WEEK 2

Data types:数据种类:

Categorical data: 即非量化数据 包含nominal 和 ordinal

Nominal data: 由数字代表类别的数据 eg. 1 = white, 2 = blue, 3 = yellow

Dichotomous Data: 二分数据 eg. 1 = true, 0 = false;; 1 = male, 2 = female

Ordinal data: 由数字代表大小(非量化)的数据 eg. 1 = low, 2 = med, 3 = high

Quantitive data: 量化数据 包含interval和ratio

Interval data: 没有绝对零值的数据 eg. 温度(0度不代表绝对0度)

Ratio data: 有绝对零值的数据 eg. 长度

Week 3

SQL.

Key database concepts

**– Table** – an arrangement of related information stored in columns and rows.

**– Field / attribute** – coloumn in a table, contains homogenous set of information.

**– Field data types** - kind of data that can be stored in a field. For example, a field whose data type is Text can store data consisting of either text or number characters, but a Number field can store only numerical data.

**– Primary Key (PK)** – a field in a table whose value is uniquely identifies each record in the table. A PK cannot be null. (就是ID)

**– Record** – A row in table.

Foreign Key (比如说 一本书的ID是1 那它作者的id可以是2 这个2就是foreign key)

– When we need to refer to a record in a separate table we reference its ID as a foreign key.

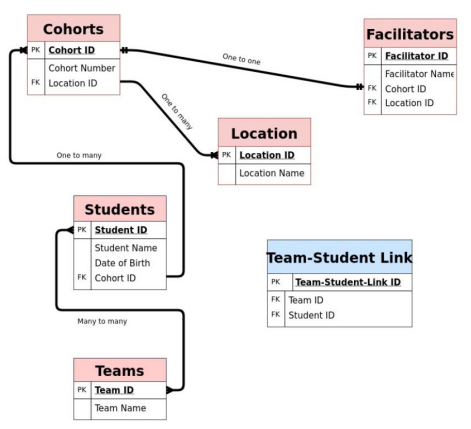
– A foreign key is defined in a second table, but it refers to the primary key or a unique key in the first table.

Entity Relationship

**– One-One Relationship (1-1 Relationship)（一行对一行）**

**– One-Many Relationship (1-M Relationship)（一行对多行）**

Entity Relationship Diagram



SQL example（重点！）

– The working-horse command: **SELECT(a)– FROM(b) – WHERE(c)** （从a处选取c条件的b）

– retrieves data (rows) from one or more tables of a relational database that fulfill a search condition

– Example 1:

SELECT \*

FROM Student

– Example 2:

SELECT name, email

FROM Student

WHERE sid=5312666

– Example 3:

SELECT COUNT(\*)

FROM Student

WHERE gender=‘f’

SELECT Statement

– 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

More SELECT Statement Options

|  |  |
| --- | --- |
| SQL Statement | Meaning |
| SELECT COUNT(\*) FROM *T* | count how many tuples(元组) 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 *T* |
| SELECT \* FROM *T* ORDER BY *a* | order the result by attribute *a* (in ascending order; add DESC for descending order) |

SQL Data Types

– Integers

– Floats, Numeric

– 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**（并列）: || (eg. 'hello ' || 'there')

– Date, Timestamp

– DATE

– TIME

– TIMESTAMP

– INTERVAL

– CURRENT\_DATE: db system’s current date

– CURRENT\_TIME: db system’s current timestamp

– 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’

• Oracle syntax: TO\_DATE(’01-03-2012’, ‘DD-Mon-YYYY’)

– +/- INTERVAL:

• e.g. ‘2012-04-01’ + INTERVAL ’36 HOUR’

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 Table

WHERE textTypeField LIKE 'H%';

NULL Values

– The predicate is null or is not null can be used to check for nulls

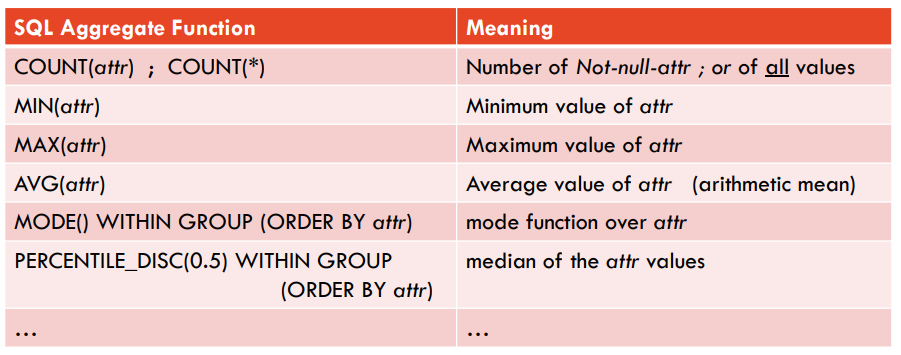
– e.g. Find measurements with an unknown intensity error value.

SELECT \*

FROM Measurements

WHERE FieldName IS NULL

SQL Aggregate（整合） Functions



Summarising a Database with SQL

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 （把不同的两组数据通过一个ID联系到一起）

– 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 galaxy data.

SELECT \*

FROM Galaxy JOIN Measurement USING (gid)

LIMIT 3;

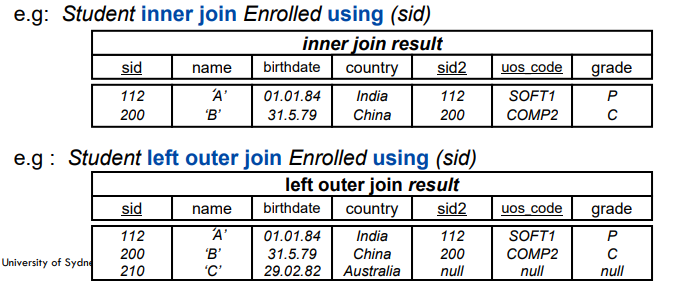
– Which measurements were taken at stationed 409204?

SELECT \*

FROM measurements m INNER JOIN stations s ON m.stationid=s.id

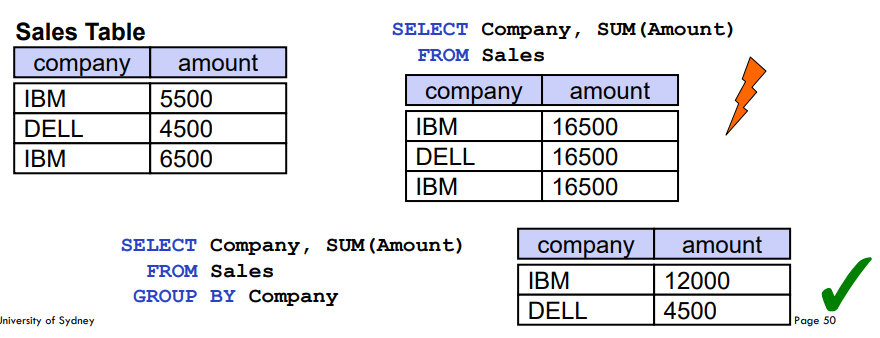
WHERE stationid = 409204;

例子：



SQL Grouping（根据同种类（同group）数据做出操作）

例子：



Example: Filtering Groups with HAVING Clause

– GROUP BY Example:

– What was the average mark of each course?

SELECT uos\_code as unit\_of\_study, AVG(mark)

FROM Assessment

GROUP BY uos\_code

– HAVING clause: can further filter groups to fulfil a predicate

– Example:

SELECT uos\_code as unit\_of\_study, AVG(mark)

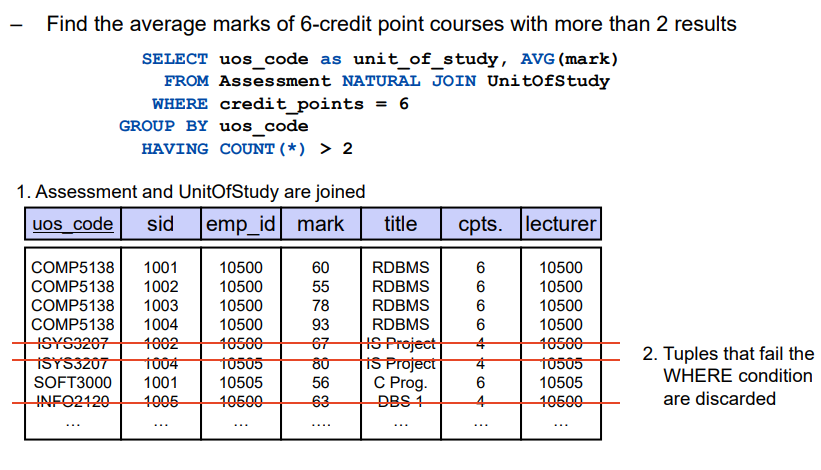
FROM Assessment

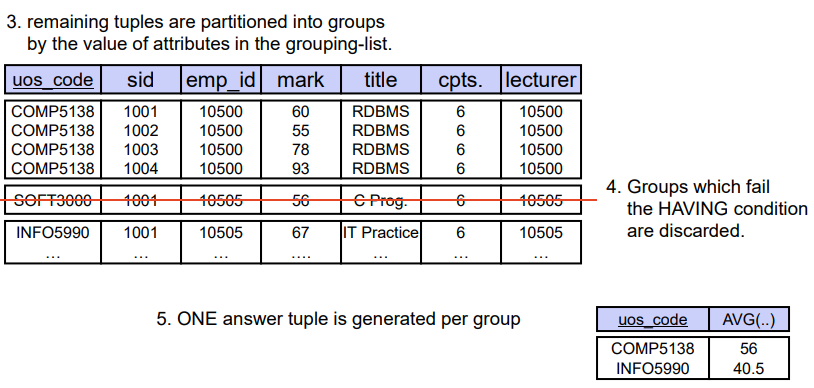
GROUP BY uos\_code

HAVING AVG(mark) > 10

– Note: Predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

例子：





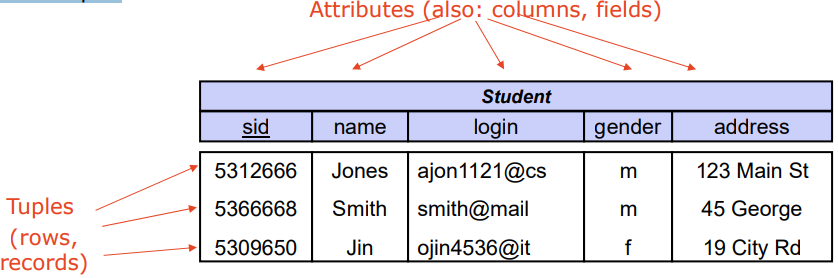
Week4

Relational Databases

– Informal Definition: A **relation** is a **named, two-dimensional** table of data

– Table consists of rows (record) and columns (attribute or field)

– Example:



SQL DML statements

– 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

– **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’

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)*

);

Combining data from multiple tables

For example, the following query lists all stations belong to Victoria Government

set search\_path to waterwaysdata;

SELECT \* FROM Stations, Organisations

WHERE Stations.owner = Organisations.code AND Organisations.name = 'Victoria Government';

All tables accessed by the query are listed in the FROM clause, separated by comma.

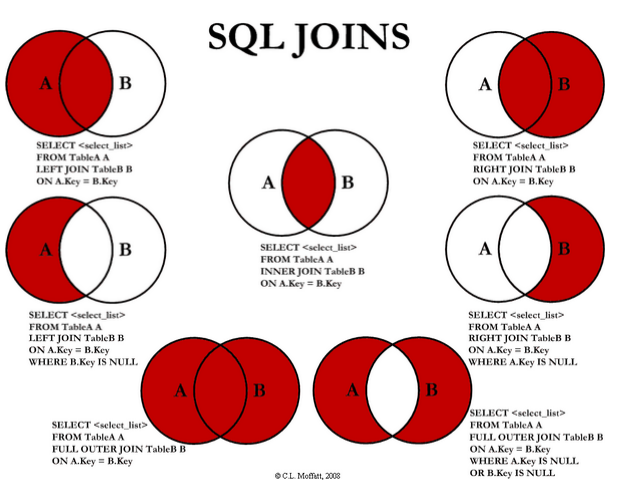
Natural Joins

A natural join between two tables combines those rows whose **values agree** in all common attributes, ie. those that are **named the same**. These common columns are also shown only once in the result.

SELECT \* FROM measurements NATURAL JOIN sensors;

If there is no common attribute in two tables; the query produces a cross join, also called the cross product of both tables (undesired result).

SQL joins



SQL Subqueries

• A subquery is a query within a query. You can create subqueries within your SQL statements. These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

SELECT \* FROM measurement WHERE sensor in ( SELECT sensor FROM sensor );

select \* from measurement where sensor='temp'

and value>( select AVG(value) from measurement where sensor='temp' );

week 5 Scalable Analytics: The Role of Indexes and Data Partitioning数据目录与分区

Storage Hierarchy

– **primary** storage: Fastest media but **volatile** (cache, RAM).

– **secondary** storage: next level in hierarchy, **non-volatile**, moderately fast access time

– also called **on-line storage**

– E.g.: hard disks, solid-state drives

– **tertiary** storage: lowest level in hierarchy, **non-volatile**, slow access time

– also called **off-line storage**

– E.g. magnetic tape, optical storage

– Typical storage hierarchy:

– Main memory (RAM) for **currently used** data.

– Disk for the **main database** (secondary storage).

– Tapes for archiving **older versions** of the data (tertiary storage).

Accessing a Database Disk Page

– Time to access (read/write) a disk block:

– **seek time** (moving arms to position disk head on track)

– **rotational delay** (waiting for block to rotate under head)

– **transfer time** (actually moving data to/from disk surface)

– Key to lower I/O cost: reduce seek/rotation delays!

Alternative File Organizations

– **Heap Files**

– a record can be placed anywhere in the file where there is space (random order)

– suitable when typical access is a file scan retrieving all records.

– **Sorted Files**

– store records in sequential order, based on the value of the search key of each record

– best if records must be retrieved in some order, or only a `range’ of records is needed.

– **Indexes**

– data structures to organize records via trees or hashing

– like sorted files, they speed up searches for a subset of records, based on values in certain (“search key”) fields

– Updates are much faster than in sorted files.

(Unordered) Heap Files

– Simplest file structure contains records in **no particular order**.

– Access method is a **linear scan**

– In average half of the pages in a file must be read, in the worst case even the whole file

– Efficient if all rows are returned (SELECT \* FROM table)

– Very **inefficient** if **a few** rows are requested

– Rows appended to end of file as they are inserted

– Hence the file is unordered

– Deleted rows create gaps in file

– File must be periodically compacted to recover space

Sorted File

– Rows are **sorted** based on some attribute(s)

– Successive rows are stored in same (or successive) pages

– Access method could be a **binary search**

– Equality or range query based on that attribute has cost **log2B** to retrieve page containing first row

– Problem: **Maintaining sorted order**

– After the correct position for an insert has been determined, shifting of subsequent tuples necessary to make space (very expensive)

– Hence sorted files typically are not used per-se by DBMS, but rather in form of index-organised (clustered) files

Indices（目录）

– Just remember a book index: Index is a set of pages (a separate file) with pointers (page numbers) to the data page which contains the value

– Instead of scanning through whole book (relation) each time, using the index is much faster to navigate (less data to search)

– Index typically much smaller than the actual data

**Index Definition in SQL**

– Create an index

CREATE INDEX <name> ON <relation-name (attribute list) >

– Example:

CREATE INDEX <StudentName> ON <Student(name) >

– Index on primary key generally created automatically

– Use CREATE UNIQUE INDEX to indirectly specify and enforce the condition that the search key is a candidate key.

– Not really required if SQL unique integrity constraint is supported

– To drop an index

DROP INDEX <index-name>

Clustering Index （聚簇索引）

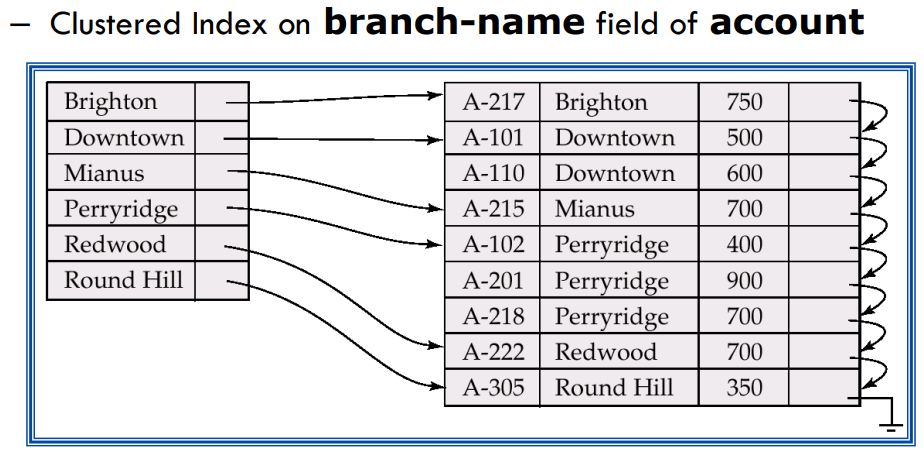
– index entries and rows are ordered in the same way

– The particular **index structure** (eg, hash, tree) dictates how the rows are organized in the storage structure

– There can be **at most one clustering index** on a table

– e.g The white pages of the phone book in alphabetical order.

– CREATE TABLE statement generally creates an clustered index on primary key

例子：

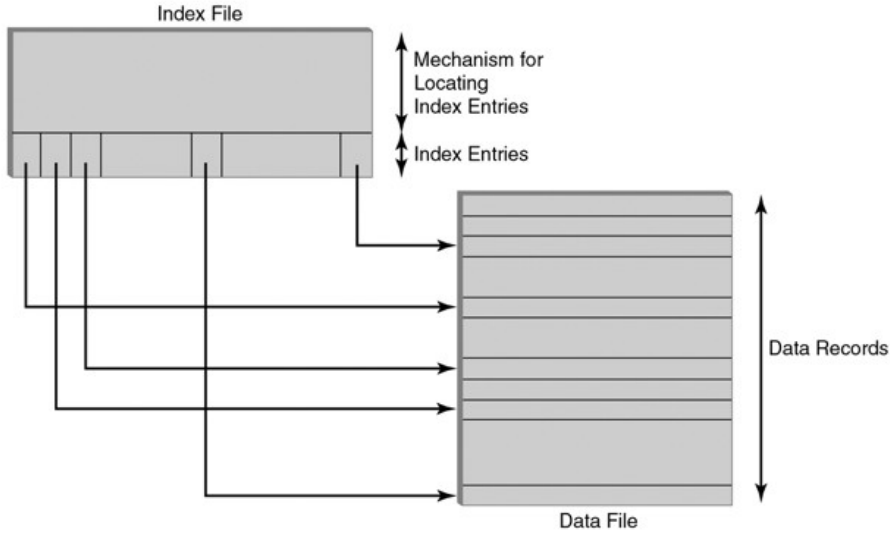
Unclustered Index

– Index entries and rows are not ordered in the same way

– There can be many secondary indices on a table

– Index created by CREATE INDEX is generally an unclustered, secondary index

例子：



Choosing an Index

– An index should support a query of the application that has a significant impact on performance

– Choice based on frequency of invocation, execution time, acquired locks, table size

– Example 1:

SELECT E.Id FROM Employee E

WHERE E.Salary < :upper AND E.Salary > :lower

– This is a **range search** on Salary.

– Since the primary key is Id, it is likely that there is a clustered, main index on that attribute that is of no use for this query.

– Choose a B+ tree index with search key Salary

– Example 2:

SELECT T.studId FROM Transcript T

WHERE T.grade = :grade

– This is an **equality search** on grade.

– We know the primary key is *(studId,* *semester, uosCode*)

– It is likely that there is a main, clustered index on these PK attributes

– but it is of no use for this query…

– Hence: Choose a B+ tree index (or hash index) with search key *Grade*

– Again: a **covering index** with composite search key (*grade, studId*) would allow to answer complete query out of index

• but then only as B-Tree index…

– Example 3:

SELECT T.uosCode, COUNT(\*) FROM Transcript T

WHERE T.year = 2009 AND T.semester = ‘Sem1’ GROUP BY T.uosCode

– This is a **group-by query** with an **equality search** on *year* and *semester*.

– If the primary key is (*studId, year, semester, uosCode*), it is likely that there is a clustered index on these sequence of attributes

– But the search condition is on year and semester => must be prefix!

– Hence PK index not of use

– Covering INDEX: (*year, semester, uosCode*)

Distributed Databases

– **Data Partitioning**

– Storing sub-sets of the original data set at different places

• can be in different tables in schema on same server, or at remote sites

– Goal is to query smaller data sets & to gain scalability by parallelism

– Sub-sets can be defined by

• columns: **Vertical Partitioning** （竖着切）

• rows: **Horizontal Partitioning** (横着切)(if each partition is stored on a different site also called Sharding)

Week 6 text data processing

Structured data

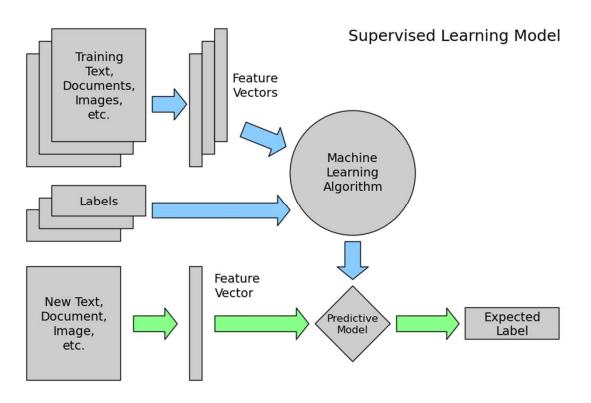
* Fielded
* Stored in database

**Text data**

* Unstructured
* Text-heavy
* Ambiguities
* Eg. Images, videos, email, social media

**Text categorisation**

**Spam detection** as **supervised** classification



**Feature vectors from text**

– Represent document as a multiset of words

– Keep frequency information

– Disregard grammar and word order

Tokenisation

– Split a string (document) into pieces called tokens

– Possibly remove some characters, e.g., punctuation

– eg. “Friends, Romans, Romans, countrymen” -> [“Friends”, “Romans”, “Romans”, “countrymen”]

Normalisation

– Map similar words to the same token

– Stemming/lemmatisation

– Avoid grammatical and derivational sparseness

– E.g., “was” => “be” – Lower casing, encoding – E.g., “Naïve” => “naive”

–eg. [“Friends”, “Romans”, “Romans”, “countrymen”] -> [“friend”, “roman”, “roman”, “countrymen”]

Indicator features

– Binary indicator feature for each word in a document

– Ignore frequencies

–eg. [“friend”, “roman”, “roman”, “countrymen”] ->{“friend”: 1, “roman”: 1, “countrymen”: 1}

**Term frequency** weighting(TF)

– Term frequency

– Give more weight to terms that are common in document

**– TF = |occurrences of term in doc|**

– Damping

– Sometimes want to reduce impact of high counts

**– TF = log(|occurrences of term in doc|)**

Eg. [“friend”, “roman”, “roman”, “countrymen”] ⬇ {“friend”: 1, “roman”: 2, “countryman”: 1}

**TFIDF** Weighting

– **Inverse document frequency(IDF)**

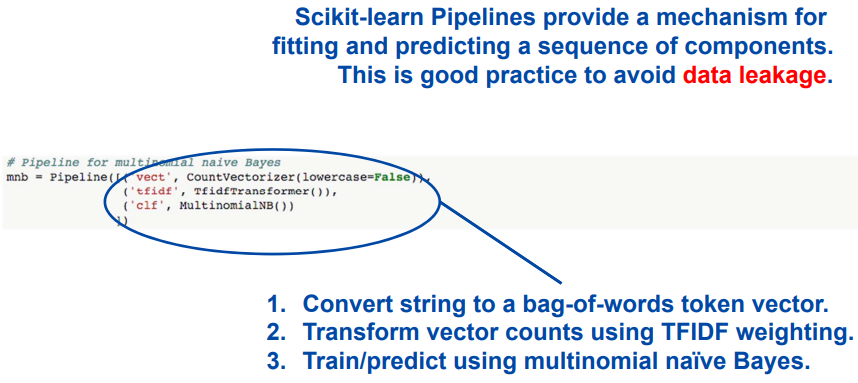
– Give less weight to terms that are common across documents

– **IDF = log(|total docs|/|docs containing term|)**

– T**FIDF – TFIDF = TF \* IDF**

Eg.[“friend”, “roman”, “countrymen”] ⬇ {“friend”: 0.1, “roman”: 0.8, “countrymen”: 0.2}

Use **scikit-learn Pipeline** to manage **cross validation**



The curse of dimensionality

– In practice, the inclusion of **more features** leads to **worse performance**

– The number of training examples required increases **exponentially** with dimensionality.

**Text-driven forecasting**

Definition: Given a body of text ***T*** pertinent to a social phenomenon, make a concrete prediction about a measurement ***M*** of that phenomenon, obtainable only in the future.

Some text-driven forecasting tasks

– Predict box office gross for films

– T: description, script, reviews, etc

– M: how much the film earns at the box office

– Predict volatility of a stock

– T: annual report, etc

– M: volatility over the following year

– Predict blog reader behaviour

– T: political blog posts, etc

– M: number of reader comments

Collect data from ***DBpedia***

**Information extraction**

Natural language processing

– Understanding

– Tokenisation

– POS tagging

– Parsing

– Generation

– Summarisation

Information extraction

– Named entity recognition

– Entity disambiguation

– Relation extraction

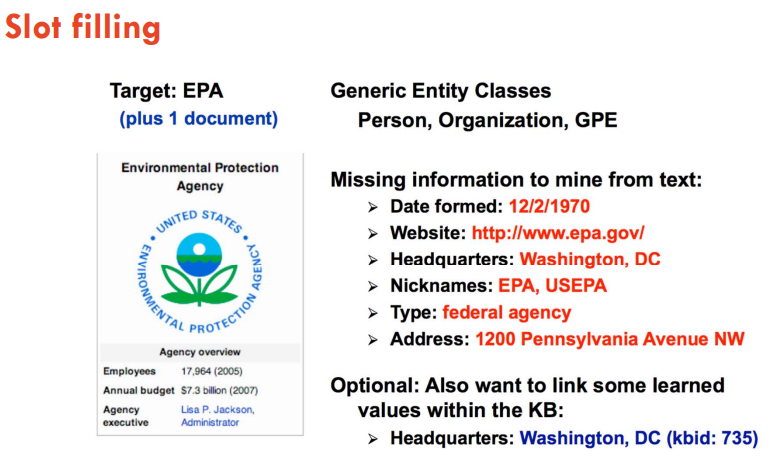
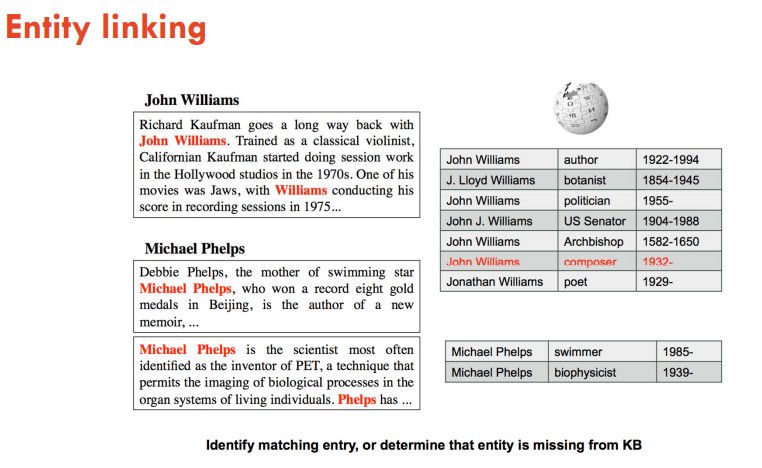
Knowledge base population (KBP)

– Aim is to build structured knowledge bases from massive unstructured text corpora

– Two subtasks:

– Entity linking: identify mentions of entities, link to KB or NIL

– Slot filling: extract and populate facts for given entity



WEEK8 Time Series Data

Learning Objectives

– Motivation for **Temporal Data**

– **Temporal Database** Concepts

– **Time data types**

– **Kinds of time**

– **Time queries**

– **Support for Temporal Data** with **current DBMS**

– **Time Series Data**

– **Use Case**

Temporal Data

– Almost all data is qualified with time (period or point)

* Eg. 天气预报（某天）
* Eg. 健康状况（随时间变化）

Big data vs Long data

– **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, 30 years.

– big data doesn’t explicitly include **historical information as a key component**

**Concepts in Temporal Databases**

– **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

– **Kinds of time**

– User-defined time an uninterpreted time value

– E.g. a birthdate or a publication time

– Valid time when a fact was true in the modelled reality

– records the time when a fact is true in the real world.

– Can move forward and backward

– Transaction time when a fact was stored in the database

records the history of database activity.

– Only moves forward (as you cannot go back in history and change things)

– Therefore allows rollback (very useful for auditing)

– **Temporal statements**

– Current now

“What is now?” • E.g. “How many products do we currently have in stock?”

– Sequenced at each instant of time

“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 ignoring time

“What was at any time?” • E.g. “How many products A did we have at any time in stock?”

Temporal Data Types

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

**Temporal Support in current database systems**(重点 代码！)

SQL:2011

– SQL:2011 adds **period definitions as metadata**（不是单独的datatype）

– 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)

**Tables** with **Range Types** in PostgreSQL

– new data type **DATARANGE** to represent (time) intervals

CREATE TABLE Emp (

eName VARCHAR,

ePeriod **DATERANGE**,

eDept VARCHAR );

INSERT INTO Emp VALUES ('Anton', '[2010, 2014]', 'SIT');

**Indexes on Range Types** in PostgreSQL

– Provides a powerful, extensible **index**: **General inverted search tree** (GiST)

– On periods and combinations of period and other attributes

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

examples：

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

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

SELECT \* FROM Dept WHERE upper\_inf(DPeriod) = TRUE

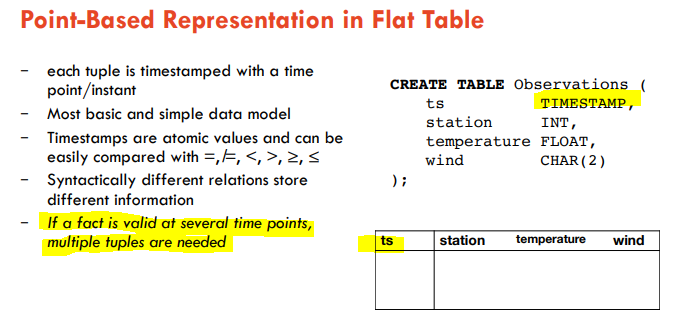
– Some additional functions on ranges: UNION (+), INTERSECTION (\*), DIFFERENCE (-)

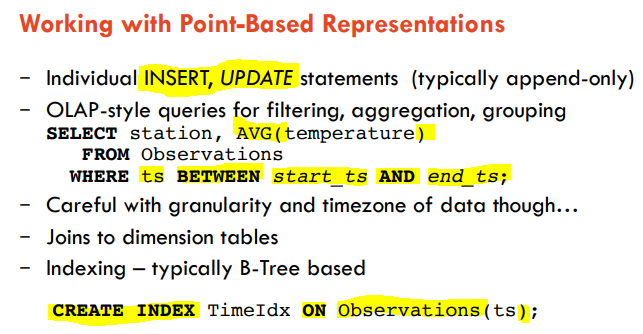
SELECT '[2010, 2013)' + '[2012, 2015)'

Timeseries Data（时序数据）

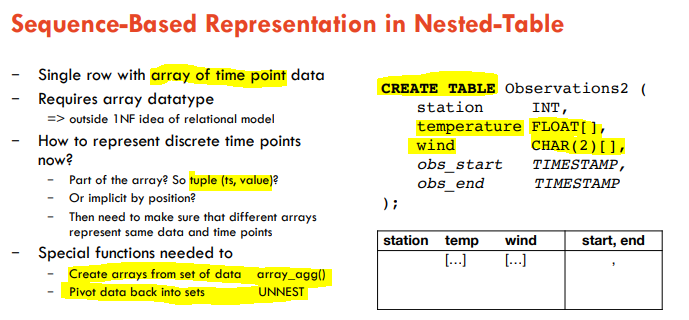
Point-based representation （时间点数据）

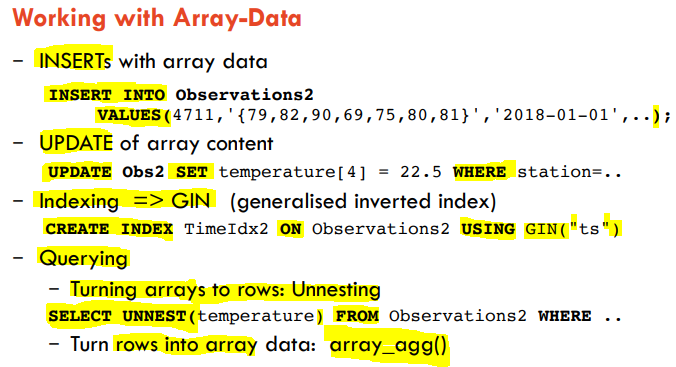
– Multiple rows with atomic data types





Sequence-Based Representation in Nested-Table（用数组表示序列数据）





Week9 scraping web data

Learning Objectives

– Web Scraping basics

– Retrieving data from Web Services

– Semi-structured Data

– HTML, XML and JSON

– Storing and querying semi-structured data in PostgreSQL

Web Scraping and Web Services

– Existing files: Excel Sheets, CSV, …

– Databases

– Querying existing databases with SQL

– Scraping the Web

– Web crawling + HTML parsing

– Programming APIs

– 'querying' web service APIs

– more details on following slides

**Web Scraping** – General Approach

– **Reconnaissance**

– Identify source, and check its structure and content

– **Webpage retrieval**

– Download one or multiple pages from source

– Typically in a script or program that auto-generates new URLs based on website structure and its URL format

– **Data extraction** from webpage

– Content parsing, raw data extraction

– **Data cleaning** and transformation into required format

– **Data storage** / Analysis / combining with other data sets

Data Cleaning

– Data from website hardly is in a clean format

– neither from the format

– nor from the content

– Rules of thumb:

– Be prepared that things are different than they are supposed to be (“ , ; \t)

– Clean data before further processing or storage

• Eg. empty cells; placeholders; special characters or excess spaces;

• Do it programmatically so that you can re-use your solution

– Cross-check data consistency once loaded; eg. spelling variants of same entity?

What we get: Semistructured Data

– JSON and XML are examples of so-called semistructured data models

– data with non-rigid structure

– Characteristics of Semistructured Data

– Missing or additional attributes

– Multiple attributes

– Nesting: semistructured objects ('documents') are hierarchical / have tree-structure

– Different types in different objects

– Heterogeneous collections

Week10 (Geo-)Spatial Data(空间数据)

**Learning Objectives**

– Spatial Data Types

– **Point/Raster Data** and **Spatial Objects**

– Querying Spatial Data with **Spatial Operations**

– Representing **Spatial Data in Databases**

– **Spatial Indexes**

– Spatial Data **Exchange with JSON and XML**

**Spatial Data**

– **Spatial data** is about objects and entities which have a location and/or a geometry

– A special form is **geospatial data** refers to a location on earth.

Types of Spatial Data

– **Point Data**

– **Points in a multidimensional space**

– E.g. geo-location of some data entities from location-aware apps

• As time-series: **trajectory data** of moving objects (such as cars)

– E.g. **raster data** such as satellite imagery, where each pixel stores a measured value

– **Region Data**

– Objects have spatial extent with location and boundary

– DB typically uses geometric approximations constructed using line segments, polygons, etc., called **vector data**.

Models of Spatial Information

Field-based Model

– Three main concepts:

– Spatial Framework is a partitioning of space

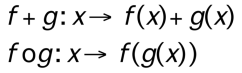
• e.g., grid imposed by Latitude and Longitude

– Field Functions:

f: Spatial Framework -> Attribute Domain

– Field Operations

• Examples, addition(+) and composition(o).



Types of Field Operations

– Local: value of the new field at a given location in the spatial frame-work depends only on the value of the input field at that location (e.g., Thresholding) (极点)

– Focal: value of the resulting field at a given location depends on the values that the input field assumes in a small neighborhood of the location (e.g., Gradient) （范围）

– Zonal: Zonal operations are naturally associated with aggregate operators or the integration function. An operation that calculates the average height of the trees for each species is a zonal operation. （平均）

– Exercise:

– (I) Identify peaks (points higher than its neighbors) - focal

– (II) Identify mountain ranges (elevation over 2000 feet) - local

– (III) Determine average elevation of a set of river basins - zonal

Object Model

– Object model concepts

– Objects: distinct identifiable things relevant to an application

• Objects have attributes and operations

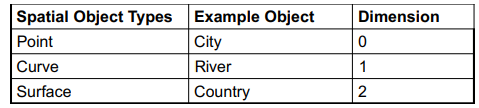
– Attribute: a simple (e.g. numeric, string) property of an object

– Operations: function maps object attributes to other objects

Classifying Spatial Objects

– Simple

• 0- dimensional (points), 1 dimensional (curves), 2 dimensional (surfaces)

Eg. 

– Collections • Polygon collection (e.g. boundary of Japan or Hawaii)

Coordinate Systems

– Each instance of a spatial type has a spatial reference identifier (SRID)

– Different SRID means different results for some operations, such as area.

– List typically according to prevalent standards (EPSG/OGP).

– Types

– Cartesian Coordinates • point positions from a defined origin along axes on a plane (轴线坐标)

– Geodetic Coordinates (Geographic Coordinates) • angular coordinates (longitude and latitude) （经纬坐标）

– Projected Coordinates • are planar Cartesian coordinates that result from performing a mathematical mapping from a point on the Earth surface to a plane（投射坐标）

**Types of Spatial Queries**

– **Spatial Range Queries**

– *Find all cities within 50 kilometers of Sydney*

– Query has **associated region** (location, boundary)

– Answer includes **overlapping** or contained **data regions**

– **Nearest-Neighbor Queries**

– *Find the 3 cities nearest to Sydney*

– Results must be **ordered by proximity**

– **Spatial Join Queries**

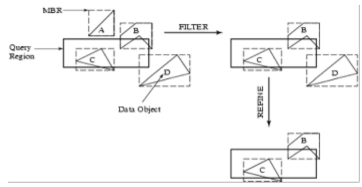
– Find all cities near a lake

– Expensive, join condition involves **regions and proximity**

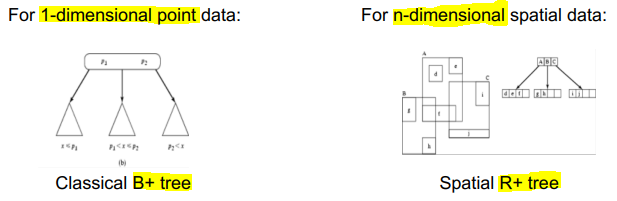
**Spatial Query Processing: Filter-Refine Strategy**

– **Filter Step**: Objects with minimum bounding box (MBR) intersecting query regions

– **Refine Step**: Query region really intersecting only with B and C



**Indexing of Spatial Data**



**The R-Tree**

– The R-tree is a tree-structured index that remains balanced on inserts and deletes.

– Each key stored in a leaf entry is intuitively a box, or collection of intervals, with one interval per dimension.

**R-Tree Properties**

– Leaf entry = < n-dimensional box, rid >

– This is Alternative (2), with key value being a box.

– Box is the tightest bounding box for a data object.

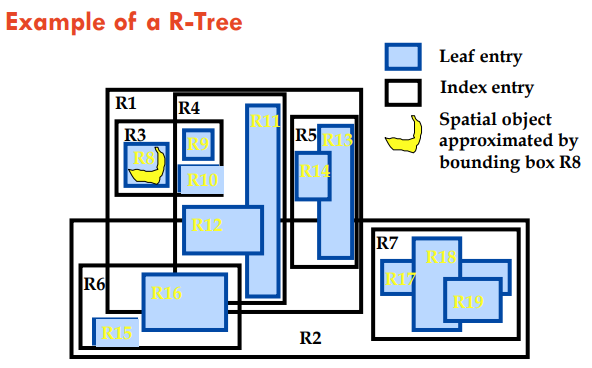
– Non-leaf entry = < n-dim box, ptr to child node >

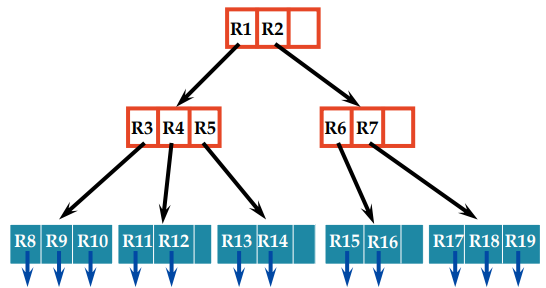
– Box covers all boxes in child node (in fact, subtree).

– All leaves at same distance from root.

– Nodes can be kept 50% full (except root).

– Can choose a parameter m that is <= 50%, and ensure that every node is at least m% full.





**Data-Exchange of Spatial Data using XML and JSON (spatial web APIs)**

Motivation: Geo-aware Web API

– Web APIs are very important for modern/mobile applications

– Need to be able to represent spatial data in JSON or XML formats

– GeoJSON

– KML

Use Cases

1. Ingest Data with spatial features

– Such as geographic location (point data) or a geometry (spatial object)

– Store and index in database for later processing

• Spatial indexing important

– Be aware of spatial query types and spatial joins

1. Use web APIs with support for GeoJSON to lookup further data

– Eg. Boundaries of localities

– Parsing GeoJSON required => semi-structured data techniques of last week

1. Use GeoJSON or KML to export or visualize spatial data

Week11 Image Processing

Image Data Processing / Analysis

Image analysis is the extraction of meaningful information from images; mainly from digital images by means of digital image processing techniques.

Use Case Scenarios

Use Case 1: Measurement of Proteins and Telomeres Colocalisation

Use Case 2: Measurement of telomere lengths

Use Case 3: Quantification of DNA and Mitotic Spindles

**Image Data**

Types of Images

• **True Colour or RGB Image**: Each **pixel** has a particular color; that color is described by the amount of pixels in **RGB channel** (red, green and blue)

• **Gray-scale image** - **Single channel** - Each pixel is a shade of gray.

• Binary image - Each pixel is just black (0) or white (1) - **Single channel** - Referred as **‘mask’** in the image processing domain

**Aspects of Image Processing**

– **Image Enhancement**: Processing an image so that the result is more suitable for a particular application. (sharpening or deblurring an out of focus image, highlighting edges, improving image contrast, or brightening an image, removing noise)

– **Image Restoration**: This may be considered as reversing the damage done to an image by a known cause. (removing of blur caused by linear motion, removal of optical distortions)

– **Image Segmentation**: This involves subdividing an image into constituent parts, or isolating certain aspects of an image. (finding lines, circles, or particular shapes in an image, in an aerial photograph, identifying cars, trees, buildings, or roads.

**Data Analysis Using Images**

– Removal of noise

– Extraction of region of interest (ROI) using binary mask

• Image enhancement

– Okay to extract ROI or shape features

– Not-Okay for intensities-based analysis

– Measurements

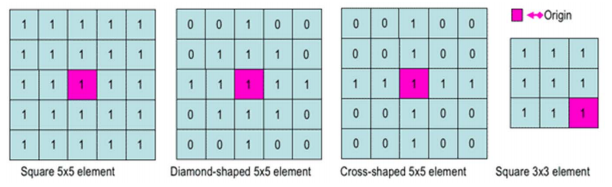
• Binary image

• Greyscale intensities

**Morphological Image Processing**

– Morphological techniques probe an image with a small shape or template called a **structuring element**.

– The **structuring element** is a small binary image, i.e. a small matrix of pixels, each with a value of zero or one

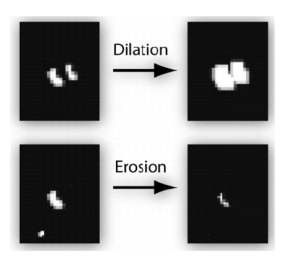


**Morphological Dilation**

– The value of the output pixel is the **maximum** value of all the pixels in the input pixel's neighborhood.

**Morphological Erosion**

– The value of the output pixel is the **minimum** value of all the pixels in the input pixel's neighborhood.



**Image Enhancement**

– Image denoising

– Point spread function (PSF)

– Gussain blurr

**Digital filters**

– In image processing filters are mainly used to suppress either the high frequencies in the image, i.e. smoothing the image, or the low frequencies, i.e. enhancing or detecting edges in the image.

– Median filter

– Average filter

– Weiner filter

– Spatial filter

Point Spread Function (PSF)

– PSF is described as the impulse response of the optical system. It blurs out any point-like object to a certain minimal size and shape called the Point Spread Function (PSF).

– The captured image become convoluted.

Deconvolution

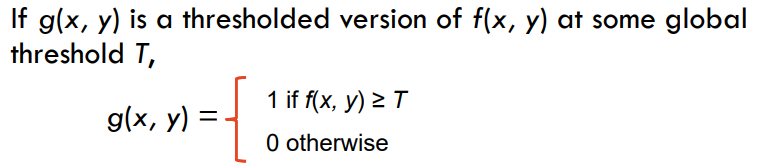
– Deconvolution is an algorithm-based process used to reverse the effects of convolution on an image.

**ROI Segmentation by Thresholding and Histogram**

Thresholding

– Thresholding creates binary images from grey-level ones by turning all pixels below some threshold to zero and all pixels about that threshold to one.

– the separation of light and dark regions



Thresholding Algorithms

– Global thresholding choose threshold T that separates object from background

Thresholding issues

– Many objects at different gray levels

– Variations in background gray level

– Noise in image.

Global vs Adaptive/local thresholding

– Local threshold T(x, y) is calculated for each pixel, based on some local statistics such as range, variance, or surface-fitting parameters of the neighborhood pixels within a local block of size wxw.

Binary Image Uses

– Object recognition

– Spatial location

– Size measurements

– Surveillance

**histogram of the pixel intensity values**

– Histograms plots how many times (frequency) each intensity value in image occurs

– Different images can have same histogram

– Half of pixels are gray, half are white – same histogram = same statistics

– We **can’t** reconstruct image from histogram