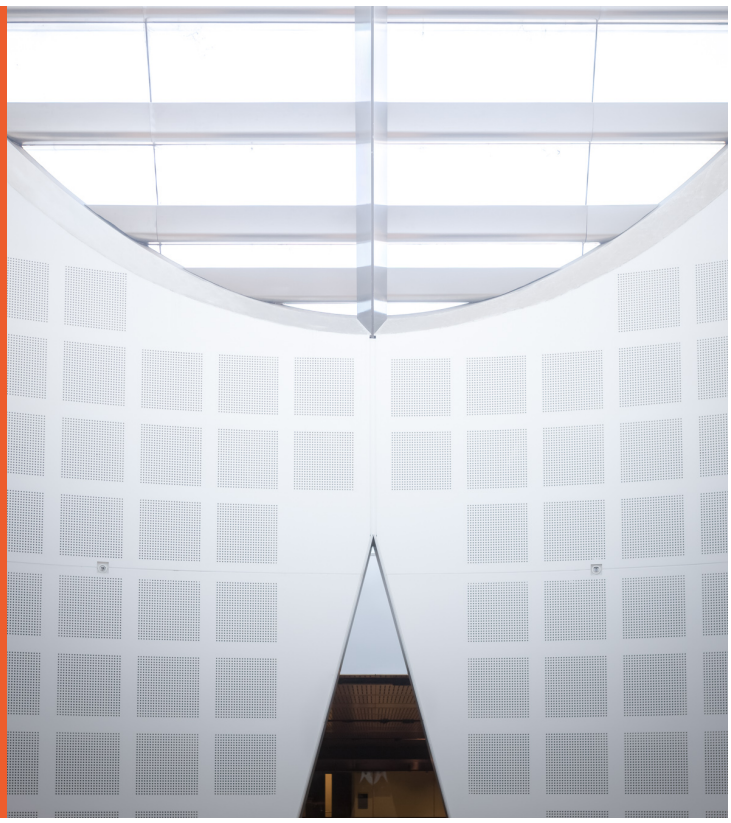
DATA2001: Data Science, Big Data and Data Diversity

W3: Accessing Data in Relational Databases; Introduction to SQL

Presented by
A/Prof Uwe Roehm
School of Computer Science

THE UNIVERSITY OF SYDNEY



Overview of Week 3

THE UNIVERSITY OF SYDNEY

Last Week: Data Analysis with Python

Last Week's Objective

Using Jupyter/iPython for cleaning and exploring a data set programmatically.

Lecture

- Data types, cleaning, preprocessing
- Descriptive statistics, e.g., median, quartiles, IQR, outliers
- Descriptive visualisation, e.g., boxplots, confidence intervals

Readings

- Data Science from Scratch: Ch 4-5

Exercises

- matplotlib: Visualisation
- numpy/scipy: Descriptive stats

TODO in W2

- Explore the survey data

Today: Accessing Data in Relational Databases and Introduction to SQL

Objective

To be able to extract a data set from a database, as well as to leverage on the SQL capabilities for in-database data summarisation and analysis.

Lecture

- Data Gathering reprise
- Relational Database Concepts
- SQL querying
- Summarising data with SQL

Readings

- Data Science from Scratch, Ch 23

Exercises

- Creating database / tables
- SQL Querying (Postgresq and SQL/Grok)
- Data Summarization using SQL

TODO in W3

- Data Exploration using SQL

Announcement

SQL Online Helpdesk available in Grok

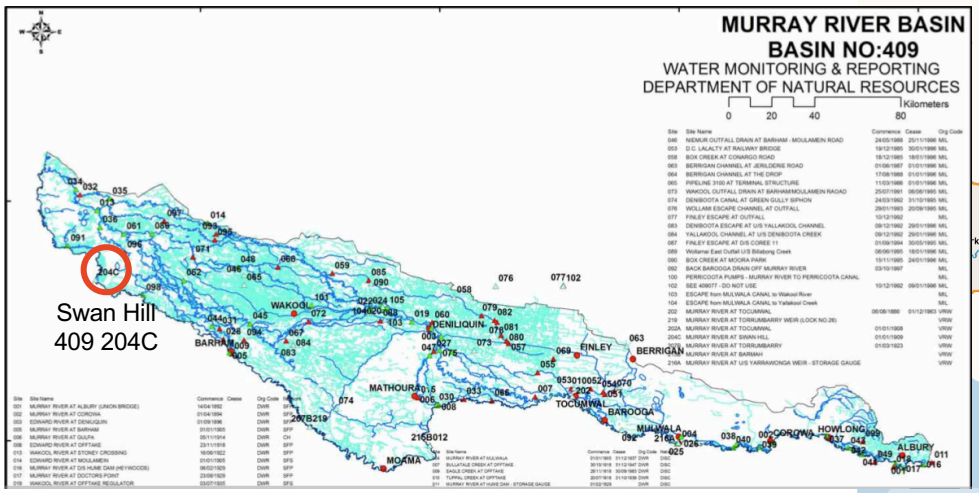
Mon, Tue, Thu 3-4 pm

Wed, Fri 1-2 pm

Motivating Example

THE UNIVERSITY OF SYDNEY

Motivating Example: Water Data about Murray River Basin in NSW



[Source: www.waterinfo.nsw.gov.au] [https://realtimedata.waternsw.com.au] [mdba.gov.au]

- automatic monitoring stations
- periodically send data about water level, water flow, water temperature, salinity…

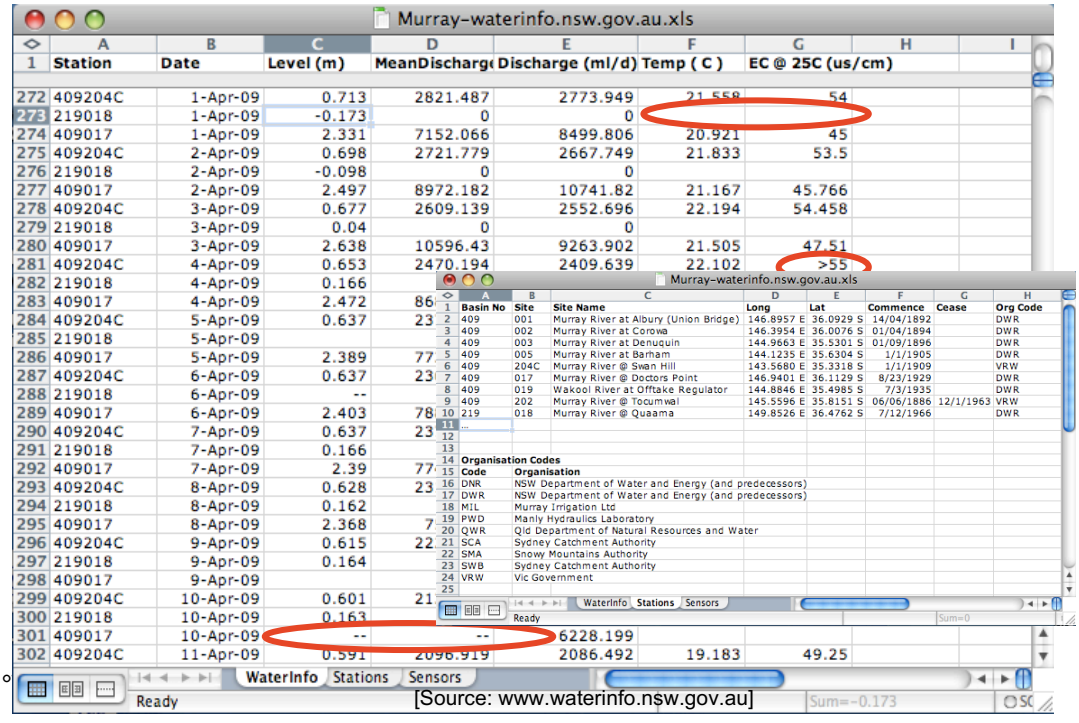
Approach 1: CSV Files

- Just copy the data in tab-delimited text files
- Send those around

- Pros:
  - On Unix systems, you can apply all kinds of command-line tools
  - Easy to import into a spreadsheet program

- Cons:
  - No clear standard for CSV files, especially with regard to integration of meta-data (CSV files are not self-describing)
  - No security, no data integrity, easy to manipulate or to corrupt

Approach 2: Spreadsheet



[Source: www.waterinfo.nsw.gov.au]

Issues with both Approaches

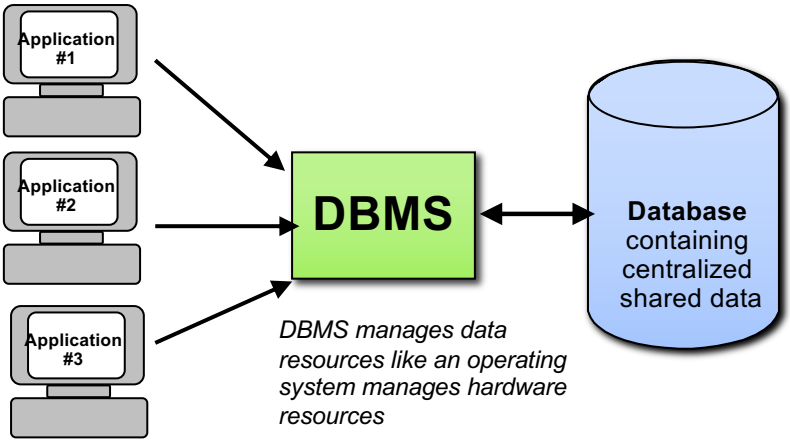
- Both approaches are not ideal
  - CSV files are **missing meta-data** and are easy to manipulate/falsify
  - Spreadsheets allow arbitrary content in each cell and are also easy to manipulate; combining data from different sheets is complicated
  - often data repeated in different files (**redundancy**)
    - Major problem: when data changes in one file, could cause inconsistencies
    - Compromises **data integrity**
  - limited in the size of data
- Makes sharing of data very hard

- Can we do better?

Databases

Solution: The Database Approach

- Central repository of shared data
- Data is managed by a Database Management System (DBMS)
- Stored in a standardized, convenient form



What is a Database?

A database is a shared collection of logically related data and its description.

The database represents the **entities** (real-world things), the **attributes** (their relevant properties), and the logical **relationships** between the entities.

Databases in the Internet Age…

- Ebay (in 2005)
  - More than 100 back-end databases
  - ca. 5 billion SQL/day
- Salesforce.com
  - ca. 1.3 billion transactions per day
- Pinterest
  - Started with 1 MySQL database
  - Now 180+ web servers, 240 API servers, 88 MySQL DBMSs, …
- Wikipedia: (as of Feb 2015 - http://stats.wikimedia.org/EN/Sitemap.htm)
  - over 400 servers
  - 237 languages, millions of articles, multiple GB, 3 million edits/month
- SkyServer: Sloan Digital Sky Survey (optical spectra of stars, galaxies, quasars)
  - public available "virtual telescope", including SQL access, hosted on SQL Server, >125 TB
- UCSC Genome Browser
  - providing public MySQL access to various genomes, e.g., Human Ref Genome

Main Advantages of Databases

- **Program-Data Independence**
  - Metadata stored in DBMS, so applications don't need to worry about data formats
  - Data queries/updates managed by DBMS so programs don't need to process data access routines
  - Results in:
    - Reduced application development time
    - Increased maintenance productivity
    - Efficient access
- **Minimal Data Redundancy**
  - Leads to increased data integrity/consistency

Advantages of Databases (cont'd)

- Improved Data Sharing
  - Different users get different views of the data
  - Efficient concurrent access
- Enforcement of Standards
  - All data access is done in the same way
- Improved Data Quality
  - Integrity constraints, data validation rules
- Better Data Accessibility/ Responsiveness
  - Use of standard data query language (SQL)
- Security, Backup/Recovery, Concurrency
  - Disaster recovery is easier

Relation Database Systems

- store data in **tables** as **rows** with multiple **attributes**
- rows *of the same format* form a 'table' (**relation: a set of tuples**)
- Every relation has a **schema**, which describes the columns, or fields, and their types
- A relational database is a collection of such tables
  (which typically are related to each other by **key** attributes)
- Example:

primary key →

Student				
sid	name	email	gender	address
5312666	Jones	ajon1121@cs	m	123 Main St
5366668	Smith	smith@mail	m	45 George
5309650	Jin	ojin4536@it	f	19 City Rd

schema (header row) *data* (rows)

Primary Key

- A primary key is a unique attribute which the database uses to identify a row in a table.
- It is a unique, auto-incrementing ID which is filled in by the database - in other words it is NEVER NULL
  - (NULL has the special meaning in databases of “unknown” or “not given”)
- A primary ID number will only ever be issued once

Authors

authorID	given_name	family_name
1	Charles	Dickens
2	Virginia	Woolf

Foreign Key

- When we need to refer to a record in a separate table we reference its ID as a *foreign key*.
- A **foreign key** is defined in a second table, but it refers to the primary key or a unique key in the first table.

Books

bookID	title	authorID
1001	Orlando	2
1002	David Copperfield	1

Kinds of Relationships

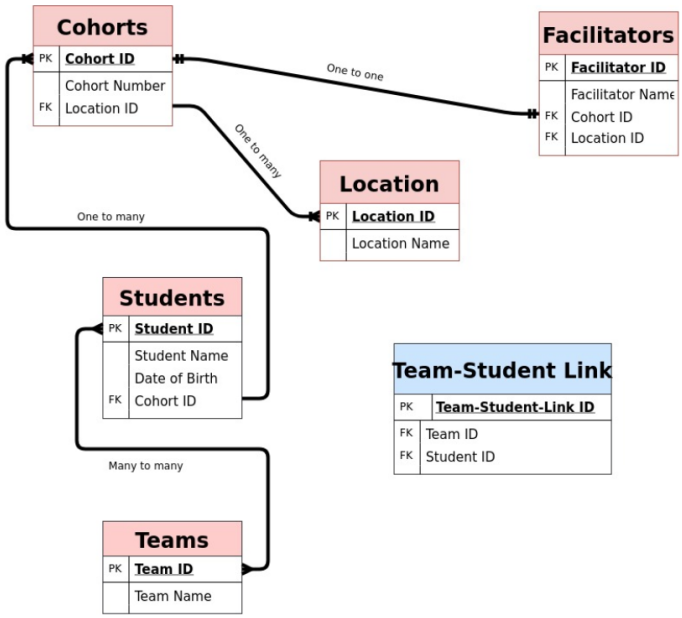
- **One-One Relationship (1-1 Relationship):** One-to-One (1-1) relationship is defined as the relationship between two tables where both the tables should be associated with each other based on only one matching row.

- **One-Many Relationship (1-M Relationship):** The One-to-Many relationship is defined as a relationship between two tables where a row from one table can have multiple matching rows in another table.

- **Many-to-Many Relationship (M-N Relationship)**

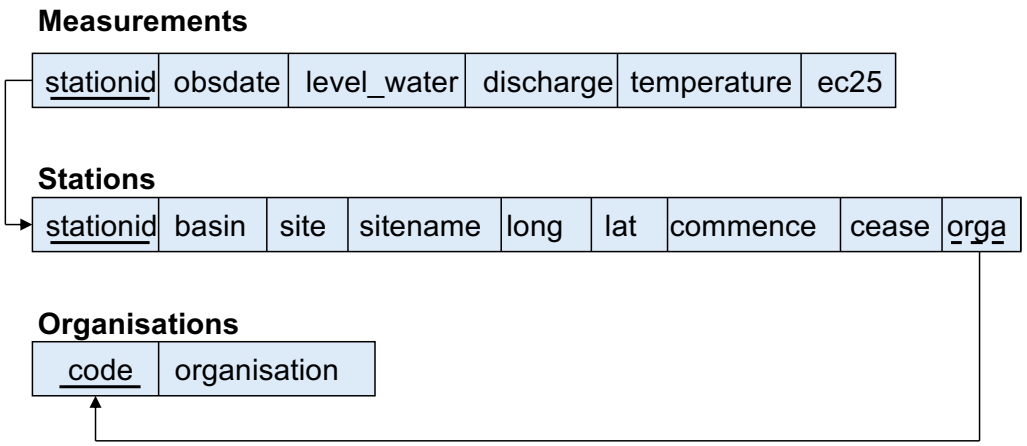
Entity Relationship Diagram

- A *normalised* relational database tries to avoid redundancies
  - Every fact ideally stored only once
  - That's some difference to spreadsheet where lots of data gets repeated and then tends to become inconsistent
- Different notations used…



WaterInfo Example as a Relational Database – Option 1

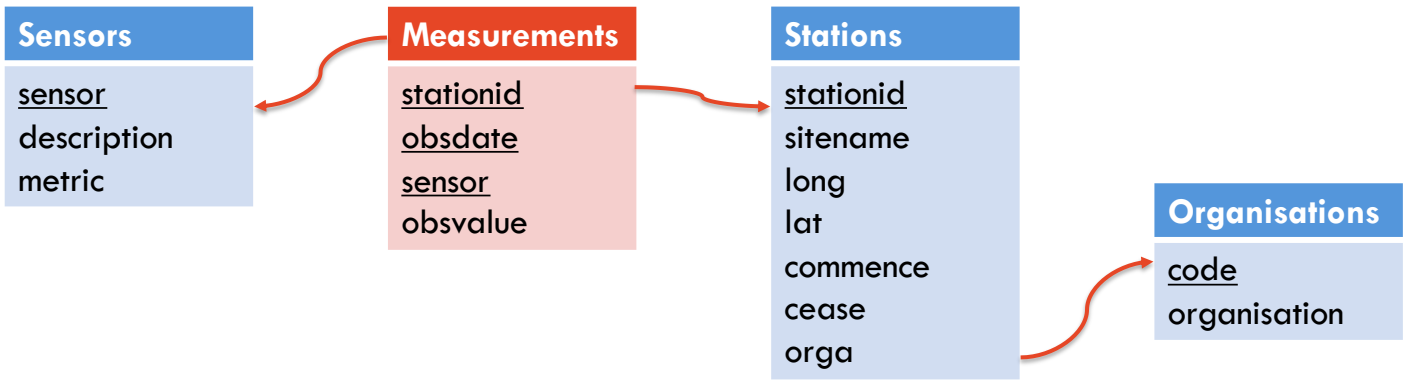
- **Design Option 1**: Straight-forward 1:1 mapping
  - Because a spreadsheet is in principle a table, we can always map it 1:1
  - Some problems though: e.g. the Station <-> Basin+Site mapping



- A lot of NULL values for any missing measurement or if no sensor

Modeling our Water Data Set

- **Design Option 2**: Normalised Relational Schema
  - measurements are facts, other data describes the dimensions
  - measurement entries get 'folded' into separate rows of the fact table
  - allows us to avoid NULLs as much as possible, but hard to read



Some Remarks

- Not all tables qualify as a relation:
  - Every relation must have a unique name.
  - Attributes (columns) in tables must have unique names.
    => The order of the columns is irrelevant.
  - All tuples in a relation have the same structure;
    constructed from the same set of attributes
  - Every attribute value is atomic (not multivalued, not composite).
  - Every row is unique
    (can't have two rows with exactly the same values for all their fields)
  - The order of the rows is immaterial

SQL

SQL (Structured Query Language) Example

- The *working-horse* command: **SELECT – FROM – WHERE**
- retrieves data (rows) from one or more tables of a relational database that fulfill a search condition
- Example 1:

```
SELECT *
  FROM Stations
```

- Example 2:

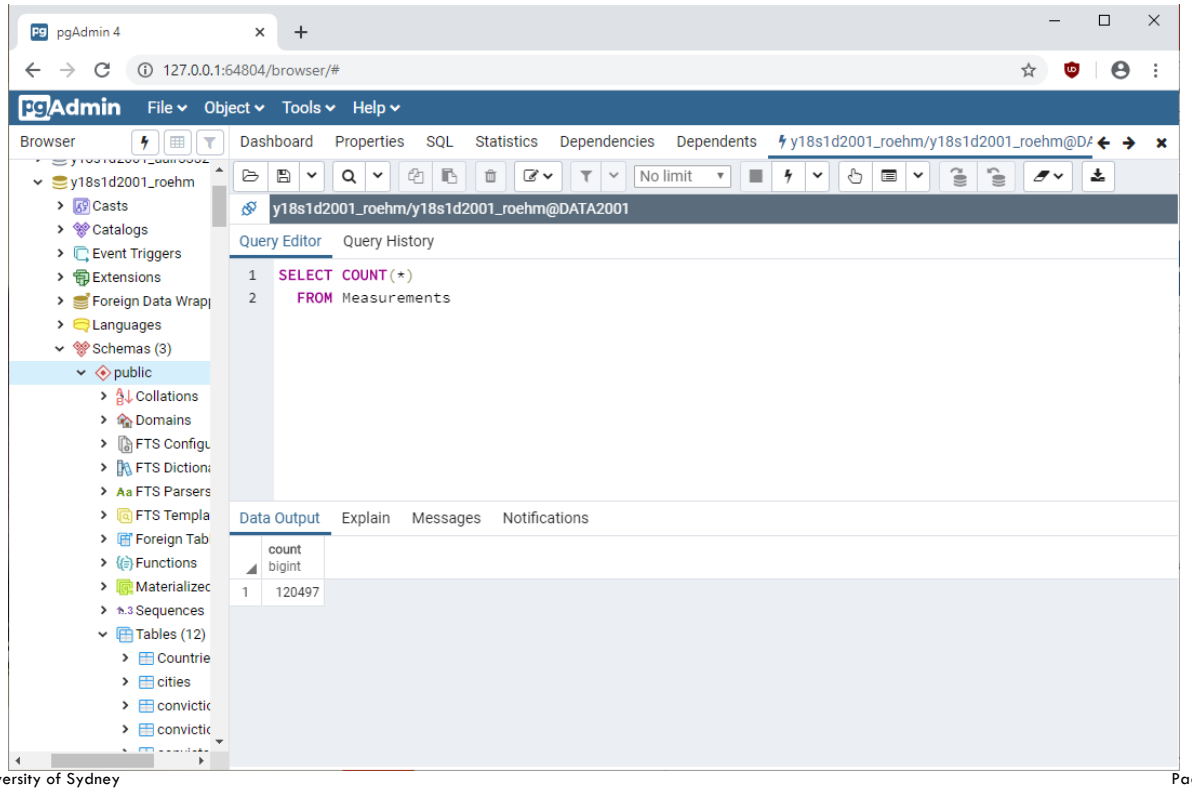
```
SELECT sitename, orga, commence
  FROM Stations
 WHERE stationId = 409001
```

- Example 3:

```
SELECT COUNT(*)
  FROM Stations
 WHERE orga = 'SCA'
```


Declarative Queries: "What" not "How"

- It is convenient to indicate declaratively *what* information is needed, and leave it to the system to work out *how* to process through the data to extract what you need
  - Programming is hard, and choosing between different computations is hard
- Users should be offered a way to express their requests declaratively
  - A query language can be based on logic
  - Select…where…



SELECT Statement

- SQL: "Lingua Franca" of the database world
- SELECT: retrieves data (rows) from one or more tables that fulfill a search condition
- Clauses of the SELECT statement:
  - **SELECT** Lists the attributes (and expressions) that should be returned from the query
  - **FROM** Indicate the table(s) from which data will be obtained
  - **WHERE** Indicate the conditions to include a tuple in the result
  - **GROUP BY** Indicate the categorization of tuples
  - **HAVING** Indicate the conditions to include a category
  - **ORDER BY** Sorts the result according to specified criteria
- The result of an SQL query is a relation

More SELECT Statement Options

SQL Statement	Meaning
SELECT COUNT(*) FROM *T*	count how many tuples are stored in table *T*
SELECT * FROM *T*	list the content of table *T*
SELECT * FROM *T* LIMIT *n*	only list *n* tuples from a table
SELECT * FROM *T* ORDER BY *a*	order the result by attribute *a* (in ascending order; add DESC for descending order)

SQL Data Types

- Integers
  - Standard integer arithmetic and comparisons available
- Floats, Numeric
  - Floating point numbers with many mathematical operators and functions
- Strings (CHAR, VARCHAR)
  - SQL string literals must be enclosed in single quotes ('like this')
  - CHAR: fixed length; VARCHAR: variable length strings up-to max length
  - String comparison is case-sensitive
  - Pattern matching with LIKE operator and % and _ placeholders
  - String concatenation: || (eg. 'hello ' || 'there')
- Date, Timestamp

Comparison Operations

- Comparison operators in SQL: **= , > , >= , < , <= , != , <>, BETWEEN**
- Comparison results can be combined using logical connectives: **and, or, not**
- Example 1:

```
SELECT *
  FROM Tablename
 WHERE     ( AttName1 BETWEEN -90 AND -50 )
       AND
           ( AttName2 >= -45 )
       AND
            ( AttName3 = 'H168' );
```

- Example 2:

```
SELECT *
  FROM Stations
 WHERE siteName LIKE 'Murray River%';
```


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
- 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*
    - Oracle syntax: TO_DATE('01-03-2012', 'DD-Mon-YYYY')
  - **+/- INTERVAL**:
    - e.g. '2012-04-01' + INTERVAL '36 HOUR'
- Many more -> check database system's manual

NULL Values

- Tuples can have missing values for some attributes, denoted by **NULL**
  - Integral part of SQL to handle missing / unknown information
  - **null** signifies that a value *does not exist*, it does *not mean "0" or "blank"*!
- The predicate **is null** or **is not null** can be used to check for nulls
  - e.g. Find measurements (Option1 storage) with an unknown discharge value.

```
SELECT *
  FROM Measurements
 WHERE discharge IS NULL
```

- Consequence: Three-valued logic
  - The result of any arithmetic expression involving null is null
    - e.g. 5 + null returns null
  - However, (most) aggregate functions simply ignore nulls

NULL Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - e.g. *5 < null* or *null <> null* or *null = null*

a	b	a = b	a AND b	a OR b	NOT a	a IS NULL
true	true	true	true	true	false	false
true	false	false	false	true	false	false
false	true	false	false	true	true	false
false	false	false	false	false	true	false
true	NULL	unknown	unknown	true	false	false
false	NULL	unknown	false	unknown	true	false
NULL	true	unknown	unknown	true	unknown	true
NULL	false	unknown	false	unknown	unknown	true
NULL	NULL	unknown	unknown	unknown	unknown	true

- Result of **where** clause predicate is treated as false if it evaluates to unknown
  - e.g: **select** sid **from** enrolled **where** grade = 'unknown'
    ignores all students without a grade so far

Summarising Data with SQL

Summarising a Database with SQL

- SQL covered so far merely allows simple exploring and retrieving of a data set
- But we can do more with SQL:
  - Data categorization and **aggregation**
  - **Complex filtering**
  - **Joining** multiple tables
  - Nested queries
  - Ranking
  - Etc.

- Today just a first introduction
  - Next week more details such as grouping queries
  - Basis of data summarisation is the GROUP BY clause

SQL Aggregate Functions

SQL Aggregate Function	Meaning
COUNT(*attr*) ; COUNT(*)	Number of *Not-null-attr* ; or of all values
MIN(*attr*)	Minimum value of *attr*
MAX(*attr*)	Maximum value of *attr*
AVG(*attr*)	Average value of *attr* (arithmetic mean)
MODE() WITHIN GROUP (ORDER BY *attr*)	mode function over *attr*
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY *attr*)	median of the *attr* values
…	…

Example:

```
SELECT AVG(obsvalue)
  FROM Measurements
 WHERE stationid = 409001 AND sensor = 'flow';
```

JOIN: Querying Multiple Tables

- Often data that is stored in multiple different relations must be combined
- We say that the relations are **joined**
  - **FROM** clause lists all relations involved in the query
  - join-predicates can be explicitly stated in the **where** clause; do not forget it!
- Examples:
  - Produces the cross-product *Table1* x *Table2*

```
SELECT *
  FROM Table1, Table2;
```

  - Find the start date and end date of all epochs abbreviated with 'nov04':

```
SELECT Field1, Field2
  FROM Table1 t1, Table2 t2
 WHERE t1.field1Id = t2.field2Id
       AND t1.field3Id = t2.field4Id;
```


SQL Join Operators

- SQL offers join operators to directly formulate the natural join, equi-join, and the theta join RA operations.
  - R **natural join** S
  - R [**inner**] **join** S **on** <join condition>
  - R [**inner**] **join** S **using** (<list of attributes>)
- These additional operations are typically used in the from clause
  - List all details of the first three measurements including station details.

```
SELECT *
  FROM Stations JOIN Measurements USING (stationid)
 LIMIT 3;
```

  - Which measurements were taken at stationed 409204 in 2016?

```
SELECT *
  FROM Measurements INNER JOIN Stations USING (stationid)
 WHERE stationid = 409204 AND extract(year from obsdate)=2016;
```

Natural Join

- The **natural join** between two tables is a join that combines any tuples matching on *common attributes* having *same values* as implicit join condition
  - typically the case for a Primary Key – Foreign Key Relationship
  - Example:

```
SELECT obsdate, obsvalue, metric
  FROM Measurements NATURAL JOIN Sensors
```

- **Careful** – two major pitfalls:
  - Matches any common attributes – whatever has the same name, must have the same value; so if, e.g., two tables have both a name attribute, this would have to match two, whether wanted or not
  - If there are no common attributes, it computes *cross-product* of two tables, which might look to you like a valid result, but is typically not what you want…
  - Hence: if in doubt, use INNER JOIN and formulate the join condition explicitly

Outlook: Advanced SQL

- OLAP Queries
  - multiple groupings by one query: GROUPING SETS, CUBE, ROLLUP
- Window Queries
  - Can define windows ('groups') per each row, incl. relative and overlapping
  - Allow to order data in a window plus new order-dependent aggregate fcts.
  - eg. moving average, cumulative aggregation, ranking, n-tiles, top-k
- Recursive Queries
  - Needed for working with recursive structures such as trees or graphs
- User-defined 'stored procedures'
  - UDFs (user-defined functions), UDAs (user-defined aggregates)
  - Allow to execute arbitrary code close to the data inside DBMS

Review

THE UNIVERSITY OF SYDNEY

Tips and Tricks

- SQL provides *declarative* querying
  - can be very powerful & fast if you are familiar with SQL
  - but lacks good integration with iterative DM/ML algorithms or visualisation
    - typically still requires data processing outside dbms
- Schema required first and schema can be limiting
  - So be careful with consistency constraint and typing
  - Schema can evolve though (ALTER TABLE statement)
- Careful with NULL values as they make queries difficult
  - Better **not** to store something rather than to have NULL 'placeholder'
- RM not well fitted for *semi-structured data* such as JSON or XML
  - Yes, there are extensions nowadays (cf. XML and JSON in postgres docu)
  - Recent rise of NoSQL databases

Next Week: Grouping Queries and Data Gathering

- So far we looked at the DBMS as the *sink* for data
  (DBMS: Database Management Systems)
- But SQL databases are also a common source for data analysis
- Two approaches:
  1. push queries into DBMS, let it work and just retrieve query result
  2. extract large chunks or even all data, and then analyse outside DBMS

- We will look into this in Week 4:
  how to use Python to retrieve data from a relational database