COMP5339: Data Engineering

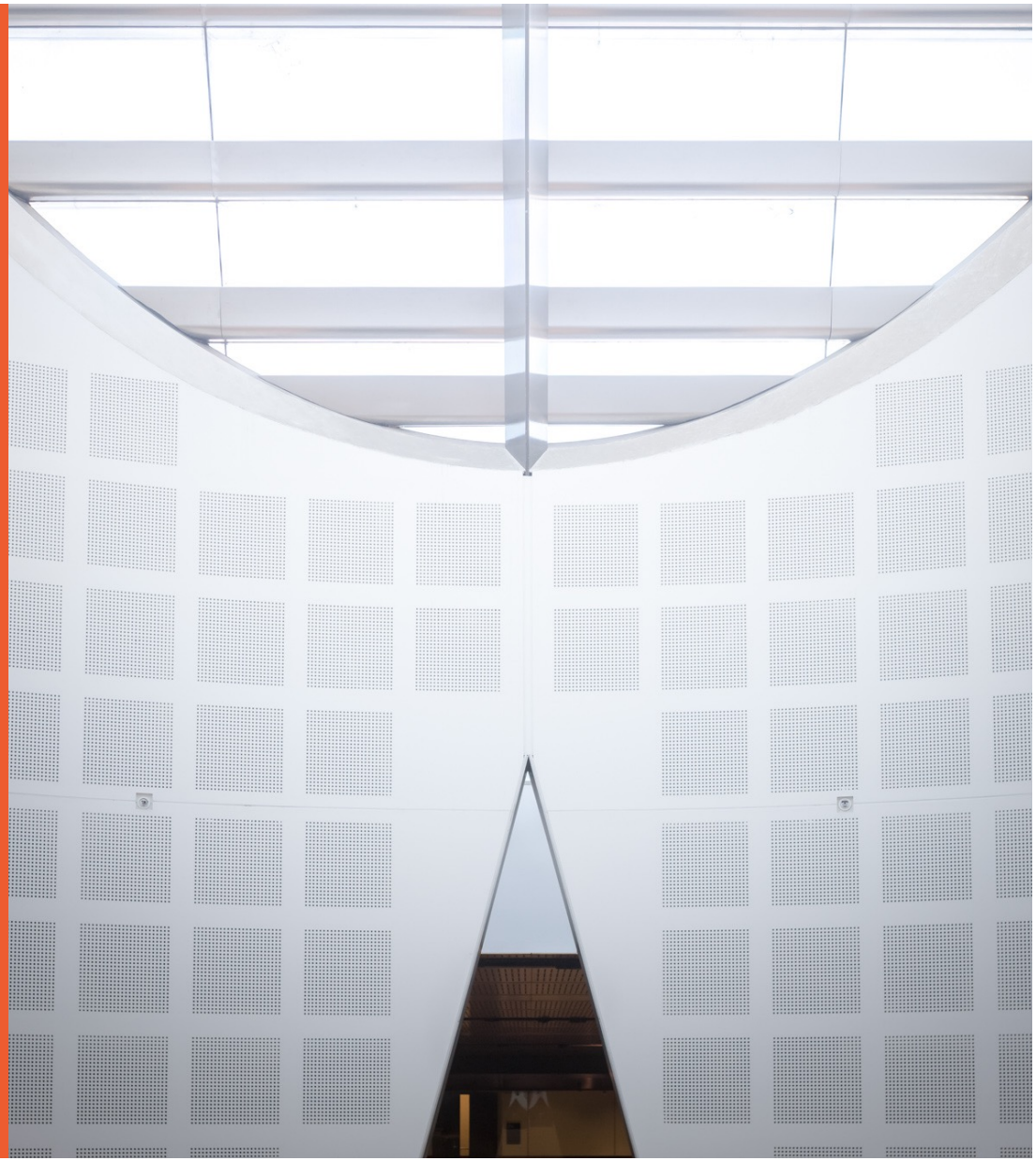
W7: Temporal Data Engineering

Presented by

Uwe Roehm

School of Computer Science

THE UNIVERSITY OF SYDNEY



Temporal Data is Ubiquitous

- Almost all data is marked with time (period or point)
  - Web stores
  - Data warehousing
  - Medical records, loans, ...
  - Sensor data and time series
  - Transport information
  - ...

- How do we represent and analyse temporal data & time series data?

Some Examples

Weather Observations

```
obsdate,  city,      temp,wind,wind_dir
2017-01-1,Adelaide,  21.1,  33,  SW
2017-01-1,Brisbane,  31.6,  13,  NE
2017-01-1,Hobart,    17.3,   9,  SSW
2017-01-1,Melbourne,20.0,  11,  SSW
2017-01-1,Perth,     27.9,  19,  SW
2017-01-1,Sydney,    24.9,  19,  E
2017-01-2,Adelaide,  24.7,  28,  SSE
2017-01-2,Brisbane,  30.4,  15,  NE
2017-01-2,Melbourne,16.3,  19,  S
```

Water Measurements

Station	Date	Level (m)	Discharge (ml/d)	Temp (C)	EC @ 25C (us/cm)
409017	4/04/2009	2.472	7817.866	20.569	49.823
219018	4/04/2009	0.166	0.371		
409204	5/04/2009	0.637	2358.659	20.633	51.75
409017	5/04/2009	2.389	7744.308		52.604
409204	6/04/2009	0.637	2390.783	20.125	51.5
409017	6/04/2009	2.403	7861.138	20.715	52.688
219018	6/04/2009	2.403	0.239		
409204	7/04/2009	0.637	2358.659	19.157	50.24
409017	7/04/2009	2.39	7649.435	20.281	44.875
219018	7/04/2009	0.166	0.119		
409204	4/04/2009	0.653	2409.639	20.181	55.906
409204	8/04/2009	0.628	2271.601	18.583	49.417
409017	8/04/2009	2.368	7299.264	19.925	40.771
219018	8/04/2009	0.162	0.091		
409204	9/04/2009	0.615	2185.795	18.604	49.094

Big Data also means Historic Data

- Businesses can enhance their customer relationships by embracing many of Arbesman's "long data" ideas [Arbesman: *Stop Hyping Big Data and Start Paying Attention to 'Long Data'* (Wired 2013)]:
  - Big data is more about taking a slice in time across many different channels. But long data involves looking at information on a much longer timescale.
  - Many companies have data that goes back 10, 20, or 30 years. A longer-term view can provide information that businesses might miss if they examine data that only goes back five years or less.
  - There's no international standard definition of big data. But the most popular description – Gartner's 3V of high volume, high velocity, and high variety – doesn't explicitly include historical information as a key component of big data.

[Jeff Bertolucci, InformationWeek 2013]

Temporal Databases

Temporal Database Management System

- A temporal database management system (DBMS) is a DBMS that provides built-in support for the time dimension
  - special facilities for storing, querying, and updating data with respect to time.
- A temporal DBMS can distinguish between historical data, current data, and data that will be in effect in the future.
- The intent of a temporal database management system is to reason with time.
  - A temporal DBMS provides a temporal version of SQL

Temporal Databases

- A temporal database stores data relating to time periods and time instances.
  - It offers temporal data types and stores information relating to past, present and future time.
  - For example, it stores the history of a stock or the movement of employees within an organization.

- Differences between a temporal database and a conventional database
  - A temporal database maintains data with respect to time and allows time-based reasoning
  - A conventional database captures only a current snapshot of reality.
    - cannot directly support historical queries about past status and cannot represent inherently retroactive or proactive changes.

- Examples: TimescaleDB, InfluxDB

Concepts in Temporal Databases

理解每个terminology的意思

- Temporal data types
  - Instant | Point — *something happening at an instant of time*
  - Interval — *a length of time; a duration*
  - Period — *an anchored duration of time (a recurring interval of time, e.g., calendar year)*
- Kinds of time
  - User-defined time — *an uninterpreted time value* 用户自定义时间，数据库不解释其意义
  - Valid time — *when a fact was true in the modelled reality* 有效时间，事实在现实中成立的时间
  - Transaction time — *when a fact was stored in the database* 事务时间，事实被记录入数据库的时间
- Temporal statements
  - Current — *now*
  - Sequenced — *at a specific instant or period of time* 针对 某个时间点或时间段 的查询
  - Nonsequenced — *ignoring time*

Cf. https://docs.teradata.com/r/Enterprise_IntelliFlex_VMware/Temporal-Table-Support

Temporal Data Types

- Instant data types:
  - DATE
    - SQL-92: day, month and year of a time instant (from year 1 to 9999)
    - Postgresql: date (no time of day) from 4713 BC to 5874897 AD
  - TIMESTAMP
    - SQL-92: date + time with variable resolution of fractions of a second (default: 1ms)
    - Postgresql: date + time of same range as DATE with 1 ms resolution; optional time zone
  - TIME
    - SQL-92: hours, minutes, seconds and optional fractional digits of second
    - not really a time instant (no date!); in PostgreSQL with 1ms resolution
- Interval data types:
  - Various specification options, eg. Year-Month Intervals: 1-2 (1 year 2 months)

Cf. https://www.postgresql.org/docs/current/static/datatype-datetime.html

Temporal Data Types

- Period data types:
  - It has a beginning bound (defined by the value of a beginning element) and an ending bound.
  - Beginning and ending elements can be DATE, TIME, or TIMESTAMP types, but both must be the same type.
  - The duration that a period represents starts from the beginning bound and extends up to, but does not include, the ending bound.

```
CREATE TABLE Policy (
        Policy_ID           INTEGER,
        Customer_ID         INTEGER,
        Policy_Type         CHAR(2) NOT NULL,
        Policy_Details      CHAR(40),
        Validity            PERIOD(DATE)
)
```

```
INSERT INTO Policy VALUES (
        541008,
        246824626,
        'AU',
        'STD-CH-345-NXY-00',
        PERIOD(DATE '2009-10-01', DATE '2009-10-31')
);
```

- SQL supports time *instants* and *intervals* (but no *periods*)

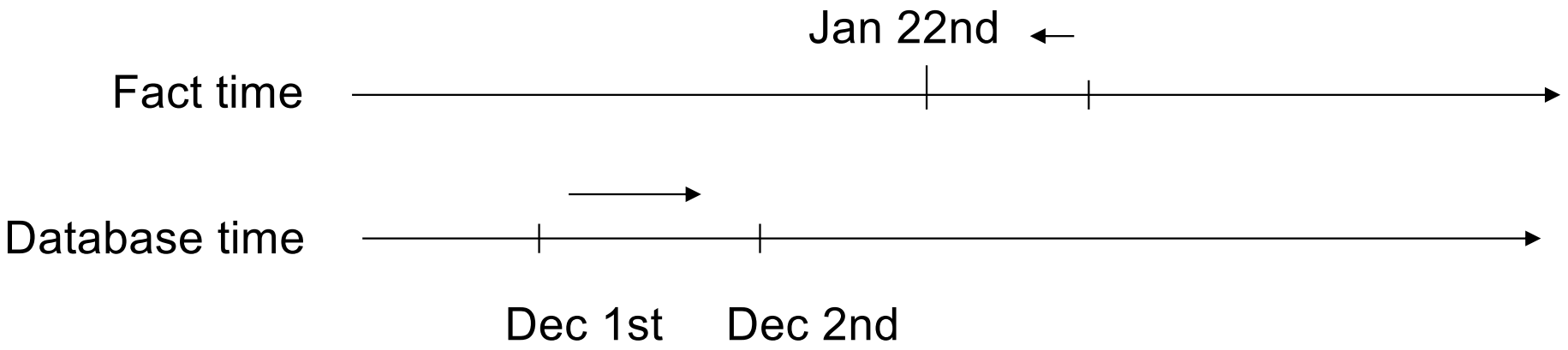
PostgreSQL不支持perio

Cf. https://docs.teradata.com/r/Enterprise_IntelliFlex_VMware/Temporal-Table-Support

Kinds of Time

- User-defined time: *an uninterpreted time value*
  - E.g. a birthdate or a publication time
- **Valid time** records the time when a fact is true in the real world.
  - Can move forward and backward
- **Transaction time** records the history of database activity.
  - Only moves forward (as you cannot go back in history and change things – alas!)
  - Therefore, allows rollback (very useful for auditing)

Two Interpretations of Time



- Database time (or **Transaction time**) only moves forward.
- Fact time (or **Valid time**) can move forward or backward
  - Are we changing history here?
    No, more our 'perception' of history…

Temporal Tables

- Temporal tables store and maintain information with respect to time, and can process statements and queries that include time-based reasoning.
- Temporal tables include one or two special columns, which store time information:
  - A transaction-time column
  - A valid-time column
  - Each of these time dimensions is represented by a column with a period data type.

```
CREATE TABLE Policy (
        Policy_ID           INTEGER,
        Customer_ID         INTEGER,
        Policy_Type         CHAR(2) NOT NULL,
        Policy_Details      CHAR(40),
        Validity            PERIOD(DATE) NOT NULL AS VALIDTIME
)
```

Temporal Tables

- A table with a transaction-time column is called a **transaction-time table**.
- A table with a valid-time column is called a **valid-time table**.
- A table with both a transaction-time and a valid-time column is called a **bitemporal table**.
- A table with neither a transaction-time nor a valid-time column is a **nontemporal table**.

Handling of NOW or UC (Until Changed)

- What should the timestamp be for current data? (valid now until changed)
  - Minimum possible timestamp *(min-timestamp approach)*
    - E.g. '00-00-0000' - quite counter-intuitive
  - NULL value
    - Complicates queries: Comparisons with NULL return FALSE in SQL…
    - Use of NULL as 'unknown' is no longer possible…
  - Maximum possible timestamp *(max-timestamp approach)*
    - E.g. '31-12-9999'
    - Simulates 'the end of time' or 'forever'
  - *Point representation*
    - Set start_time = end_time
- **Max-timestamp** approach is extensively used
- Warning: PostgreSQL supports a 'now' keyword which refers to the transaction's start time, not to 'valid until changed…'; instead use 'infinity' in PostgreSQL

3种办法设置 数据库中"当前有效数据"的时间范围

把 end_time 填成最小可能值，表示数据仍然有

结束时间为空，表示数据仍然有效

给当前数据的结束时间设置最大值，表示数据仍然有效

瞬时事件

Temporal Statements

- Current 只关心 现在的状态，不关心历史变化
  - "What is now?" CURRENT VALIDTIME **SELECT** * **FROM Policy**;
    - E.g. *"How many products do we currently have in stock?"*

- Sequenced 查询 数据的历史变化序列。可以看到每个数据在不同时间点或时间段的状态
  - "What was, and when?"
    - E.g. *"Give the sequence of how many product were in stock."*
    - Or *"When did the stock level fall below X in the past?"*

- Nonsequenced 查询 过去曾经出现过的数据，但不关注顺序
  - "What was at any time?"
    - E.g. *"How many products A did we have at any time in stock?"*

Temporal Support in Databases

THE UNIVERSITY OF SYDNEY

Temporal Support in Databases

- Limited support for temporal data management in DBMSs
  - Conventional (non-temporal) DBs represent a **static snapshot**
  - Management of temporal aspects is implemented by the application
    - Adds additional complexity to application programs
  - Some time data types and functions available in SQL, e.g., DATE, TIME, DATEADD(), DATEDIFF()
    - SQL:2011 added support for temporal tables
    - Still **very limited query support**

How to represent time in (traditional) databases?

- Basic building blocks: SQL data types DATE, TIMESTAMP
- Modelled as a set of time instants/points with a partial order
- Can add attributes of these time data types to any table, interpretation is application-specific
  - Example 1: Person (id INT, name TEXT, birthday DATE);
  - Example 2: Project (id INT, title TEXT, start DATE);
  - Example 3: TripLog(user INT, car INT, when TIMESTAMP, distance INT);

Date and Time in SQL

SQL Type	Example	Accuracy	Description
DATE	'2012-03-26'	1 day	a date (some systems incl. time)
TIME	16:12:05	ca. 1 ms	a time, often down to nanoseconds
TIMESTAMP	2012-03-26 16:12:05	ca. 1 sec	Time at a certain date: SQL Server: DATETIME
INTERVAL	'5 DAY'	years - ms	a time duration

- Comparisons
  - Normal time-order comparisons with '=', '>', <, '<=', '>=', ...
- Constants
  - CURRENT_DATE db system's current date
  - CURRENT_TIME db system's current timestamp

和数据库在的时区的时间相同

- Example:

```
SELECT *
  FROM Measurements
 WHERE obsdate < CURRENT_DATE;
```

Date and Time in SQL (cont'd)

- Database systems support a variety of date/time related ops
  - Unfortunately, not very standardized – a lot of slight differences

- Main Operations
  - **EXTRACT(** *component* **FROM** *date* **)**
    - e.g. EXTRACT(year FROM startDate)
  - **DATE** *string* (Oracle syntax: TO_DATE(*string,template*))
    - e.g. DATE '2012-03-01'
    - Some systems allow templates on how to interpret *string*
  - **+/- INTERVAL**:
    - e.g. '2012-04-01' + INTERVAL '36 HOUR'

- Many more -> check database system's manual

Semi-Temporal Databases

- We want to keep track *since when* a current fact is true.
  - We add timestamp defining the *period of validity* since a point in time

    `Employees(tfn, name, city, …, since)`
  - Q: Meaning of SINCE attribute? => typically [since, now)
    'Ever since day SINCE, supplier *s* has been under contract'
- Employees has become a '***semi-temporal' table***

tfn	name	city	since
E1	Employee1	London	1996-06-01
E2	Employee2	Sydney	1996-08-01
E3	Employee3	New York	2000-10-01
E4	Employee4	Frankfurt	2005-12-31
E5	Employee5	Singapore	2006-12-31

Adding History

- We want to keep the history of when which employees were assigned to what positions.
  - Need to add data about the *period of validity* to a EmpPosition table
  - As there is no 'time period' data type in SQL, we have to simulate it with two DATEs (V_{ts} and V_{te}).

    `EmpPosition(tfn, pos, start_date, end_date)`
  - Q: Open or closed time interval? => typically, V_{ts} is included but V_{te} is not $[V_{ts}, V_{te})$
- EmpPosition has become a ***valid-time table***

tfn	pos	start_date	end_date
E1	assistant	1996-06-01	1996-08-01
E1	consultant	1996-08-01	1998-04-01
E1	senior consultant	1998-04-01	1998-10-01
E2	manager	2000-10-01	9999-12-31
E3	sysadmin	2005-12-31	9999-12-31

<- side issue: how to represent 'till now'?

SQL:2011 added support for Valid and Transaction Time

- SQL:2011 adds period definitions as metadata to tables (and not as a new data type!)
- A period is a conceptual grouping of a physical start time and end time attribute/column
- For instance, a period `EPeriod` built from existing attributes `EStart` and `EEnd`

```
... PERIOD FOR EPeriod (EStart, EEnd) ...
```

- A period is by default a half-closed interval `[EStart, EEnd)`
- Distinction between
  - **application time** (= valid time)
  - **system time** (= transaction time)
- So far supported only by few commercial DBMS, such as IBM DB2

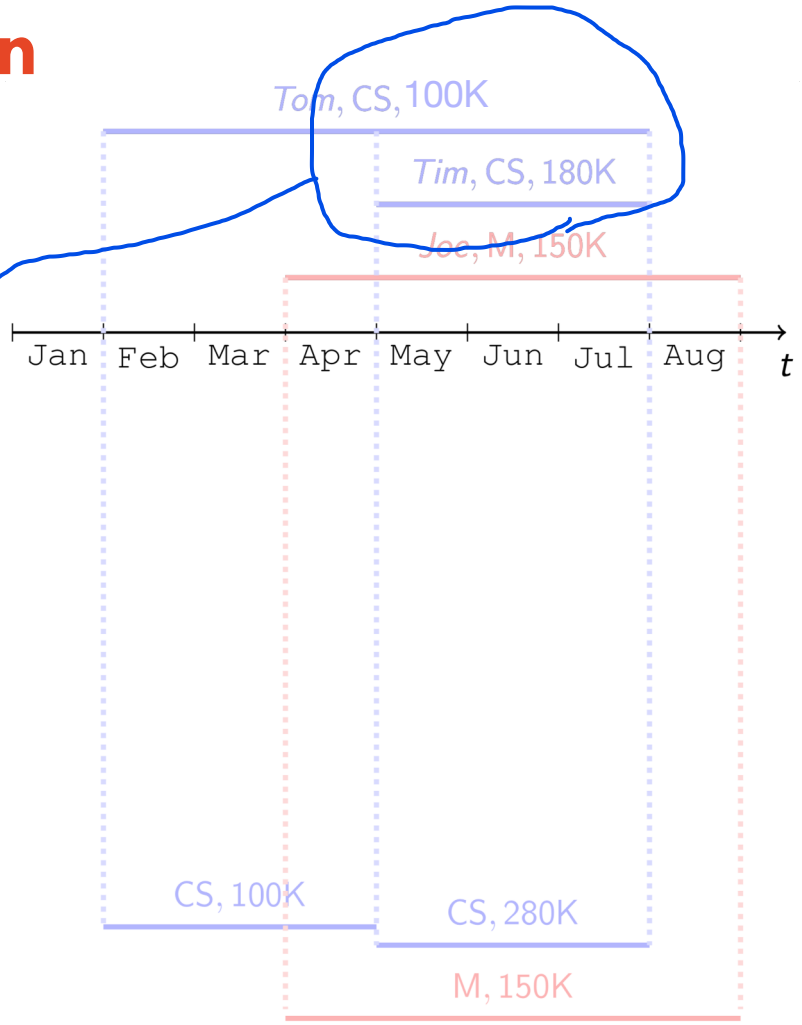
Example: Temporal Aggregation

- **Input:** external project funding

Name	Dept	Budget	TimePeriod
Tom	CS	100 K	[Feb, Jul]
Tim	CS	180 K	[May, Jul]
Joe	M	150 K	[Apr, Aug]

- **Query:** Amount of external funding per department?
- **Output:** Temporal aggregation (at each point in time)

Dept	Budget	TimePeriod
CS	100 K	[Feb, Apr]
CS	280 K	[May, Jul]
M	150 K	[Apr, Aug]



[Example from: J. Gamper, eBISS 2017]

PostgreSQL

- Key features
  - Supports all standard SQL temporal data types (points and intervals) **in addition:**
  - Range types (periods)
  - Indexes on range types
  - Temporal constraints using GiST indexes
  - Predicates and functions for range types

  - http://www.postgresql.org/docs/devel/static/rangetypes.html

Tables with Range Types in PostgreSQL

- New data type DATERANGE to represent (time) intervals

```
CREATE TABLE Emp (
    eName     VARCHAR,
    ePeriod   DATERANGE,
    eDept     VARCHAR
);
INSERT INTO Emp VALUES ('Anton', '[2010,2014]', 'SCS');
```

Indexes on Range Types in PostgreSQL

- Provides a powerful, extensible index: **Generalised Search Tree** (GiST)
  - On periods and combinations of period and other attributes
    (eg. GiST is also very useful for spatial data)

```
CREATE INDEX Emp_idx ON Emp USING GIST (EPeriod)

CREATE INDEX Emp_idx ON Emp USING GIST (EName, EPeriod)
```

- Improves range indexing for many predicates
  e.g., `EQUAL, OVERLAP, LESS THAN, CONTAINS, . . .`

Query Examples on PostgreSQL Range Types

- SQL with predicates, e.g., OVERLAPS (&&), BEFORE, AFTER

```
SELECT *
  FROM Emp JOIN Dept ON EPeriod && DPeriod
```

```
SELECT *
  FROM Dept
 WHERE upper_inf(DPeriod) = TRUE
```

- Some additional functions on ranges: UNION (+), INTERSECTION (*), DIFFERENCE (-)

```
SELECT '[2010, 2013)' + '[2012, 2015)'
```

Temporal Support in Current DBMS

age > 18

DBS	Period	Keys	Update	Predicates	Queries
SQL:2011	✓	✓	✓	✓	✗
IBM DB2	✓	✓	✓	✓	✗
MySQL	✗	✗	✗	✗	✗
MS SQL Server	✗	✗	✗	✗	✗
PostgreSQL	✓	~	✗	✓	✗
Oracle	✓	✓	✓	✓	✗
Teradata	✓	✓	✓	✓	~

[J. Gamper, eBISS 2017]

Summary

- Temporal data is ubiquitous – but only limited support in database systems

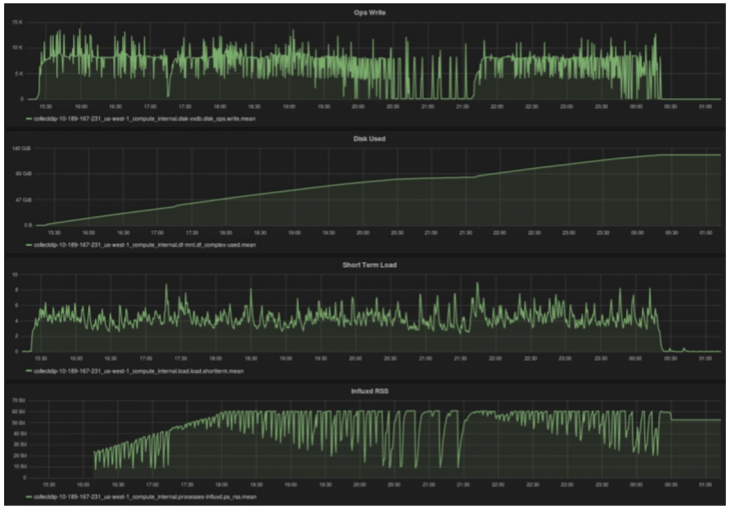
- Challenges:
  - Limited temporal data types in SQL
  - Limited support for temporal statements / queries
    - Time Periods not part of standard SQL, hence such queries difficult to represent and encode
  - Timeseries support
  - Transaction vs. Valid Time

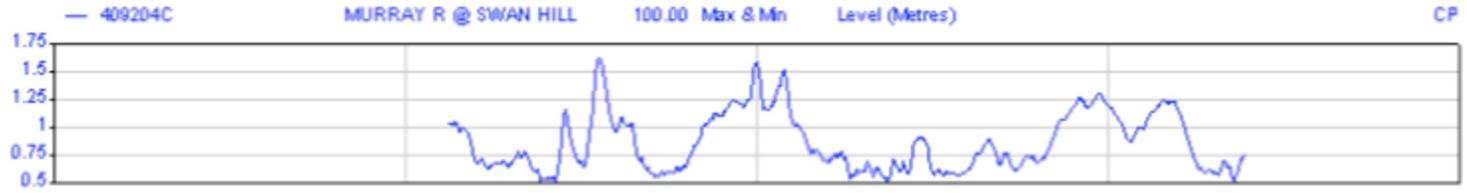
Representing Time Series Data

THE UNIVERSITY OF SYDNEY

Timeseries Data

- Series of observations (data points) made sequentially over time at discrete time points, usually at equal intervals
- Examples
  - weather observations
  - trip data
  - road usage
  - sleep patterns
  - cytometry / panel data





Working with Time Series Data

- Data ingress (file import, database access, web scraping, …)
- Storage
  - Standard file formats, databases
- Analysis
  - In database vs. applications (eg. Python programs)
- Visualisation, reporting

Timeseries Data – Storage Approaches

- Point-based representation
  - Multiple rows with atomic data types
- Sequence-based representation
  - Single row with array of time point data
  - Requires array datatype
    - UNNEST array operation, array_agg with filter
- Dedicated time-series database
  - Eg. TimescaleDB, InfluxDB, or also Scidb

Point-Based Representation in Flat Table

- Each tuple is timestamped with a time point/instant
- Most basic and simple data model
- Timestamps are atomic values and can be easily compared with =, !=, <, >, ≥, ≤
- Syntactically different relations store different information
- *If a fact is valid at several time points, multiple tuples are needed*

- Good for time domain analysis
- Allows for indexing

```
CREATE TABLE Observations (
    ts              TIMESTAMP,
    station         INT,
    temperature     FLOAT,
    wind            CHAR(2)
);
```

ts	station	temperature	wind

Working with Point-Based Representations

- Individual INSERT, *UPDATE* statements (typically append-only)
- OLAP-style queries for filtering, aggregation, grouping

```
SELECT station, AVG(temperature)
  FROM Observations
 WHERE ts BETWEEN start_ts AND end_ts;
```

- Careful with granularity and timezone of data
- Joins to dimension tables
- Indexing – typically B-Tree based

```
CREATE INDEX TimeIdx ON Observations(ts);
```

Sequence-Based Representation in Nested-Table

- Single row with array of time point data
- Requires array datatype

  => outside 1NF idea of relational model
- How to represent discrete time points now?
  - Part of the array? So tuple (ts, value)?
  - Or implicit by position?
  - Then need to make sure that different arrays represent same data and time points
- Special functions needed to
  - Create arrays from set of data array_agg()
  - Pivot data back into sets UNNEST

```
CREATE TABLE Observations2 (
    station      INT,
    temperature  FLOAT[],
    wind         CHAR(2)[],
    obs_start    TIMESTAMP,
    obs_end      TIMESTAMP
);
```

station	temp	wind	start, end
	[...]	[...]	,

Cf. https://www.postgresql.org/docs/9.2/functions-array.html

Working with Array-Data

- INSERTs with array data

```
INSERT INTO Observations2
    VALUES(4711,'{79,82,90,69,75,80,81}','2018-01-01',..);
```

- UPDATE of array content

```
UPDATE Obs2 SET temperature[4] = 22.5 WHERE station=..
```

- Indexing => GIN (generalised inverted index)

```
CREATE INDEX TimeIdx2 ON Observations2 USING GIN("ts")
```

- Querying
  - Turning arrays to rows: Unnesting

```
SELECT UNNEST(temperature) FROM Observations2 WHERE ..
```

数组 → 行 (Unnest)

行 → 数组 (array_agg)

  - Turn rows into array data: array_agg()