001=34000 vertices, @ 0.0001=10000 vertices
SELECT ST_AsText(
         ST_Simplify(
           ST_RemoveRepeatedPoints(
             ST_Makeline(
               ST_SetSRID(
                 ST_MakePoint(
                   COALESCE (value -> '68', value -> '308', value -> '318')::real,
                   COALESCE (value -> '67', value -> '307', value -> '317')::real),
               4326)
             ORDER BY timer)
           ),0.0001)) as track
FROM t_reading_hstore_min
WHERE timer >= '20210101'::date
      AND timer <= '20210217'::date;
```

Execution time: <1.3 sec

NIWA
Taihoro Nukurangi

SQL to return instruments/sensors for a survey:

```
SELECT i.instrument_code||'_'||m.measurement_code as reading,
              m.measurement_key
FROM t_measurement m,
            t_instrument I
WHERE m.measurement_key::varchar in (
                     SELECT distinct key
                     FROM (
                            SELECT skeys(value) as key
                            FROM t_reading_hstore_min
                            WHERE timer = '20210101'::date
                            AND timer <= '20210217'::date
                     ) as mytable)
            and m.instrument_key = i.instrument_key
 ORDER BY reading;
```

Execution time: ???

NIWA
Taihoro Nukurangi

SQL to return all sensors with data for a survey:

```
SELECT i.instrument_code||'_'||m.measurement_code as reading,
              m.measurement_key
FROM t_measurement m,
            t_instrument I
WHERE m.measurement_key::varchar in (
                     SELECT distinct key
                     FROM (
                            SELECT skeys(value) as key
                            FROM t_reading_hstore_min
                            WHERE timer = '20210101'::date
                            AND timer <= '20210217'::date
                     ) as mytable)
            and m.instrument_key = i.instrument_key
 ORDER BY reading;
```

Execution time: <7ms

NIWA
Taihoro Nukurangi

SQL to return all sensors with data for a survey:

```
SELECT i.instrument_code||'_'||m.measurement_code as reading,
                  m.measurement_key
FROM t_measurement m,
           t_instrument I
WHERE m.measurement_key::varchar in (
                           SELECT distinct key
                           FROM (
                           SELECT skeys(value) as key
                           FROM t_reading_hstore_min
                           WHERE timer = '20210101'::date
                                 AND timer <= '20210217'::date
                           ) as mytable)
    AND m.instrument_key = i.instrument_key
 ORDER BY reading;
```

```
          reading            | measurement_key
-----------------------------+-----------------
 AWS_AirTemp                 |              27
 AWS_BarPressur              |              37
 AWS_DewPoint                |              56
 Eppley_PIR1_mV              |             267
 Eppley_PIR1_Rdome           |             268
 GYRO_PlHDT                  |              95
 Rainfall_RG_mm              |             473
 ...
```

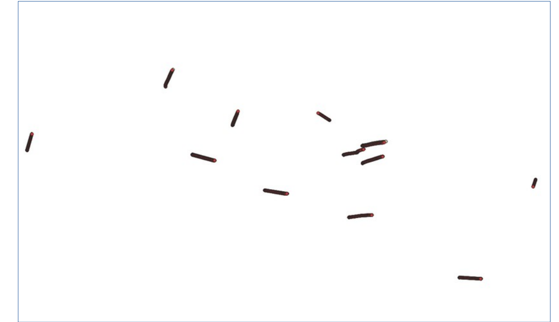
NIWA
Taihoro Nukurangi

SQL to return 1000 vessel and DTIS camera point locations from 1 min table

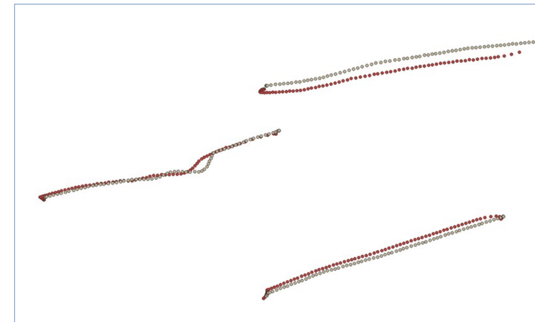
Use in QGIS to display camera & vessel positions in **near real time during deployment** (builds the point geometries on-the-fly from hstore key/value pairs in auto refresh map layer)

```
select timer,
       extract(epoch from timer) as id, h.value -> '307' as vess_lat,
       h.value -> '308' as vess_lon, h.value -> '256' as dtis_lat,
       h.value -> '257' as dtis_lon,
       (ST_SetSRID(
           ST_Makepoint(
              (h.value -> '257')::decimal(10,7),
              (h.value -> '256')::decimal(10,7)),4326)) as dtis_geom,
       (ST_SetSRID(
           ST_Makepoint(
             (h.value -> '308')::decimal(10,7),
             (h.value -> '307')::decimal(10,7)),4326)) as geom
from t_reading_hstore_min h
where (h.value -> '256') notnull
  and (h.value -> '256') != '0.0'
  and timer > '2018-05-14' –- or use now() - interval(‘1 day’)
order by timer asc limit 1000;
```

Execution time: ???



The transect data from the query from QGIS



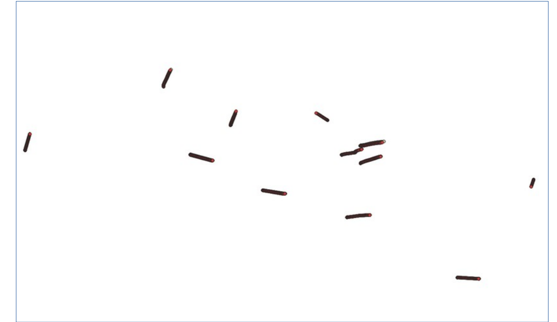
and zoomed in

NIWA
Taihoro Nukurangi

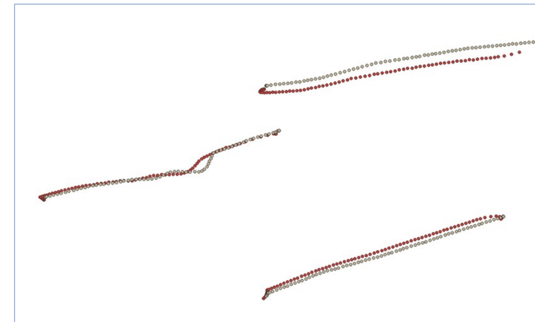
SQL to return vessel and DTIS camera (HiPAP) point locations from 1 min table

Use in QGIS to display camera & vessel positions in near real time
(builds the point geometries on-the-fly from hstore key/value pairs)

```
select timer,
       extract(epoch from timer) as id, h.value -> '307' as vess_lat,
       h.value -> '308' as vess_lon, h.value -> '256' as dtis_lat,
       h.value -> '257' as dtis_lon,
       (ST_SetSRID(
           ST_Makepoint(
              (h.value -> '257')::decimal(10,7),
              (h.value -> '256')::decimal(10,7)),4326)) as dtis_geom,
       (ST_SetSRID(
           ST_Makepoint(
             (h.value -> '308')::decimal(10,7),
             (h.value -> '307')::decimal(10,7)),4326)) as geom
from t_reading_hstore_min h
where (h.value -> '256') notnull
  and (h.value -> '256') != '0.0'
  and timer > '2018-05-14'
order by timer asc limit 1000;
```



The transect data from the query



Zoomed in

Execution time: 50ms

NIWA
Taihoro Nukurangi

On disk data volumes:

Difficult to be accurate, we are still finding new indexes to build to improve performance (which take up space)…

It seems the new approach stores around 70x the number of readings in about 40% of the space used by the old database.

Summary:

Postgresql with Postgis, Timescaledb and hstore extensions works pretty much out of the box, is very fast, and very effective.

We have not (yet) had any instances where we could not reasonably easily and quickly extract the data we desired in the format we needed.

We are pleased and impressed with this suite, very fit for this purpose.

And situation normal – *we have yet to complete the database documentation!*

NIWA
Taihoro Nukurangi

Types of sensors:

While most of the sensors are collecting environmental data, various vessel systems (engines, pumps, etc) also provide telemetry data which is monitored and captured by the DAS, and so ends up in the DAS database.

It is useful to have winch for an instrument deployed off that winch, and to have data such as load, oil pressures and temperatures from engines and pumps, not just from a science perspective, but from a vessel operation perspective.

Similarly – an acoustic scientist who finds noise in their data for a transect can see what else was happening at the time –

eg: engineers starting and testing a backup generator during a transect

NIWA
Taihoro Nukurangi

This work was funded as a NIWA IT CAPEX project

My thanks to Andrea Mari and Simon Wood for getting this done, and within budget!!!

NIWA
Taihoro Nukurangi

Thank you for attending!

Questions?

NIWA
Taihoro Nukurangi