COMP5310: Principles of Data Science

W4: Data Transformation and Storage with Python and SQL

Presented by

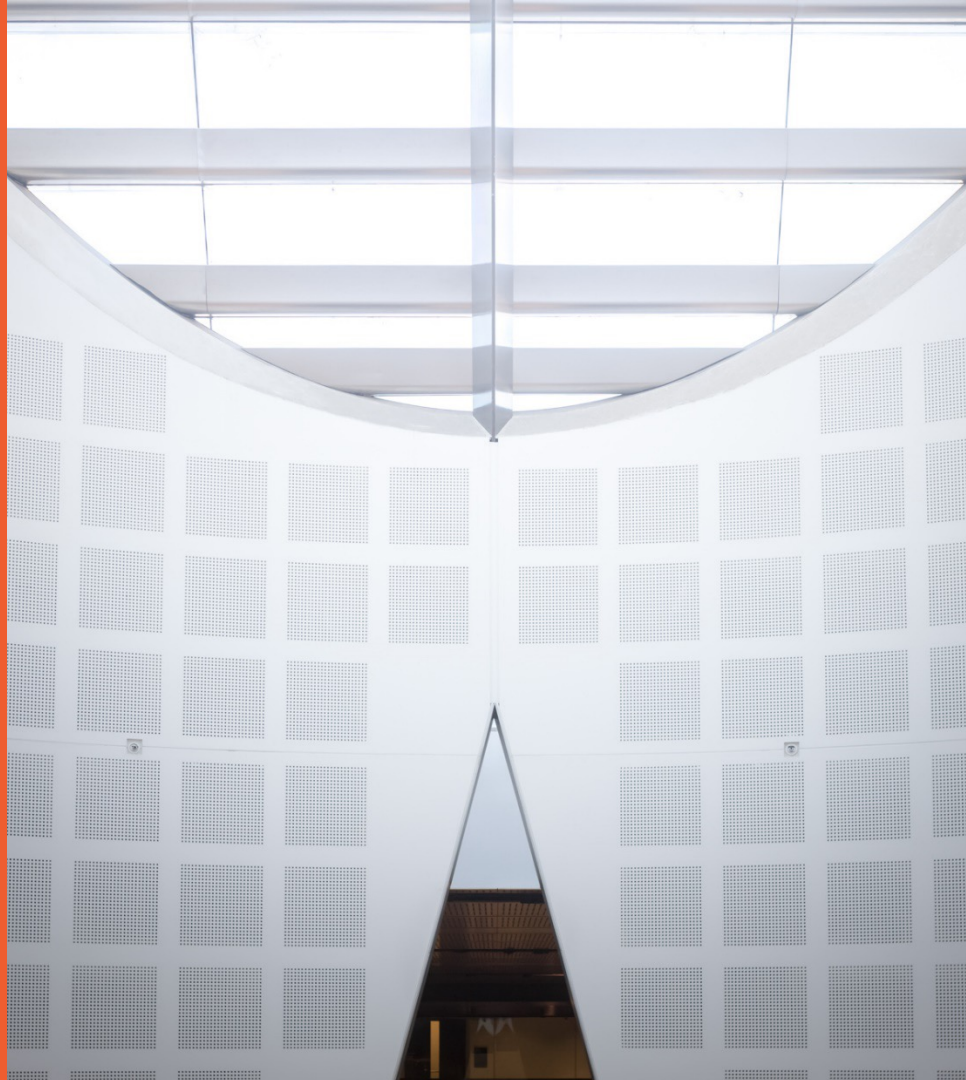
Maryam Khanian

School of Computer Science

Based on slides by previous lecturers of this unit of study

THE UNIVERSITY OF SYDNEY

The University of Sydney



Last week: Data Exploration with Python

Objective

- Learn Python tools for exploring a new data set programmatically.

Lecture

- Pandas
- Descriptive statistics, e.g., median, quartiles, IQR, outliers.
- Descriptive visualisation, e.g., boxplots.

Readings

- Data Science from Scratch: Ch 5

Exercises

- matplotlib: Visualisation.
- pandas: Descriptive stats.

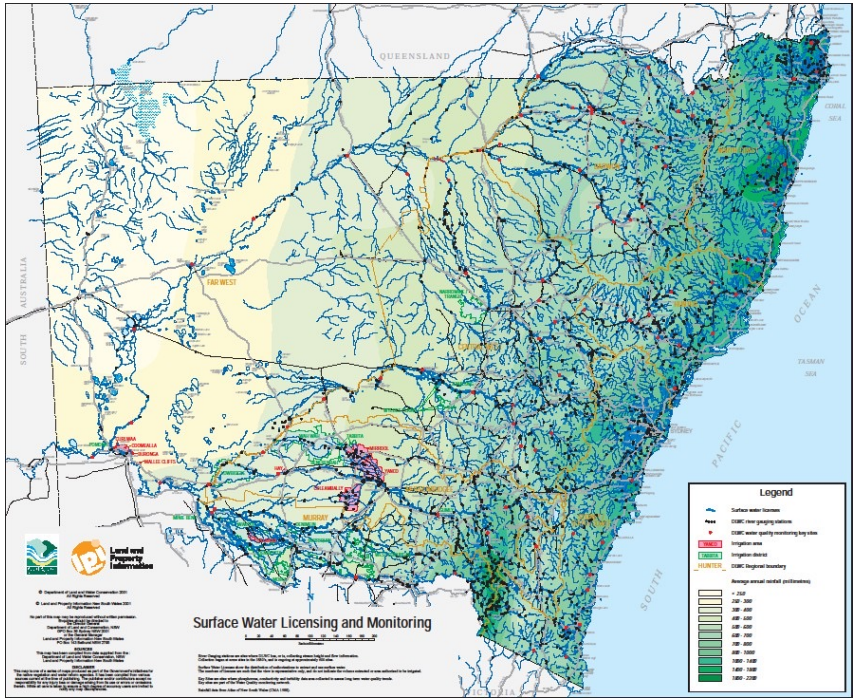
TO-DO in W3

- Ed Lessons Python modules 7-9.
- Ed Lessons SQL modules 18-19.
- Load project data with Python. Clean and prepare data

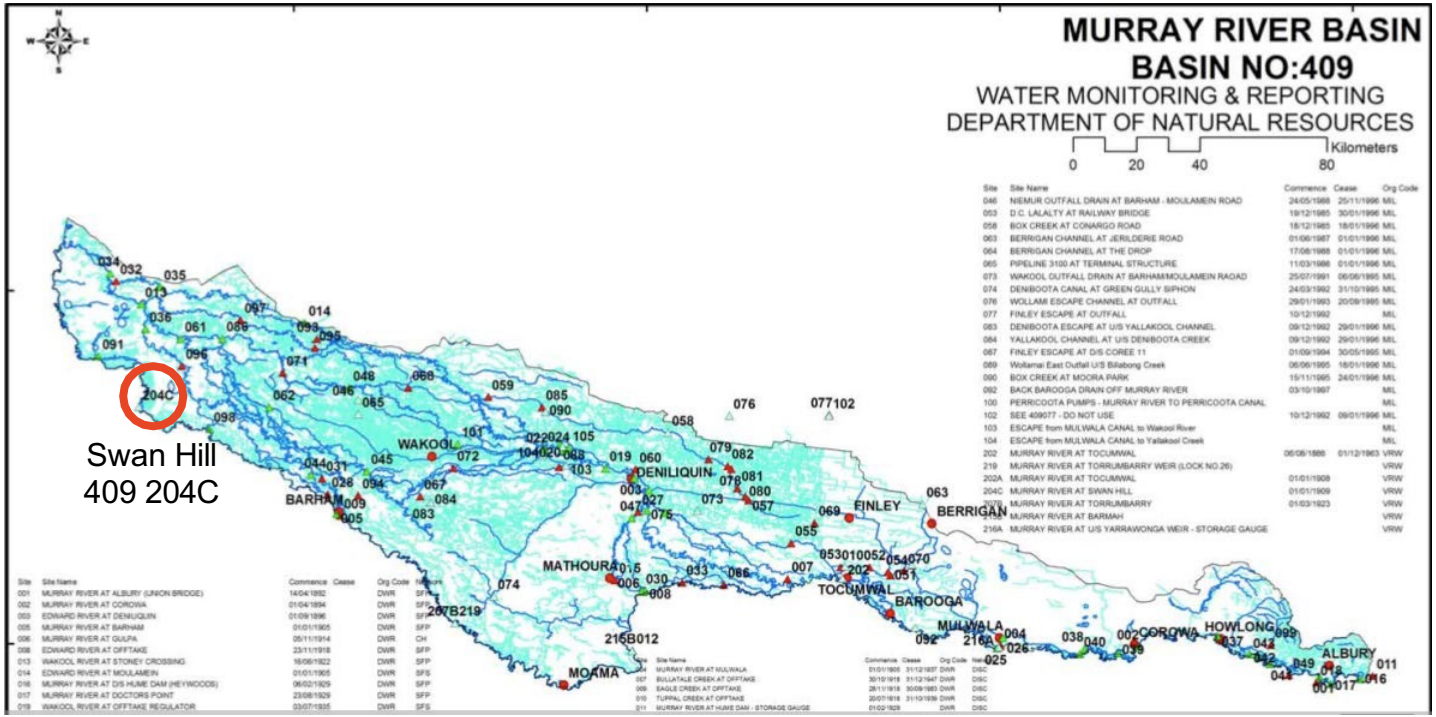
NEW SCENARIO

New data set

- Water measurements:
  - Automatic monitoring stations that are distributed over a large 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.
- Download from an online data server
  - Still typically in CSV or Excel, etc., but now problems with meta-data.
- Scrap the web yourself or use APIs of resources.
- Cf. Data Science from Scratch, 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

- Let's have a look.

RELATIONAL DATABASES

Relational databases

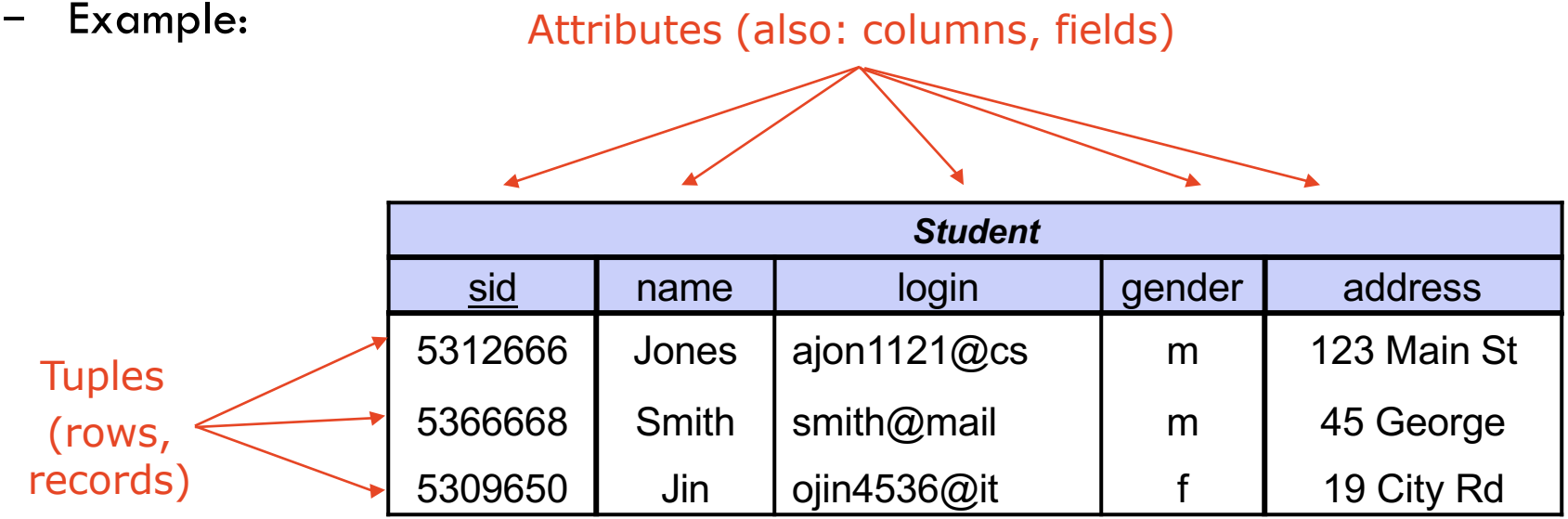
- Today's goal is to store the data in a relational database.

- The relational data model is the most widely used model today.
  - In the **relational model,** a database is a collection of one or more **relations.**
  - Each relation is a table with rows and columns.
  - Each 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:



Student

sid	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

Relation Schema vs Relation Instance

- Formally, a relation R consists of a relation schema and a relation instance
- A **relation schema** specifies name of relation, and name and data type (domain) of each attribute.
  - $R = (A_1, A_2, \ldots, A_n)$ is a ***relation schema***
    - $A_1, A_2, \ldots, A_n$ are ***attributes***, *each having a* ***data type***
  - e.g. `Student(sid: string, name: string, login: string, addr: string, gender: char)`
- A **relation instance** is a **set** of tuples (*a table*) for a schema
  - Each tuple has the same number of fields as attributes defined in schema
  - Values of a field in a tuple must conform to the data type defined in schema
  - Relation instance often abbreviated as just relation

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.
- RDBMS **table** extends mathematical **relation**
  - RDBMS allows duplicate rows, supports an order of tuples or attributes, and allows null 'values' for unknown information

DB CREATION

SQL – The Structured Query Language

- SQL is the standard declarative language for interacting with 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`

Creating Tables in SQL

- Creation of tables/relations:

```
CREATE TABLE name ( list-of-columns );
```

  - Example: Create the Student table.

```
CREATE TABLE Student (sid INTEGER,
                      name VARCHAR(20),
                     login VARCHAR(20),
                    gender CHAR,
                   address VARCHAR(50) );
```

  - This actually specifies the schema information
  - Note that the type of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

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
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"`

Integrity constraints

- When creating a table, we can specify *integrity constraints* for columns.
  - A variety of rules to maintain the integrity of data when it is manipulated
  - The rule must be satisfied for *any* instance of the database; e.g., *domain constraints.*
- Integrity constraints are declared in the schema
  - They are specified when the schema is defined.
  - Declared integrity constraints are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified integrity constraints.
  - If integrity constraints are declared, DBMS will not allow illegal instances.
  - Stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Domain constraints

- *Domain constraints* restrict attributes 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)
 );
```

level ≥ 1 and level ≤ 5

1 ≤ level ≤ 5

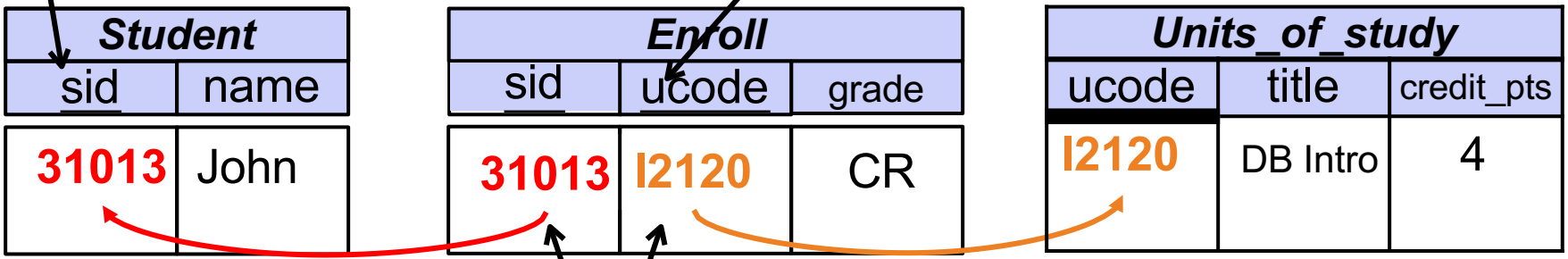
Relational keys

- **Key:** **unique, minimal** identifier of a relation.
  - Examples include employee numbers, social security numbers, etc. This is how we can guarantee that all rows are unique.
  - Keys can be **simple** (single attribute) or **composite** (multiple attributes).
- If there's at least one key for a relation, we call each of them a ***candidate key***, and one of the keys is chosen (by DBA) to be the ***primary key (PK)***
  - If we just say **key**, we typically mean ***candidate key***
- **Foreign keys:** identifiers that enable a **dependent relation** to refer to its **parent relation**
  - Must refer to a candidate key of the parent relation.
  - Like a "logical pointer".

Example: Relational keys

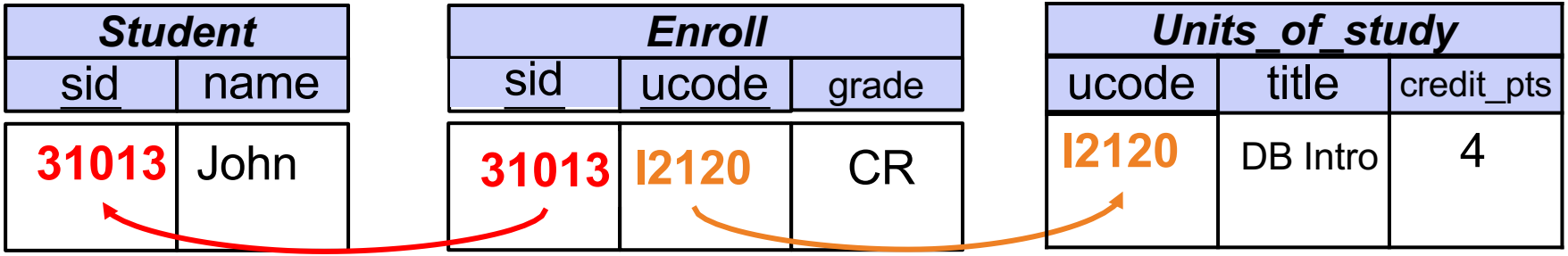
Primary key identifies each tuple of a relation.

Composite Primary Key consisting of more than one attribute.



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

Quiz: candidate key



For the student table:

Can sid be a candidate key? Y

Can sid, name be a candidate key? N

For the Enroll table:

Can sid be a candidate key? N

Can sid, ucode be a candidate key? Y

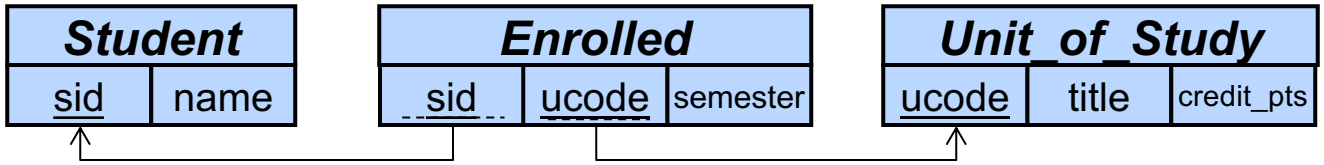
Can sid, ucode, semester be a candidate key? N

~~semester~~ grade

Key & Foreign Key in SQL

- Primary keys and foreign keys can be specified as part of the SQL **CREATE TABLE** statement:
  - The **PRIMARY KEY** clause lists attributes that comprise the primary key.
  - The **UNIQUE** clause lists attributes that comprise a candidate key.
  - The **FOREIGN KEY** clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key.
- Note that, SQL does not require every table to have a key
- By default, foreign key references the primary key of the referenced table
  - **FOREIGN KEY** (sid) **REFERENCES** Student
- Reference columns in the referenced table can be explicitly specified
  - but must be declared as primary or candidate keys
  - **FOREIGN KEY** (lecturer) **REFERENCES** Lecturer(empid)
- Tip: Name them using **CONSTRAINT** clauses
  - **CONSTRAINT** Student_PK **PRIMARY KEY** (sid)

Example: Primary & Foreign Keys




```
CREATE TABLE Student ( sid INTEGER, … ,
    CONSTRAINT Student_PK PRIMARY KEY (sid)
);
CREATE TABLE UoS  ( ucode CHAR(8), … ,
    CONSTRAINT UoS_PK PRIMARY KEY (ucode)
);
CREATE TABLE Enrolled ( sid INTEGER, ucode CHAR(8), semester VARCHAR,
    CONSTRAINT Enrolled_FK1 FOREIGN KEY (sid) REFERENCES Student,
    CONSTRAINT Enrolled_FK2 FOREIGN KEY (ucode) REFERENCES UoS,
    CONSTRAINT Enrolled_PK  PRIMARY KEY (sid,ucode)
);
```

Key constraint

- **Key Constraint:**
  No two distinct tuples can have the same values in all key attributes
- Careful: If used carelessly, this can prevent the storage of database instances that arise in practice!
- Example:

```
CREATE TABLE Enrolled  (
  sid    INTEGER,
  cid    CHAR(8),
  grade CHAR(2),
  PRIMARY KEY (sid,cid) );
```

"For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled  (
  sid    INTEGER,
  cid    CHAR(8),
  grade CHAR(2),
  PRIMARY KEY (sid)
  UNIQUE (cid, grade) );
```

"Students can take only one course and receive a single grade for that course; further, no two students in a course receive the same grade."

Foreign key constraint

- **Foreign Key Constraint (Referential Integrity):**
  For each tuple in the referring relation whose foreign key value is α, there must be a tuple in the referred relation with a candidate key that also has value α
  - e.g. Enrolled(*sid*: integer, ucode: string, semester: string)
    sid is a foreign key referring to Student:

sid	ucode	semester
1234	COMP5138	2012S1
3456	COMP5138	2012S1
5678	COMP5138	2012S2
5678	COMP5338	2013S1

Q: What can we say about the Student relation?

Keys and NULLs

- PRIMARY KEY
  - Up to one per table, and must be unique
  - Automatically disallow NULL values

- UNIQUE (candidate key)
  - Possibly many *candidate keys* (specified using UNIQUE)

- FOREIGN KEY
  - By default, allows NULL values
  - If there must be a parent tuple, then must combine with NOT NULL constraint

DATABASE LOADING WITH PYTHON

Accessing PostgreSQL from Python: psycopg2

- First, we need to import the psycopg2 module, then connect to Postgresql.
- ***Note:*** You obviously need 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

- 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: Data loading with Python

- 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?

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 in DBMS.
- Inconsistent or missing data entries.
- “Shape” of data.

Accessing PostgreSQL from Python: psycopg2

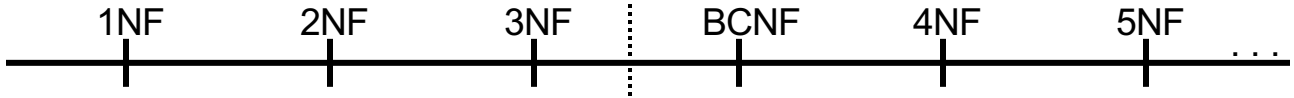
- Example: Creating a table and loading some data.

```
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")
```

DATA MODELING

Relational database theory

- ***Redundant*** data may cause anomalies when manipulating data
- Relational database theory identifies several normal forms
  - Each normal form is characterized by a set of restrictions
  - A relation needs to be decomposed if it does not satisfy the restrictions



Student

sid	name
31013	John

Enroll

sid	ucode	grade
31013	I2120	CR

Units_of_study

ucode	title	credit_pts
I2120	DB Intro	4

Relational database theory issues

- The modelling process in relational database known as OLTP (Online Transactional Processing) focuses on normalization process which yields a flexible model.
  - Making it easy to maintain dynamic relationships between business entities.
- 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
- 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

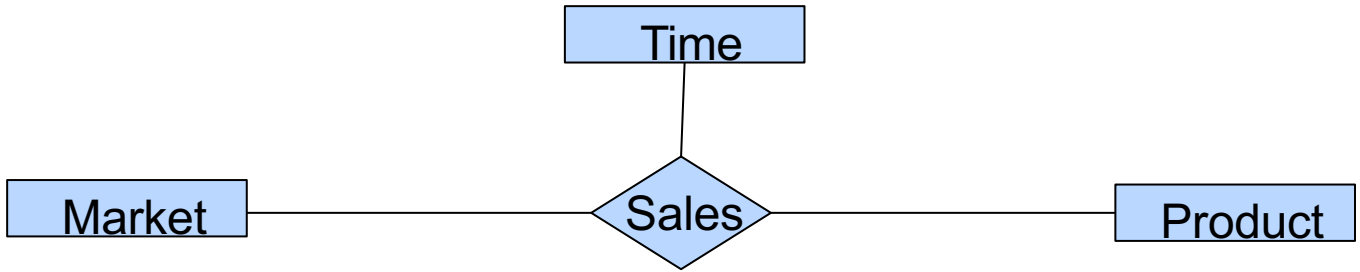
- Relational "data warehouse" applications are centered around a fact table.
  - For example, a supermarket application might be based on a table Sales (market_id, product_id, time_id, sales_amt).

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 measures, which depend on a set of dimensions.
  - 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.

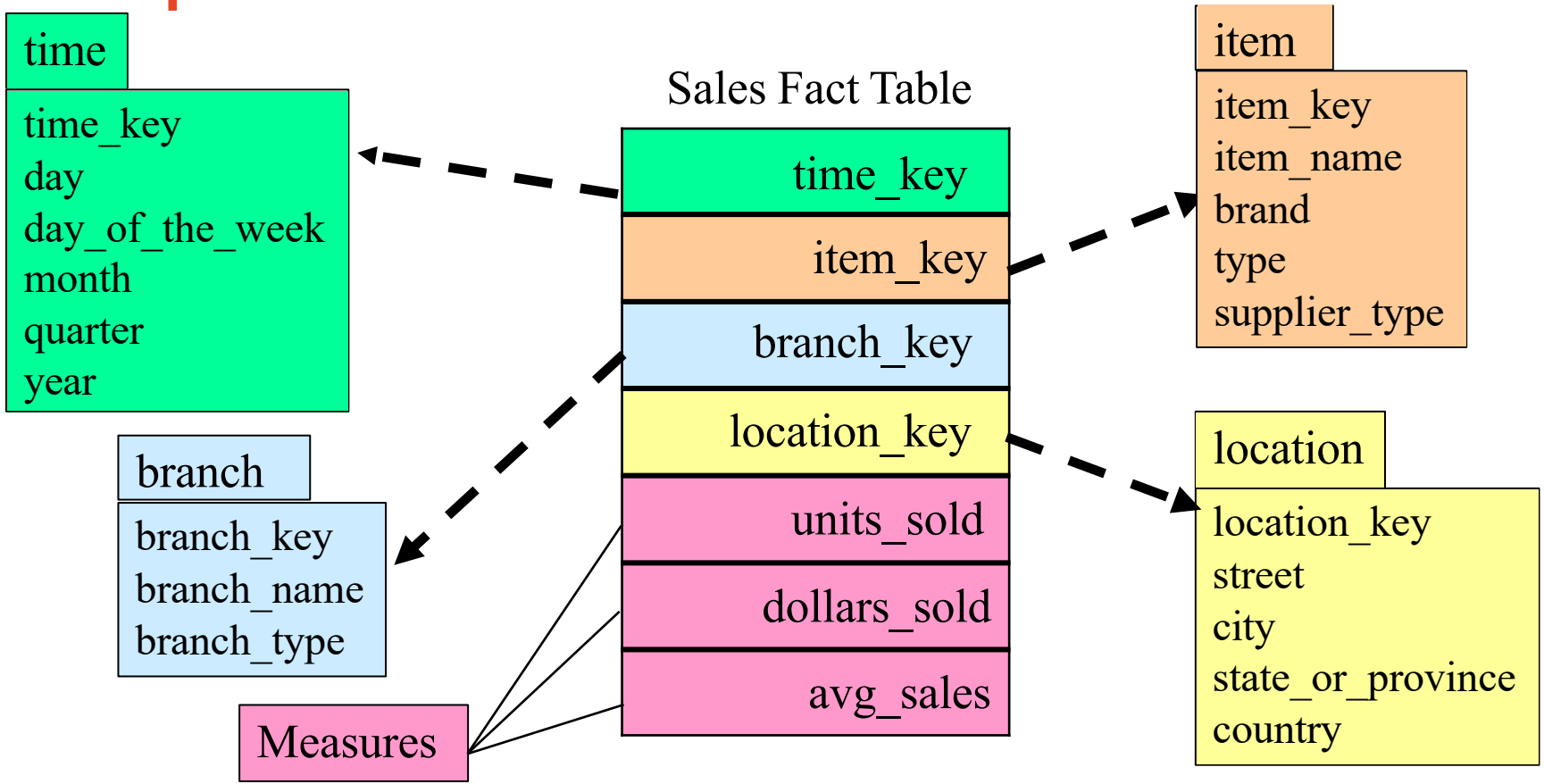
Data warehousing: Star schema

- The fact table and dimension tables linked to it look like a star.
  - This is called a **star schema**.
- Most common modelling paradigm.



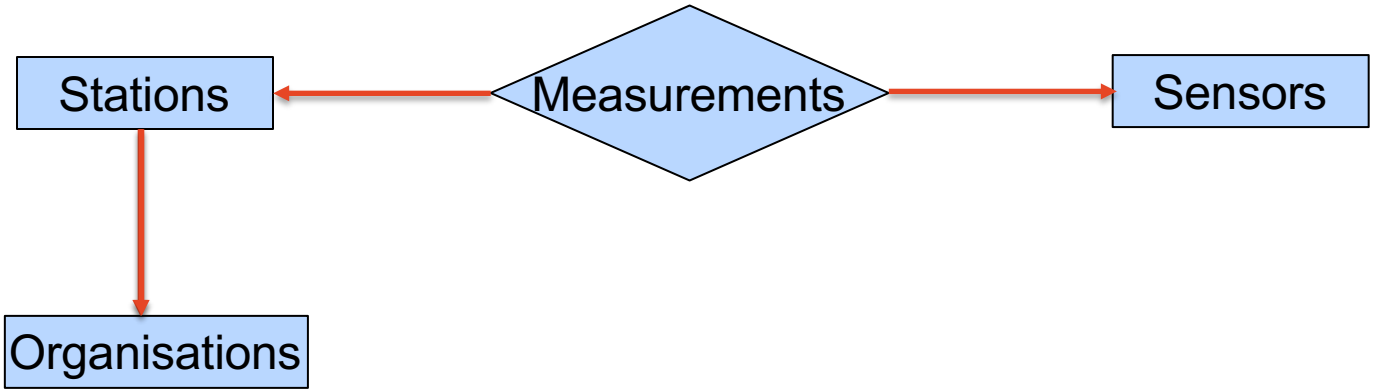
- If we map this to relations:
  - 1 central fact table.
  - n dimension tables with foreign key relationships from the fact table.
- The fact table holds the FKs referencing the dimension tables.

Example of star schema

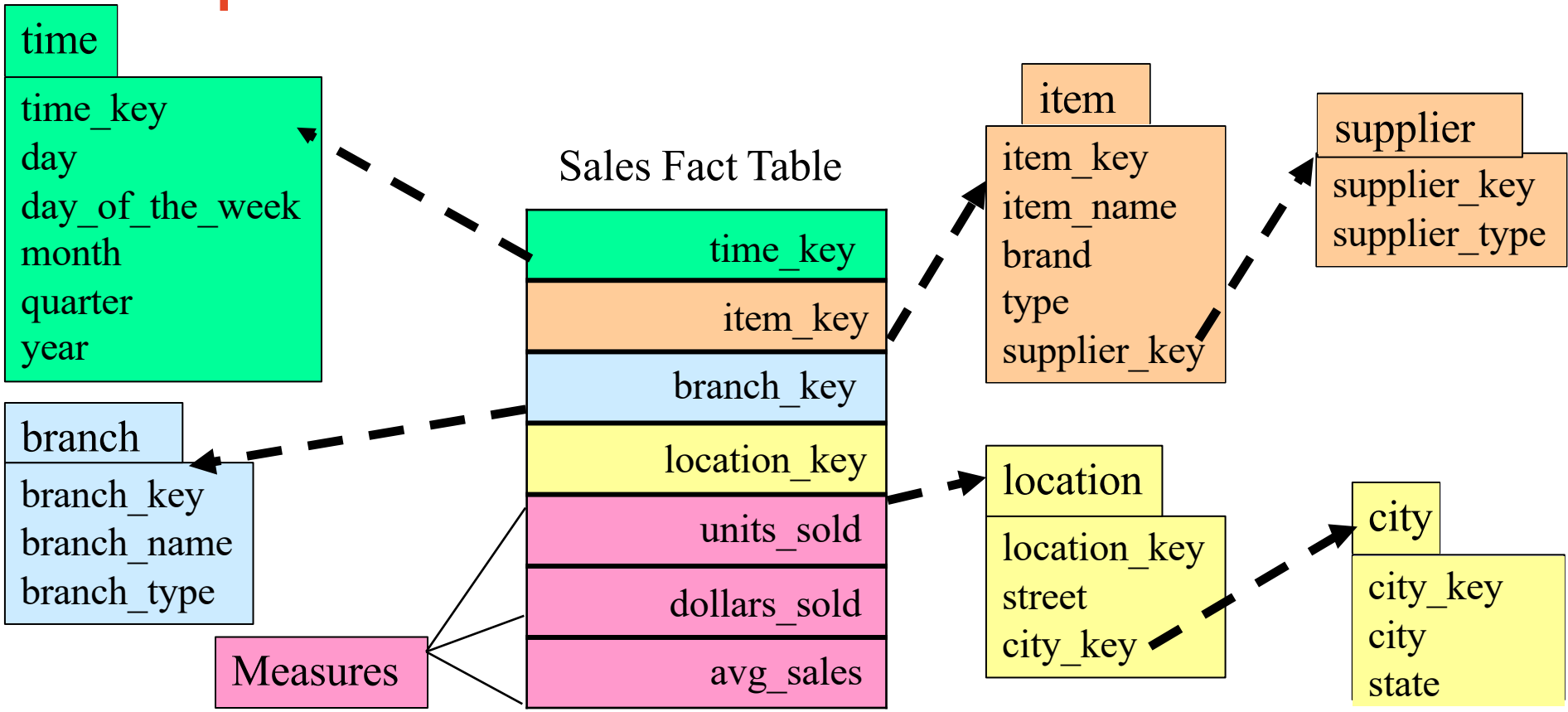


Data warehousing: Snowflake schema

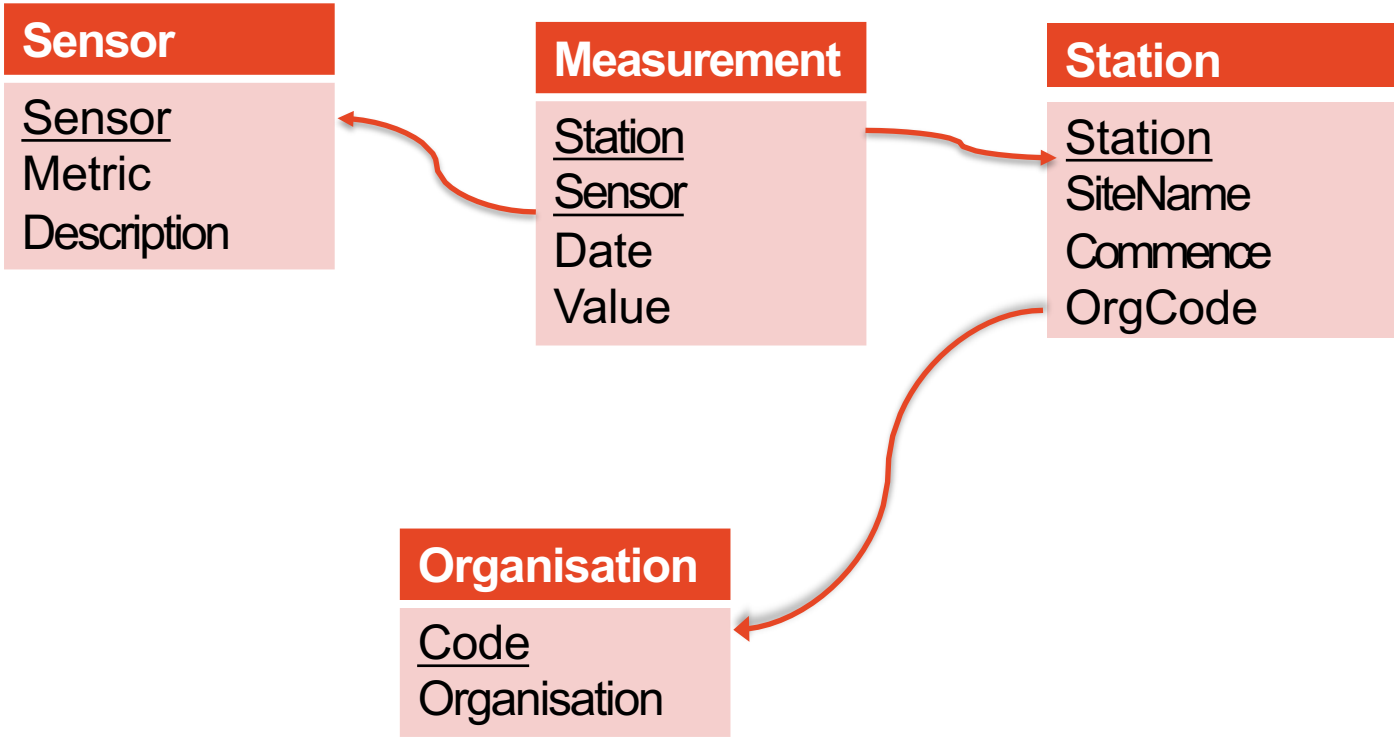
- A refinement of a star schema where some dimensional hierarchy is **normalized** into a set of smaller dimension tables, forming a shape similar to a snowflake.
- Measurements are the facts; the rest describes the dimensions.



Example of snowflake schema



Modeling our water data set



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.
- No-SQL options: document databases, key-value databases, wide-column stores, graph databases.

Review

W4 review: 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 24

Exercises

- Python/Jupyter to load data.
- Psycopg2.
- PostgreSQL to store data.

TO-DO in W4

- Ed Lessons Python modules 10-12.
- Ed Lessons SQL modules 20-21.