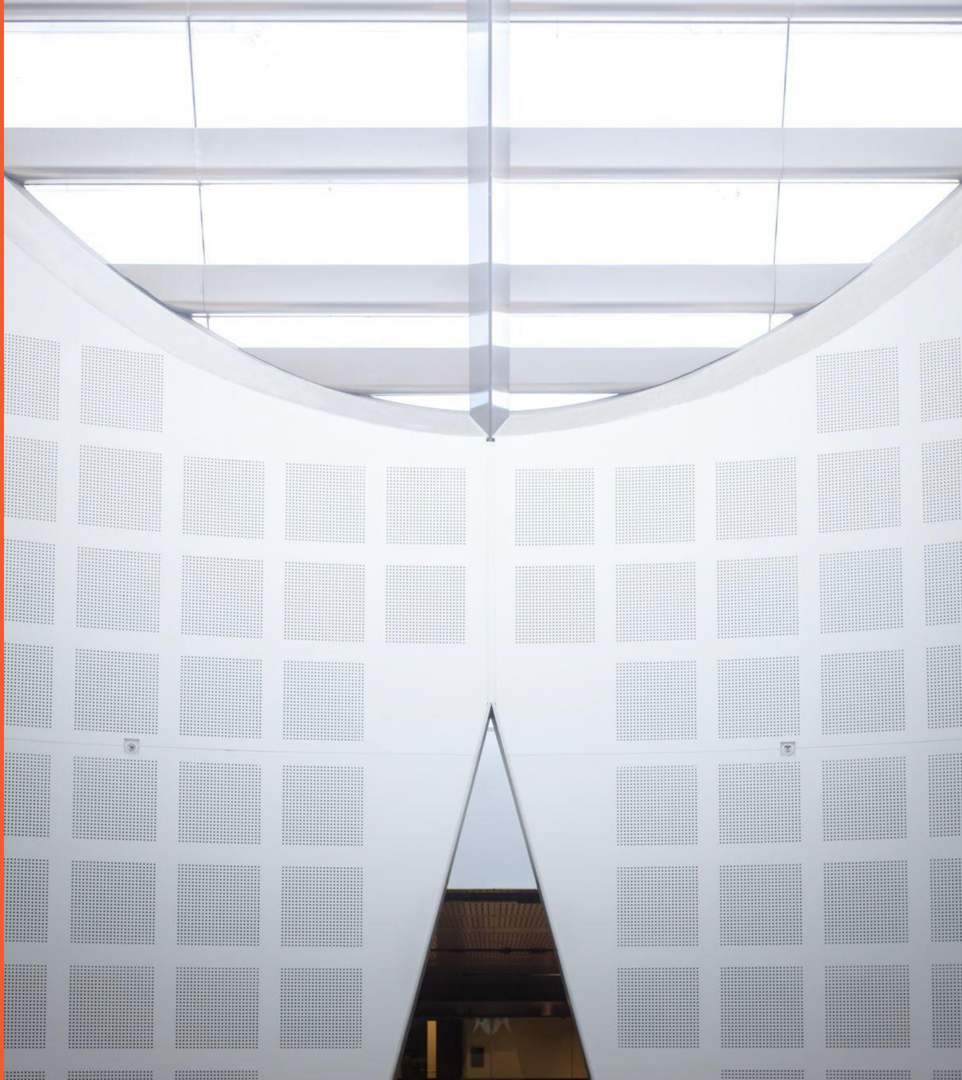
COMP5310: Principles of Data Science

W4: Data Extract, Data Transformation and Storage

Presented by
Claire Hardgrove
School of Computer Science



Overview of Week 4

THE UNIVERSITY OF SYDNEY

Today: Data Transformation and Storage with Python and SQL

Objective

Use Python and PostgreSQL to extract, clean, transform and store data.

Lecture

– DB Access from Python
– Data cleaning and preprocessing
– Data Modeling and DB Creation
– Data Loading/Storage

Readings

– Data Science from Scratch: Ch 9 + 10

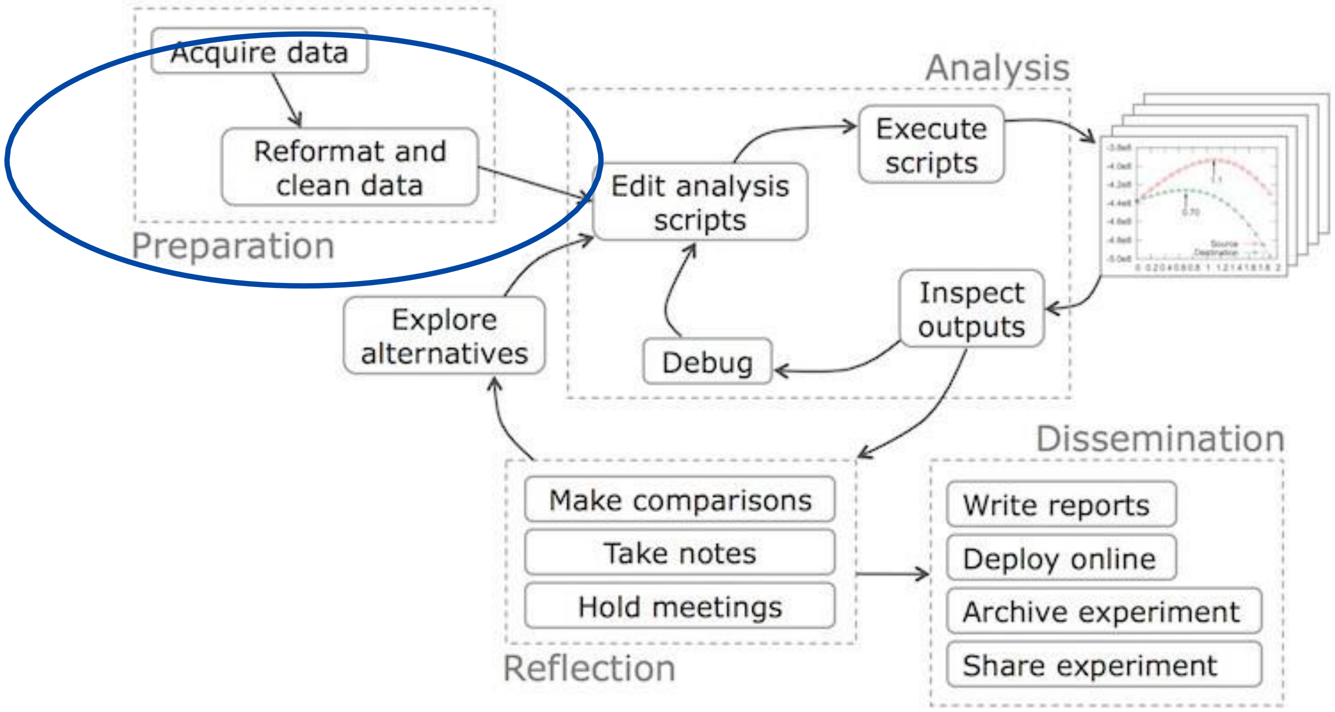
Exercises

– Python / Jupyter to load data
– psycopg2
– PostgreSQL to store data

TODO in W4

- Grok Python modules 10-12 (files, more data structures, complex data structures)
- Grok SQL modules 4 -5
- Summarise and prepare data

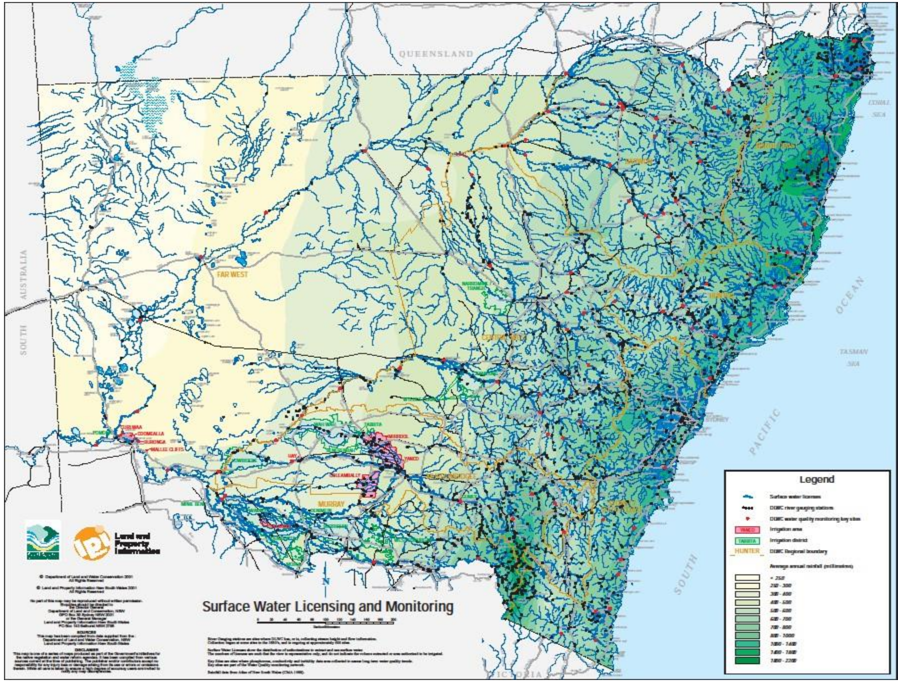
Exploratory Analysis Workflow



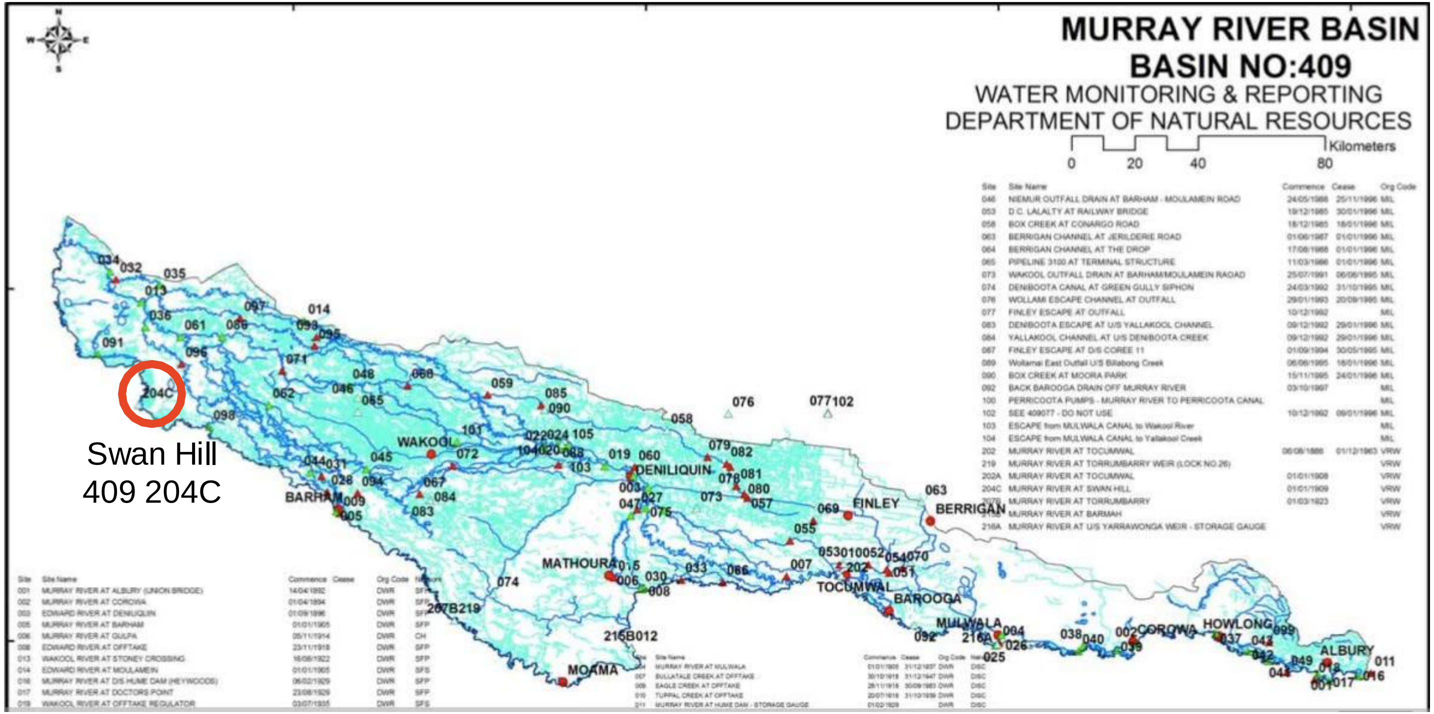
New Scenario

New Data Set

- Water measurements:
  - automatic monitoring stations that are distributed over a larger area
  - Periodically send their measured values to a central authority
  - Time-series data of:
    - water level
    - water flow
    - water temperature
    - salinity (via measuring electric conductivity) or other hydraulic properties



Example: Murray River Basin in NSW



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

Where do we get data from?

- You or your organization might have it already, or a colleagues provides you access to data.
  - Typical exchange formats: CSV, Excel, XML/JSON
- Or: Download from an online data server
  - Still typically in CSV or Excel etc, but now problems with meta-data
- Or: Scrap the web yourself or use APIs of resources
  - Cf. textbook, chapter 9

- Our data set comes from a colleague in Excel format

Water dataset

Contains four CSV data files:

- Measurements.csv
- Organisations.csv
- Sensors.csv
- Stations.csv

- Lets have a look

Relational Databases

- Today's goal is to store the data in a relational database

- **Relational data model** is the most widely used model today
  - Main concept: **relation**, basically a table with rows and columns
  - Every relation has a **schema**, which describes the columns, or fields

- This sounds like a spreadsheet, but as we will see, it has some differences

Definition of Relation

- **Informal Definition:**
  A ***relation*** is a named, two-dimensional table of data
  - Table consists of rows (record) and columns (attribute or field)
- Example:

Attributes (also: columns, fields)

Student				
<u>sid</u>	name	login	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

Tuples (rows, records)

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

Database Loading with Python

Accessing PostgreSQL from Python: psycopg2

– First, we need to import the **psycopg2** module, then connect to Postgresql
– Note: You need obviously to provide your own login name

```
import psycopg2

def pgconnect():
    # please replace with your own details
    YOUR_DBNAME = ' '
    YOUR_USERNAME = ' '
    YOUR_PW     = ' '
    try:
        conn = psycopg2.connect(host='localhost',
                                database=YOUR_DBNAME,
                                user=YOUR_USERNAME,
                                password=YOUR_PW)
        print('connected')
    except Exception as e:
        print("unable to connect to the database")
        print(e)
    return conn
```

Accessing PostgreSQL from Python: psycopg2 (cont'd)

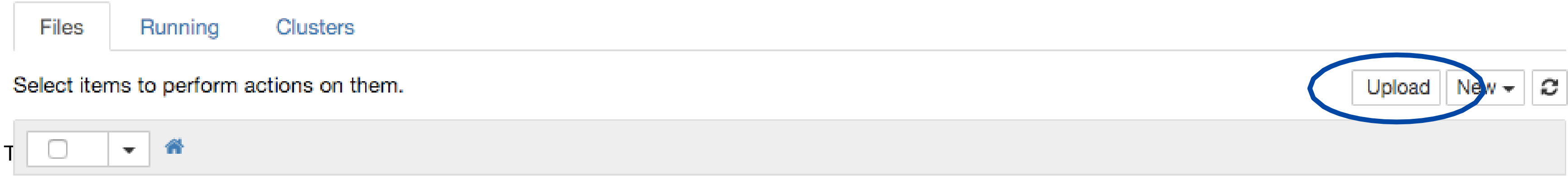
- How to execute an SQL statement on an open connection 'conn'
  - we prepared a helper function which encapsulates all the error handling:

```
def pgexec( conn, sqlcmd, args, msg ):
    """ utility function to execute some SQL statement
        can take optional arguments to fill in (dictionary)
        error and transaction handling built-in """
    retval = False
    with conn:
        with conn.cursor() as cur:
            try:
                if args is None:
                    cur.execute(sqlcmd)
                else:
                    cur.execute(sqlcmd, args)
                print("success: " + msg)
                retval = True
            except Exception as e:
                print("db error: ")
                print(e)
    return retval
```

Exercise 1: Data Loading with DB Loader

- Download data and notebook from **Canvas**
  - Four CSV data files:
    - Measurements.csv
    - Organisations.csv
    - Sensors.csv
    - Stations.csv
  - Jupyter Notebook
- Upload all those files to Jupyter server

Important:
Make sure to use the correct names including the '.csv' file extension



Exercise 1: Data Loading with Python

- Next part in Jupyter notebook
  - Load CSV data into Python
  - Helper functions for connecting and querying postgresql
    - important: **Edit your login details** in the pgconnect() function
  - Check content of Organisation table

- **Your task:** Doing the same for the 'Measurements' and 'Stations' data
  - table creation & data loading in Python
- Any other observations?
- What problems do you encounter when trying to load the table?

Accessing PostgreSQL from Python: psycopg2 (cont'd)

– Example: Creating a table and loading some data

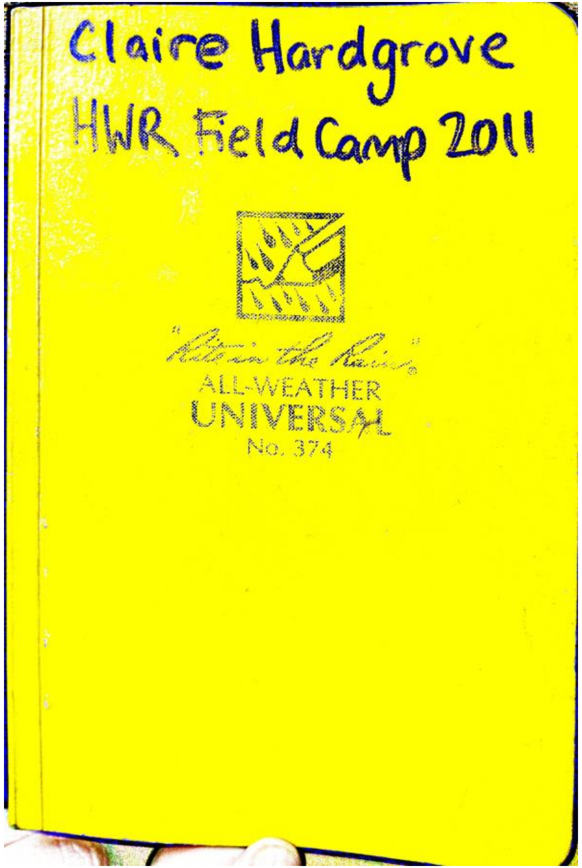
```
data_organisations = list(csv.DictReader(open('water_data/Organisations.csv')))
# 1st: login to database
conn = pgconnect()

# 2nd: ensure that the schema is in place
organisation_schema = """CREATE TABLE IF NOT EXISTS Organisation (
                         code VARCHAR(20) PRIMARY KEY,
                         orgName     VARCHAR(150)
                   )"""
pgexec (conn, organisation_schema, None, "Create Table Organisation")

# 3nd: load data
# IMPORTANT: make sure the header line of CSV is without spaces!
insert_stmt = """INSERT INTO Organisation(code,orgName)
                        VALUES (%(Code)s, %(Organisation)s)"""
for row in data_organisations:
    pgexec (conn, insert_stmt, row, "row inserted")
```

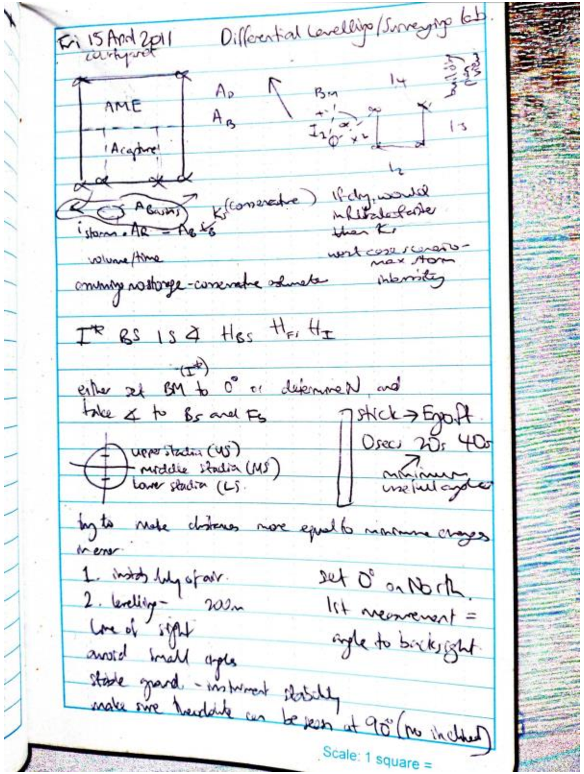
Transforming and Cleaning Data

Technical data cleaning issues

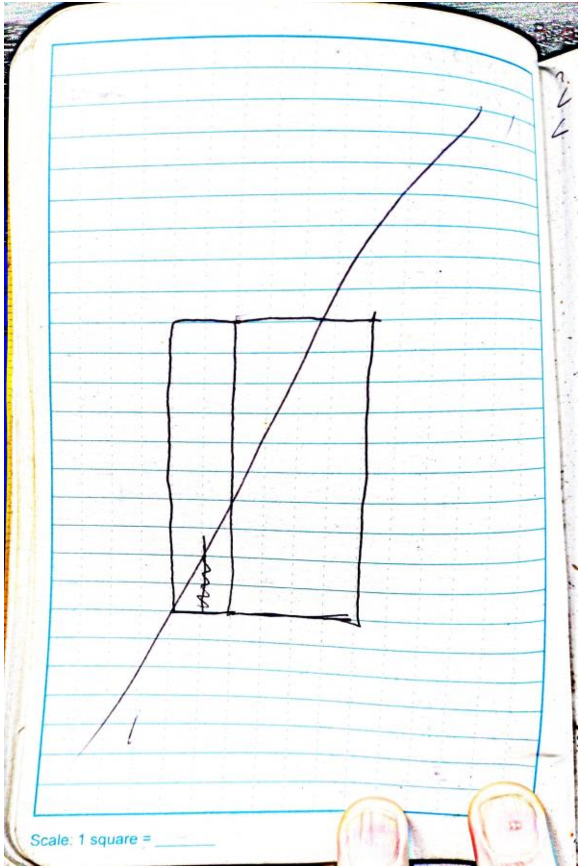
- Interpretation of data format and meta-data
- Differences in naming conventions
  - Excel headers with spaces and quotes, which both are not allowed to DBMS
- Inconsistent or missing data entries
- 'shape' of data

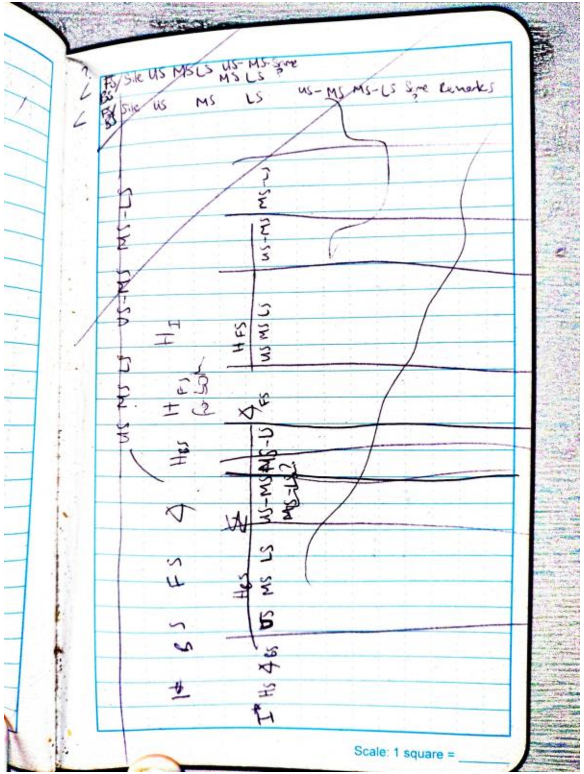
Real data issues: field data



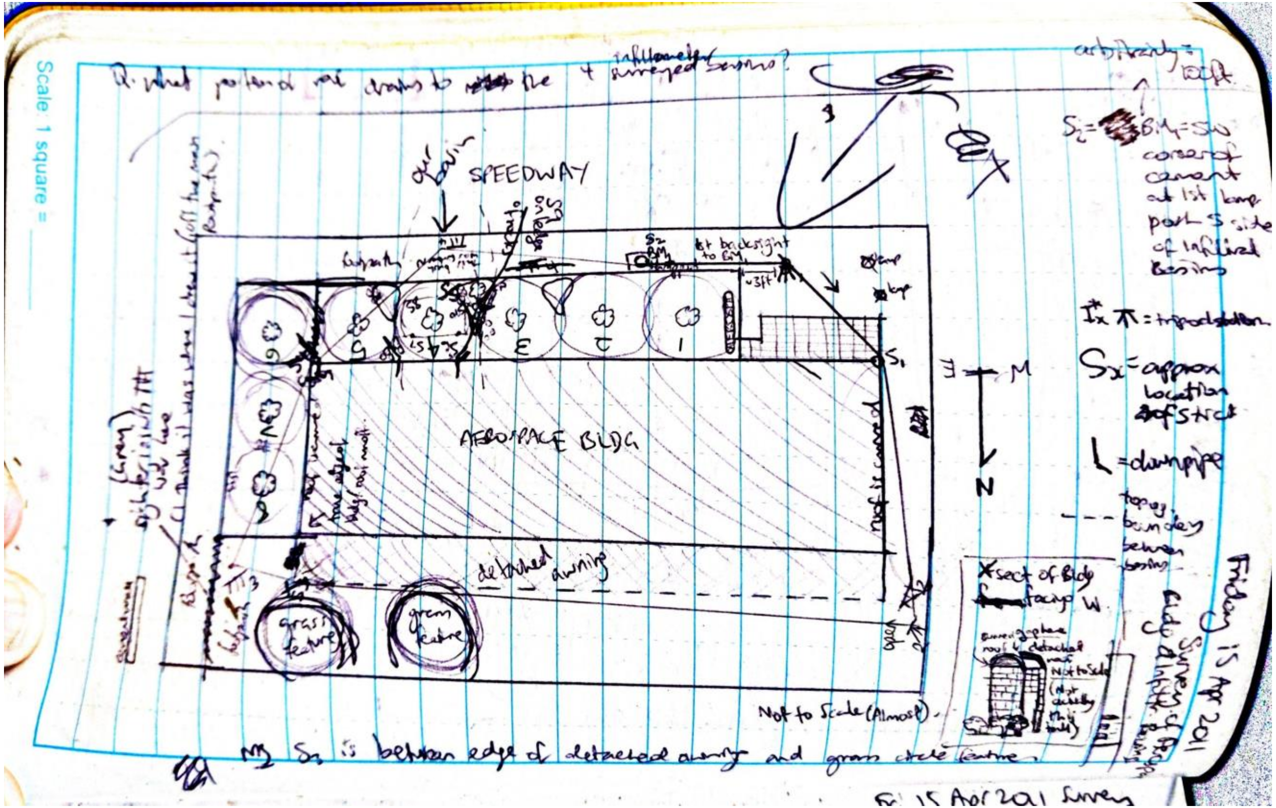


Real data issues: field data

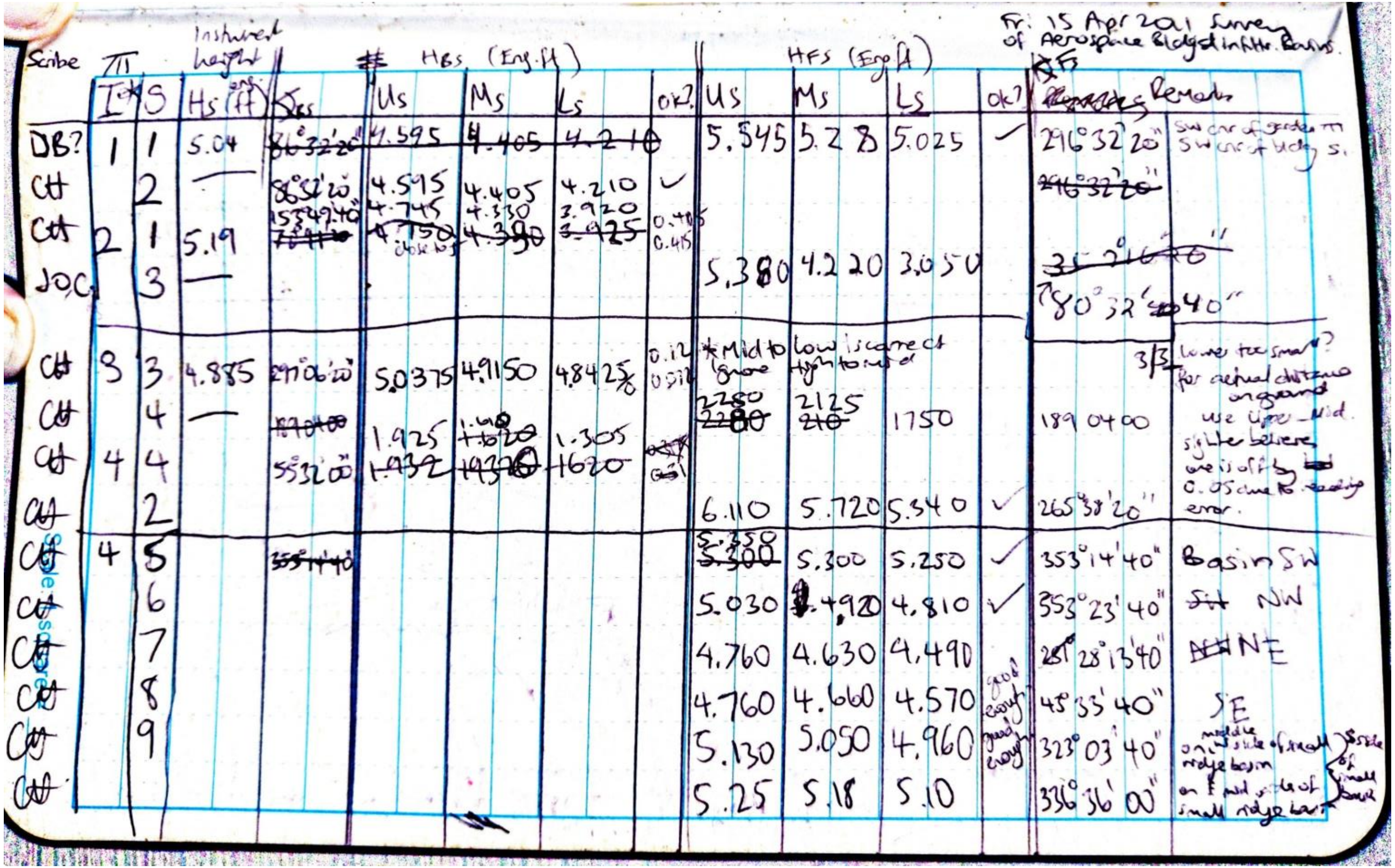




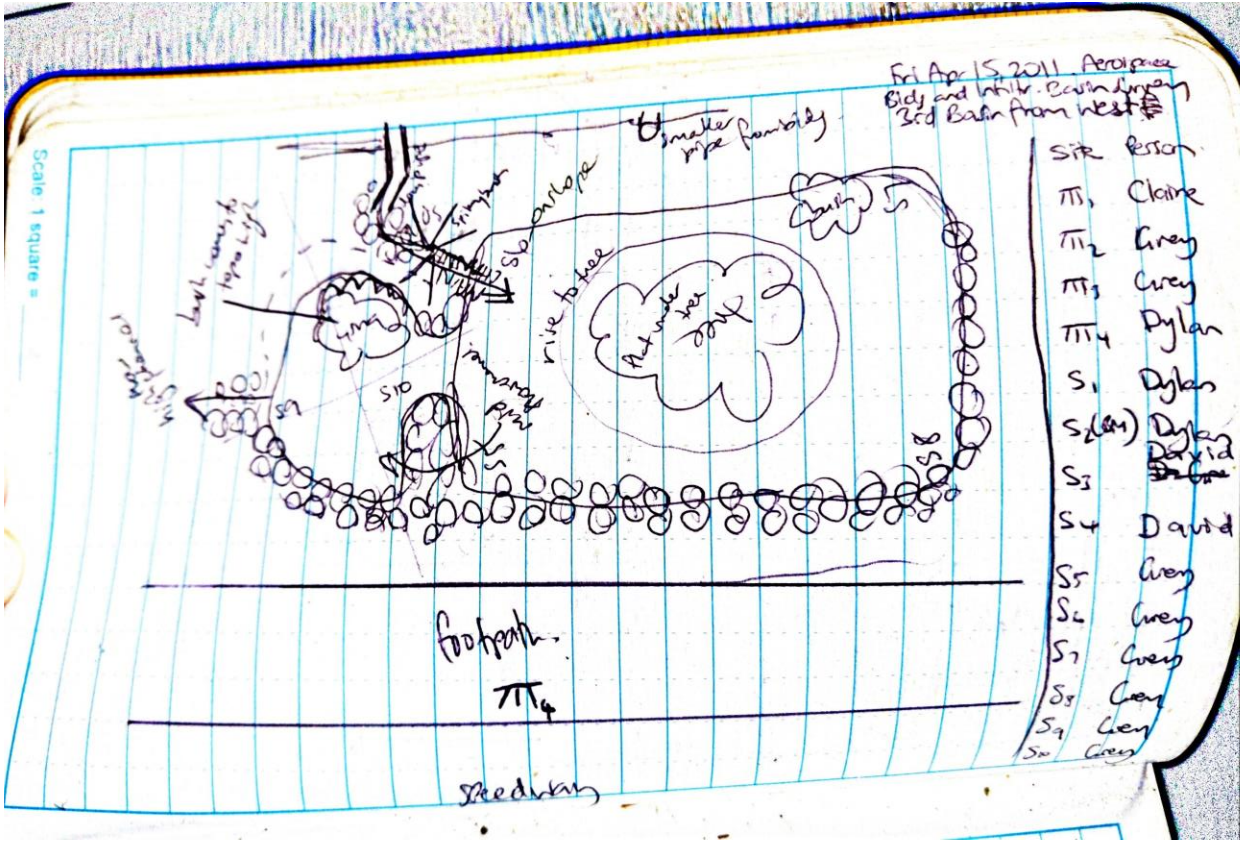
Real data issues: field data



Real data issues: field data



Real data issues: field data



Real data issues: field data

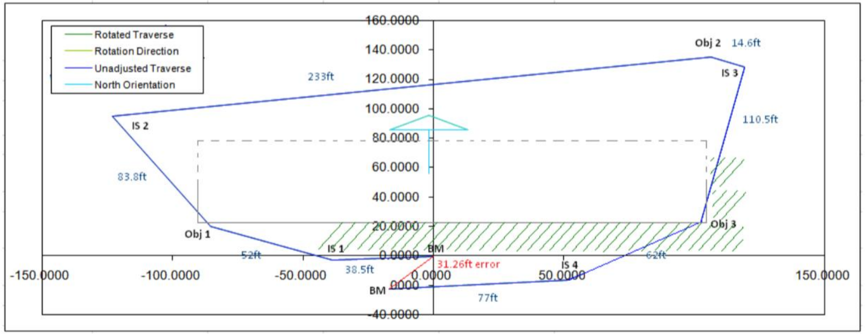


Figure1. Screenshot of the TraverseXL plot of survey locations. North is indicated in light blue, instrument stations are indicated on the inside perimeter by "IS #", objective locations on the outside perimeter by "Obj #", and distances in blue. Green hachures indicate approximate position of detention basins relative to the building, approximately located in grey (see text for limitations). Grid coordinates are left in for traverse reference.

[1] Fredericks, B. and M.G. Wing. 2006. Traverse XL: An EXCEL-based program for entering, displaying, and analyzing spatial measurement data. Surveying and Land Information Science 66(1):65-72. Spreadsheet available from http://www.cof.orst.edu/wingm/#Software: (accessed 22 April 2011)

It is possible to suggest where errors accumulated by considering the traverse plot (Figure 1). In the plot, Objectives (Obj) 1 and 3 represent the south side of the building and are approximately parallel to their true E-W alignment. Objs 2 and 3 represent the east side of the building and form close to a right angle with Obj 1- Obj3. These are probably well located (compared to other points), suggesting a substantial component of error accumulated at the end of the survey between the IS 4 and the benchmark (not IS 4 and Obj3 because this backsight was used in the plot).

[redacted]

David Bernard, Grey Nearing, Joe Calvillo, Claire Hardgrove, Jacob Meuth

In addition, the angle between Instrument Station (IS) 2 and Obj 2 is substantially in error. The northwest corner of the building was missed in the survey, but it is known that IS 2 was located beyond the northern edge of the building (see field sketch in Figure 2) and thus represents a maximum possible northern extent of the building. However, Obj 2 which is known to be located south of IS 2 from field sketches, plots north of IS 2. Moreover, Obj 2 represents the outer edge of an abutment to the building (supporting an external staircase) which does not support the roof area used for catchment calculations. Considering both the error and building shape, the true northern extent of the main roof area is substantially south of the point indicated by the survey. A rectangular outline of the building is projected onto the traverse in Figure 1 with dashed edges showing uncertainty in its northern extent.

It is possible that a combination of measurement and procedural errors were made at IS 2 including errors in setting the north bearing on the theodolite.

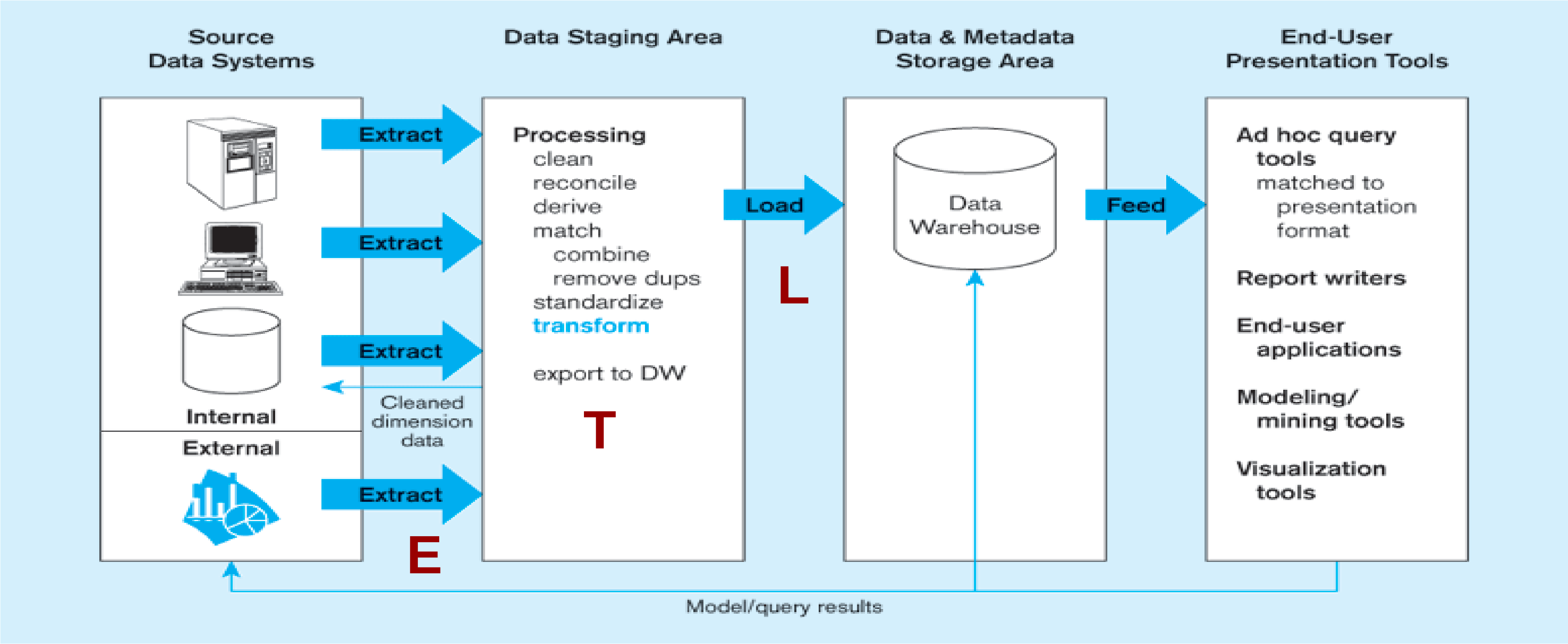
Used with permission from David Bernard, Assistant Professor Grey Nearing, Joe Calvillo, Jacob Meuth

Real data issues: QA considerations

– What is the concentration of a mineral in a rock sample?
– Larger grain size/more heterogeneous sample requires larger sample size to make sure measurements are reflecting bulk properties of the material not the properties of a specific sub sample
– How can you be sure your own sample is representative?
– How can you be sure the lab is measuring the whole sample, not using a smaller subsample for convenience?
– Example of quality controls: Duplicates (check your own sampling)/Triplicates (check the lab methodology). Were there adequate and well documented QA/QC procedures?
– Did each person doing sampling follow the same methodology?
– Did each person classify samples in the same way (relatable example: are teachers applying the same standard for grading each assignment?)
– Need for twin holes

ETL Process

- This problem is well known from data warehousing
- **ETL Process:** Capture/Extract - Data Cleansing - Transform - Load



Data Modeling

THE UNIVERSITY OF SYDNEY

Relation Database Theory and Issues

- The modelling process in relational database known as OLTP (Online Transactional Processing) focuseson normalization process which yields to a flexible model
  - making it easy to maintain dynamic relationships between business entities
- So it is effective and efficient for operational databases – a lot of updates
- However, a fully normalized data model can perform very inefficiently for queries.
- Historical data are usually large with static relationships:
  - Unnecessary joins may take unacceptably long time
- So how to proceed with a database approach?

  => **OLAP: Online Analytical Processing (Data Warehousing Approach)**

What is a Data Warehouse?

- Subject-oriented
  - Organized by subject, not by application
  - Used for analysis, data mining, etc.
- Integrated
  - Constructed by integrating multiple, heterogeneous data sources
    - relational databases, flat files, on-line transaction record
- Time Variant
  - Large volume of historical data (Gb, Tb)
  - Time attributes are important
- Non-volatile
  - Updates infrequent or does not occur
  - May be append-only

Conceptual Modeling of Data Warehouses

- Modeling data warehouses: dimensions & measures instead of relational model
- Data warehouse contains a **large** central table (**fact table**)
  - Contains the data without redundancy
- A set of **dimension tables**

Data Warehouses: Fact Tables

- Relational 'data warehouse' applications are centered around a ***fact table***
  - For example, a supermarket application might be based on a table

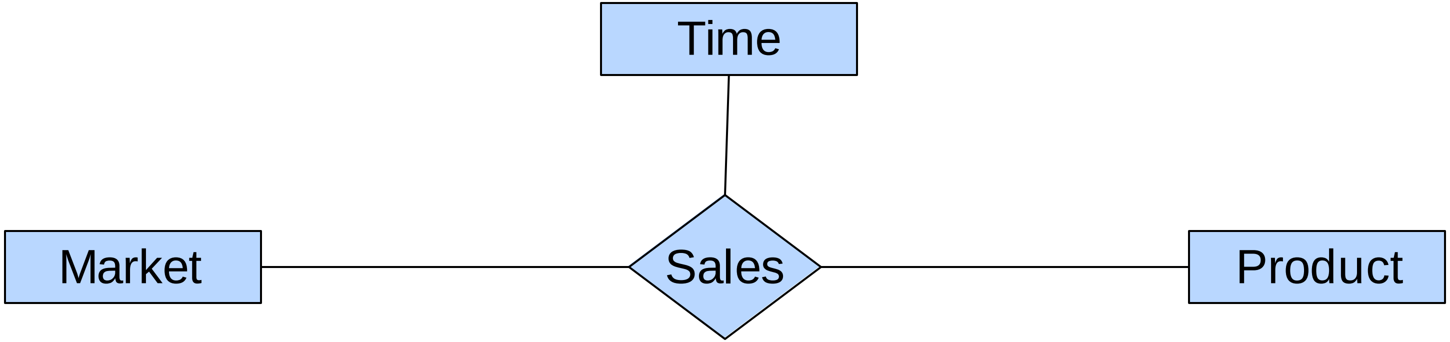
    Sales (*Market_Id*, *Product_Id*, *Time_Id*, *Sales_Amt*)

market_id	product_id	time_id	sales_amt
M1	P1	T1	3000
M1	P2	T1	1000
M1	P3	T1	500
M2	P1	T1	100
M2	P2	T1	1100
M2	P3	…	…
…	…		

- The table can be viewed as *multidimensional*
  - Collection of numeric <u>measures</u>, which depend on a set of <u>dimensions</u>
    - E.g. *Market_Id*, *Product_Id*, *Time_Id* are the dimensions that represent specific supermarkets, products, and time intervals
    - *Sales_Amt* is a function of the other three

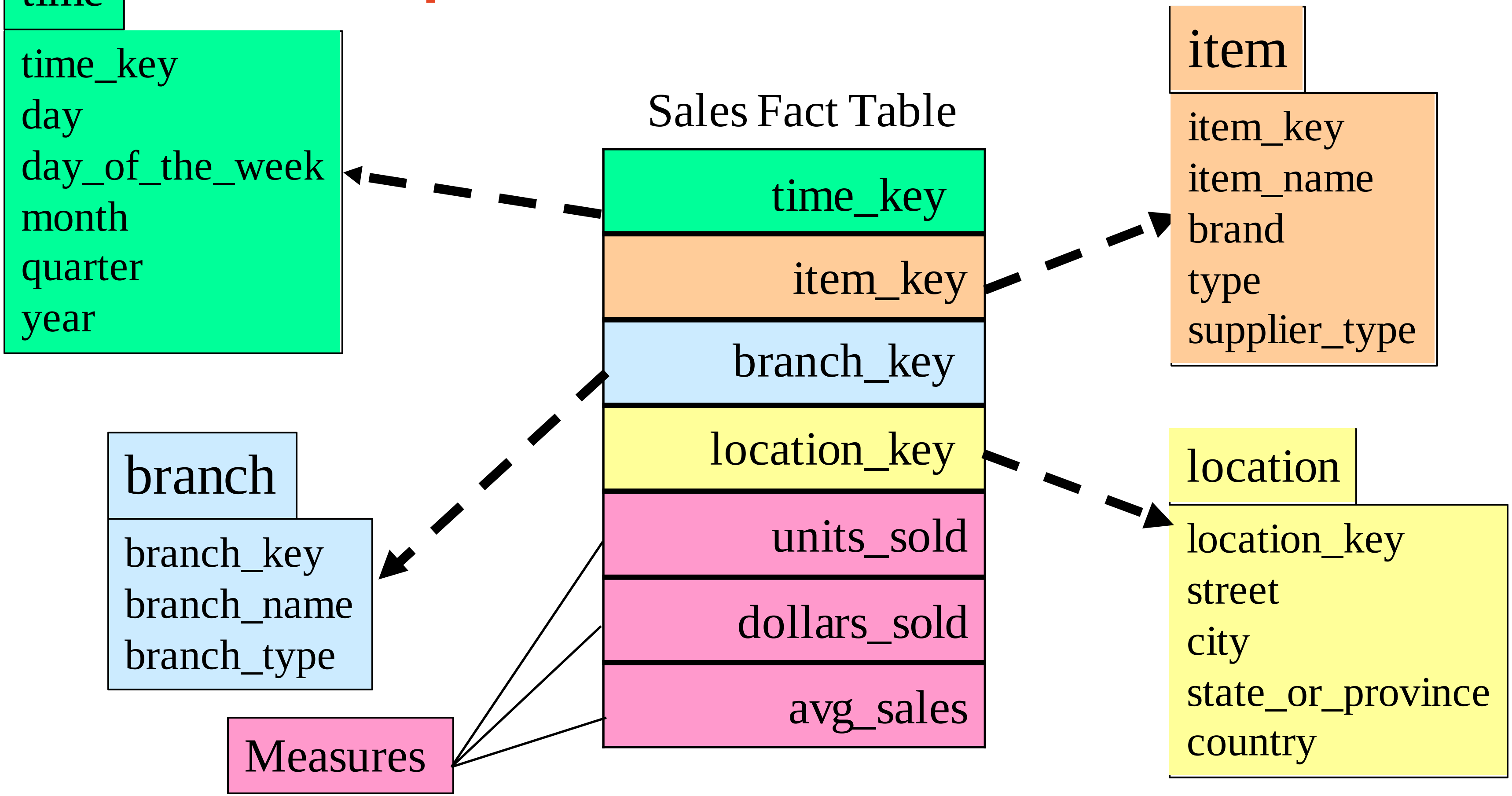
Data Warehousing: Star Schema

– The fact and dimension relations linked to it looks like a star;
– this is called a ***star schema***
– *Most common modeling paradigm*



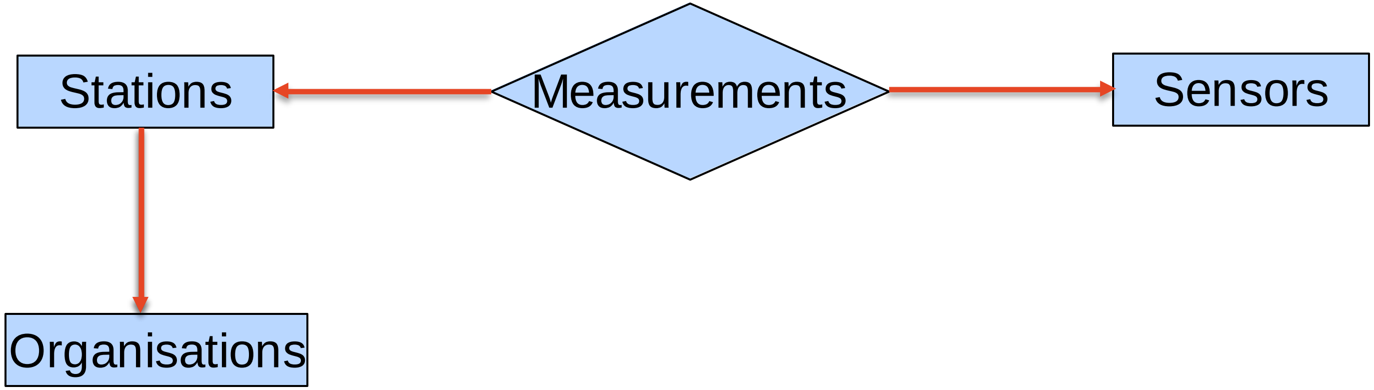
– If we map this to relations
  – 1 central fact table
  – *n* dimension tables with foreign key relationships from the fact table
  (the fact table holds the FKs referencing the dimension tables)

Example of Star Schema

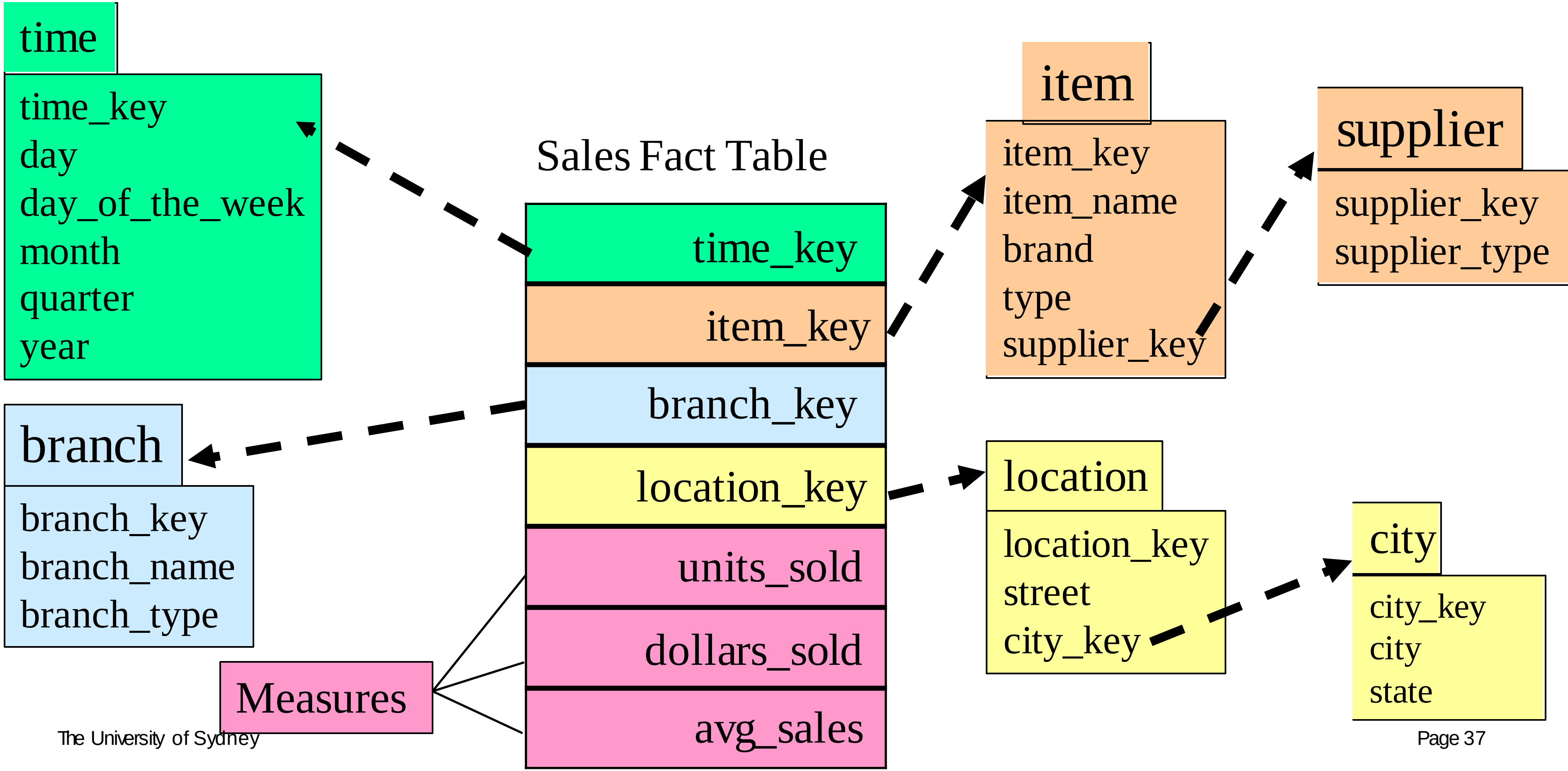


Data Warehousing: Snowflake Schema

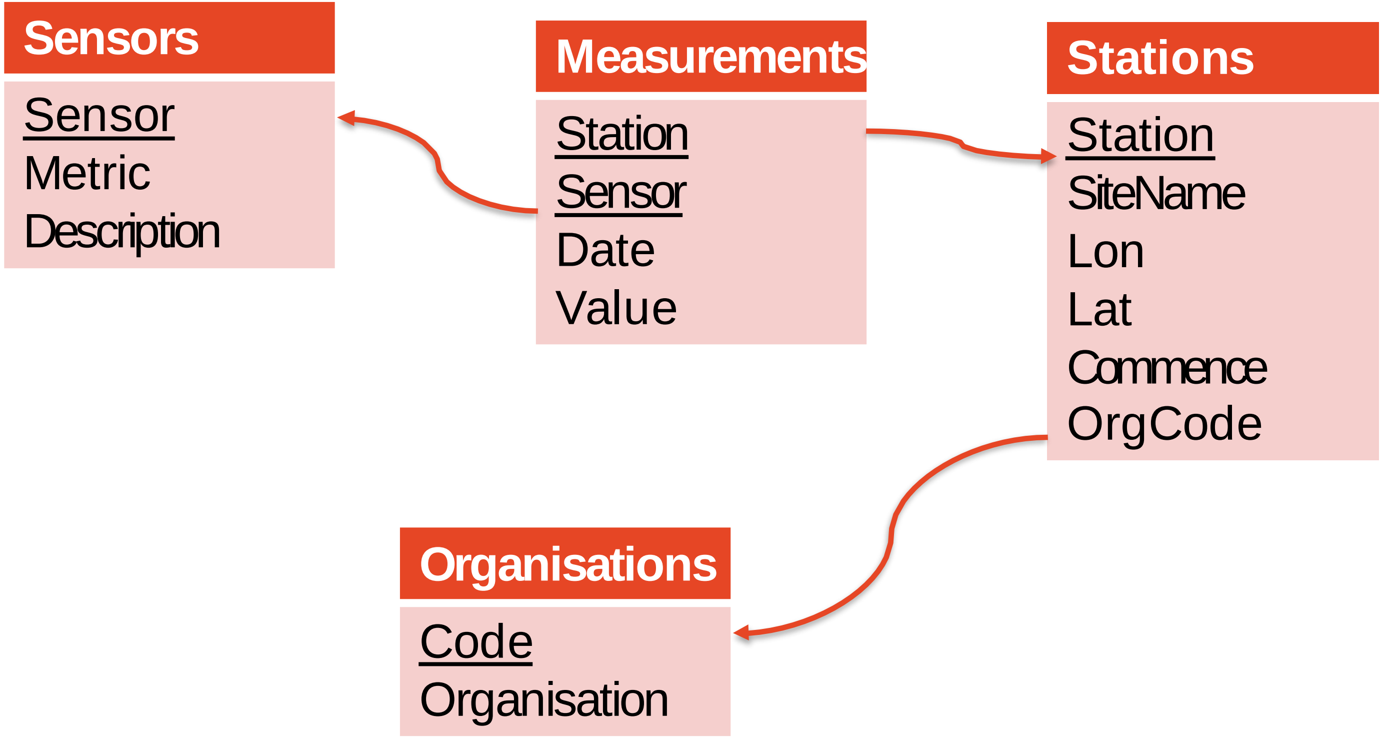
- Snowflake schema: A refinement of star schema where some dimensional hierarchy is **normalized** into a set of smaller dimension tables, forming a shape similar to snowflake
- measurements are the facts, rest describes the dimensions



Example of Snowflake Schema



Modeling our Water Data Set



Data Warehousing: Fact constellations

– Fact constellations: **Multiple** fact tables share dimension tables, viewed as a collection of stars, therefore called galaxy schema or fact constellation

Questions to ask

– Useful to have a theoretical framework
BUT:
– Do you need a relational database?
– How much data do you have and what are you trying to do with it?
– Talk to your data engineers, database administrators.
– NoSQL options: document databases, key-value databases, wide-column stores, graph databases

Database conferences:
https://vldb.org/2022/ – in Sydney in September.
https://sigmod.org

DB Creation

SQL – The Structured Query Language

- SQL is the standard declarative query language for RDBMS
- Supported commands from roughly two categories:
  - **DDL** (Data Definition Language)
    - Create, drop, or alter the relation schema
    - Example:

      CREATE TABLE *name* (*list_of_columns*)
  - **DML** (Data Manipulation Language)
    - for retrieval of information also called **query language**
    - **INSERT**, **DELETE**, **UPDATE**
    - **SELECT … FROM … WHERE**

Table Constraints and Relational Keys

- When creating a table, we can also specify *Integrity Constraints* for columns
  - eg. domain types per attribute, or **NULL / NOT NULL** constraints

- **Primary key:** <u>unique</u>, <u>minimal</u> identifier of a relation.
  - Examples include employee numbers, social security numbers, etc. This is how we can guarantee that all rows are unique.

- **Foreign keys** are identifiers that enable a <u>dependent relation</u> (on the many side of a relationship) to refer to its <u>parent relation</u> (on the one side of the relationship)
  - Must refer to a candidate key of the parent relation
  - Like a \`logical pointer'

- Keys can be **simple** (single attribute) or **composite** (multiple attributes)

Example: Relational Keys

Primary key identifies each tuple of a relation.

Composite Primary Key consisting of more than one attribute.

Student

<u>sid</u>	name
31013	John

Enroll

<u>sid</u>	<u>ucode</u>	grade
31013	I2120	CR

Units_of_study

<u>ucode</u>	title	credit_pts
I2120	DB Intro	4

Foreign key is a (set of) attribute(s) in one relation that `refers' to a tuple in another relation (like a `logical pointer').

SQL Domain Constraints

- SQL supports various domain constraints to restrict attribute to valid domains
  - NULL / NOT NULL — whether an attribute is allowed to become *NULL* (unknown)
  - DEFAULT — to specify a default value
  - CHECK(*condition*) — a Boolean *condition* that must hold for every tuple in the db instance

Example:

```
CREATE TABLE Student
 (
     sid         INTEGER        PRIMARY KEY,
     name        VARCHAR(20)    NOT NULL,
     gender      CHAR           CHECK (gender IN ('M,'F','T')),
     birthday    DATE           NULL,
     country     VARCHAR(20),
     level       INTEGER        DEFAULT 1 CHECK (level BETWEEN 1 and 5)
 );
```

Exercise 3: Schema Creation

- Next part in Jupyter notebook
  - We provided an example schema already
    - follows the mapping rules from the previous slides

- **Your Task:** Using Python + SQL, create the full SQL schema for the given data model
    - This should give you seven separate tables as compared to the five spreadsheets which we originally had

Data Loading / Storage

Data Storing

- Where are we now?
  - We have analysed our given data set
  - Cleaned it
  - Transformed it and created a corresponding relational database
- Next, we want to store the given data in our database.

SQL DML Statements

- Insertion of new data into a table / relation
  - **Syntax:**
    INSERT INTO *table* [“(”*list-of-columns*“)”] VALUES “(“ list-of-*expression*“)”
  - Example:
    INSERT INTO Students (sid, name) VALUES (53688, ‘Smith’)
- Updating of tuples in a table / relation
  - **Syntax:**
    UPDATE *table* SET *column*“=“*expression*{“,”*column*“=“*expression*}
    [WHERE *search_condition*]
  - Example:
    ```
    UPDATE students
        SET gpa = gpa - 0.1
      WHERE gpa >= 3.3
    ```
- Deleting of tuples from a table / relation
  - **Syntax:**
    DELETE FROM *table* [WHERE *search_condition*]
  - Example:
    DELETE FROM Students WHERE name = ‘Smith’

(Final) Exercise 4: Data Storage

- Next part in Jupyter notebook
  - Make sure you have the full SQL schema for the given data model
  - Load all CSV files into these tables

Review

Reprise Participation Marking

Requirements

- Submit code at end of each week
- **Jupyter Notebooks:**
  - The various exercises have placeholder cells marked as TODO:

```
# TODO: replace the content of this cell
raise NotImplementedError
```

  - The content of these cells needs to be replaced with your own solution => basis for participation marking

Output

- Code/spreadsheets from exercises

Marking

- 10% of overall mark
- each week's participation assessed as: all done, partially done, no participation

Next Time

Next Lecture : Querying and Summarising Data

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
– SQL querying
– Summarising data with SQL
– Statistic functions support in SQL

Readings

– Data Science from Scratch, Ch 23

TODO in W5

– Finish Grok Python modules
– Finish Grok SQL modules
– project data

Questions?