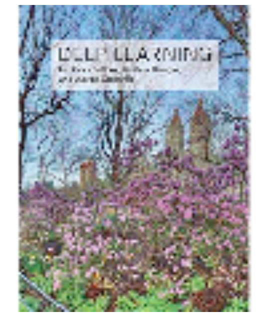
Databases & Datamining in Astronomy - 2017

git, SQL & some introductory remarks

Literature

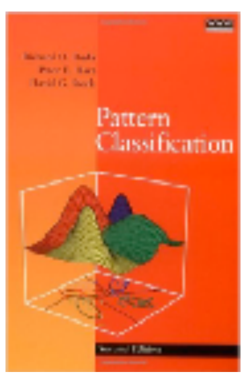
No obligatory book, but I will roughly follow:

Statistics, Data Mining, and Machine Learning in Astronomy - *Ivezic, Connolly, VanderPlas & Gray*

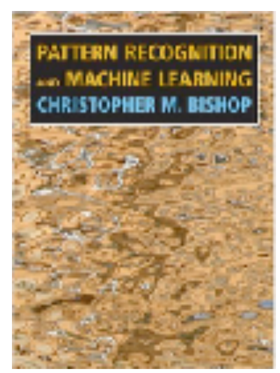


with some inspiration from:
Deep Learning - *Goodfellow, Bengio & Courville*

Other useful books among many others:



Pattern Classification - *Duda, Hart & Stork*

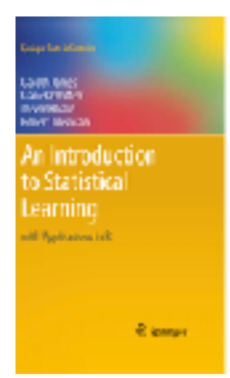


Pattern Recognition and Machine Learning - *Bishop*

Bayesian Data Analysis - *Gelman*

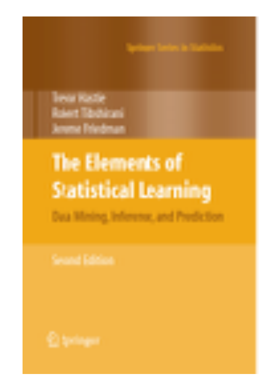
Information Theory, Inference and Learning Algorithms - *MacKay*

Online



Introduction to Statistical Learning - *James et al*

Online



Elements of Statistical Learning - *Hastie et al*

Online

Course structure

4 lectures & 4 practical classes.

Practical experience is crucial.

Evaluation:

Practical project

Lecturer: Jarle Brinchmann
Room: O451
Email: jarle@strw.

Assistant: Alex Mechev
Room: H404
Email: apmechev@strw.

Overview of the course

- 14/9 - git, SQL & Python intro. Regression & regularisation & subset selection. Introduce feature extraction & scaling. Quick tour of datamining [45 min] Statistics background & Python refresher as home-study.
- 28/9 [practical] git/SQL/regression.
- 5/10 - Visualisations, Classic & Bayesian inference. Model comparisons. Complexity/model choice, cross-validation.
- 12/10 [practical] - recap of previous practical. Markov Chain Monte Carlo practical. BIC/AIC.
- 19/10 - Density estimation - histograms, kernel estimators, KNN, GMM. Classification, linear discriminants. Dimensional reduction & PCA
- 26/10 [practical] Density estimation. Extreme-deconvolution, KNN, GMM etc.
- 2/11 - Neural networks & deep learning.
- 9/11 [practical]

Version control using git

Managing your work - version control

When you carry out a complex task, you are advised to use a version control (VC) system. We will use **git**.

A VC helps keep track of changes made to your documents/code/whatever and allows you to branch out to try something new.

There are multiple possible ways to use git - local repositories and online repositories. Here we will use an online repository: github [github.com]

(there are many others, e.g. bitbucket.org)

Also: Very important inside & outside academia!

Making a local repository

Create a working directory and go into it

```
> mkdir Project
> cd Project
```

Initialise a git repository:

```
> git init
```

Put a file in the repository - this is the simplest way

```
> touch README
```

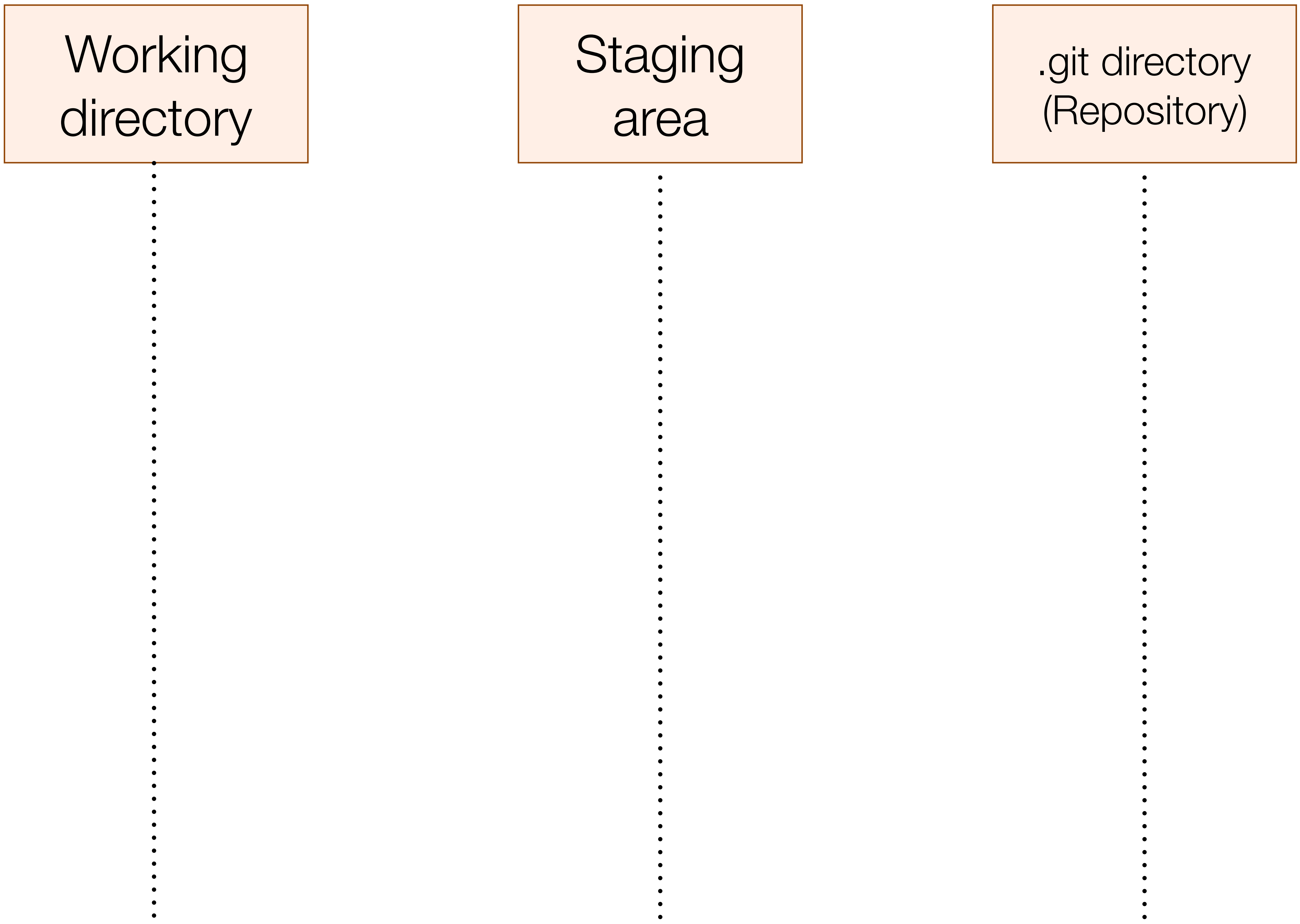
And check the status

```
> git status
```

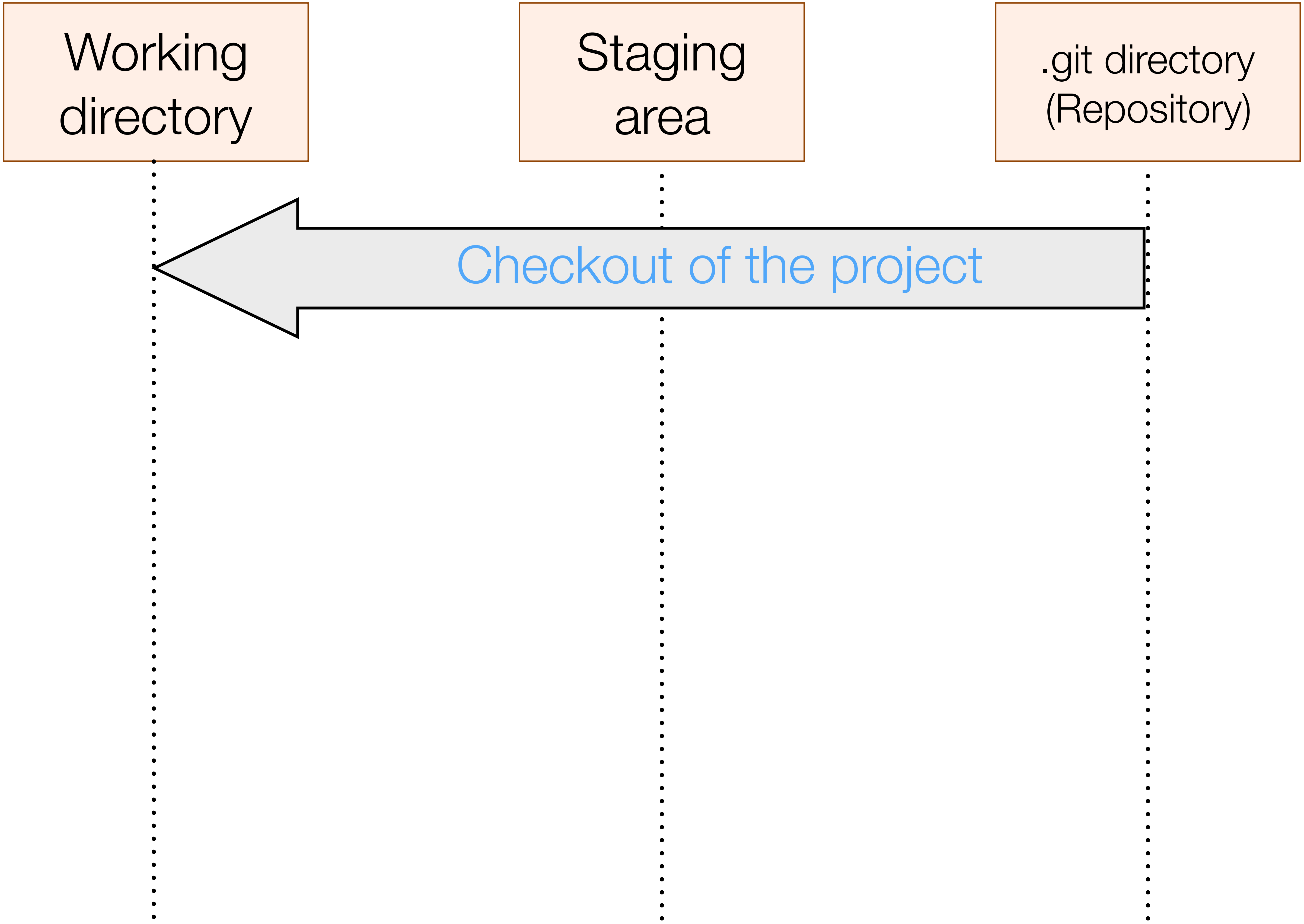
Making a local repository - still!

```
> git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be
committed)

   README

nothing added to commit but untracked files present
(use "git add" to track)
```

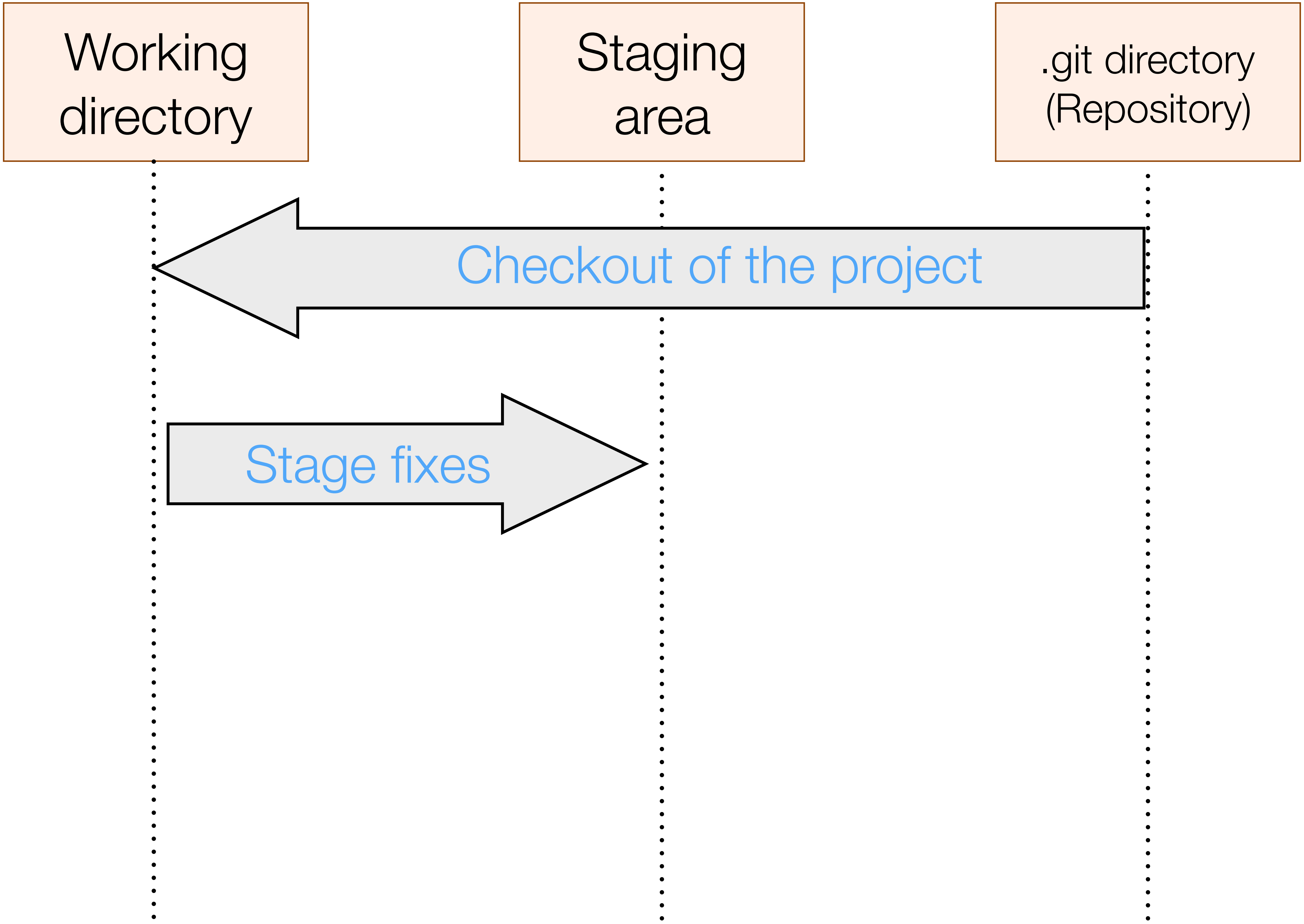
Staging, committing & pushing



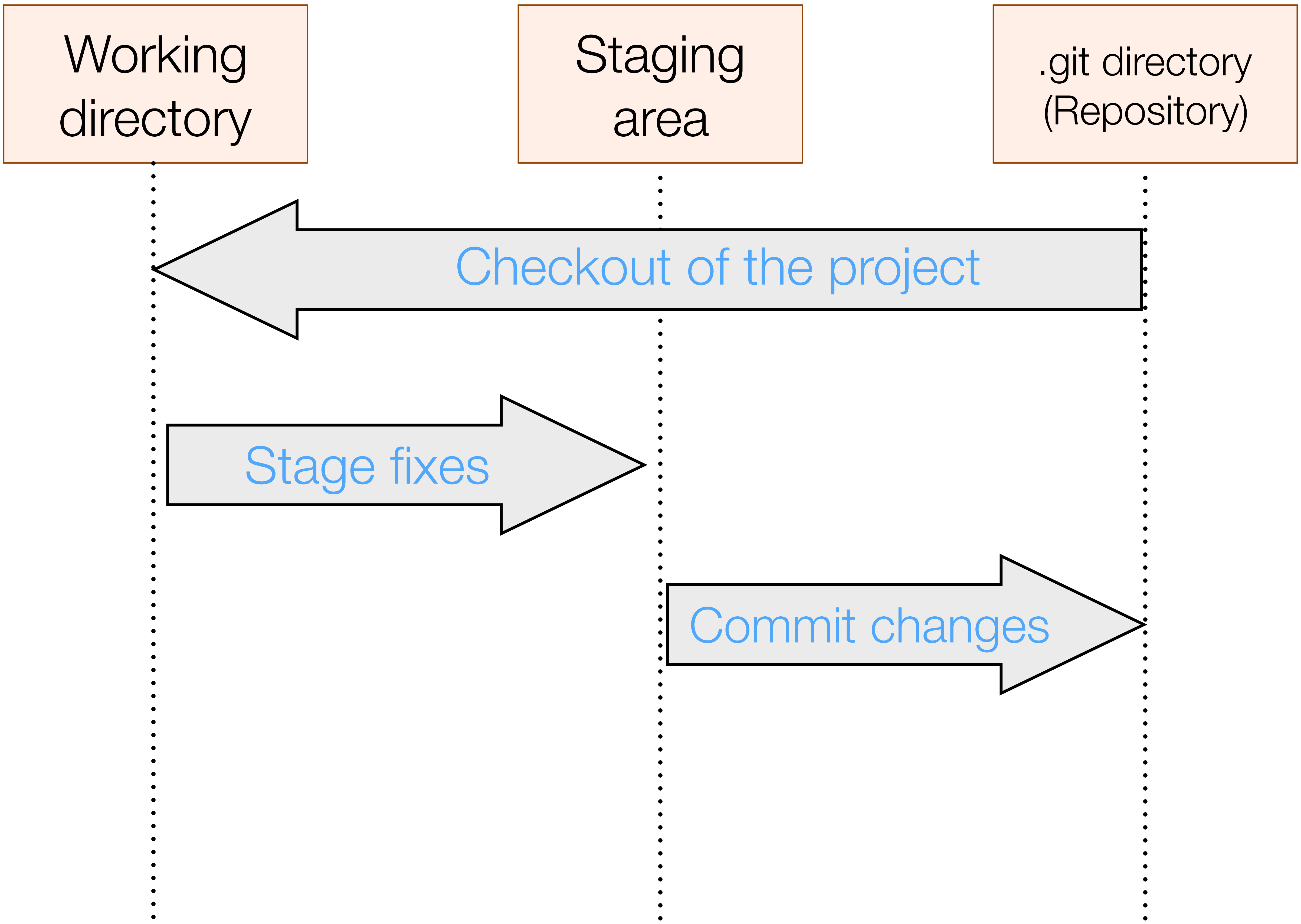
Staging, committing & pushing



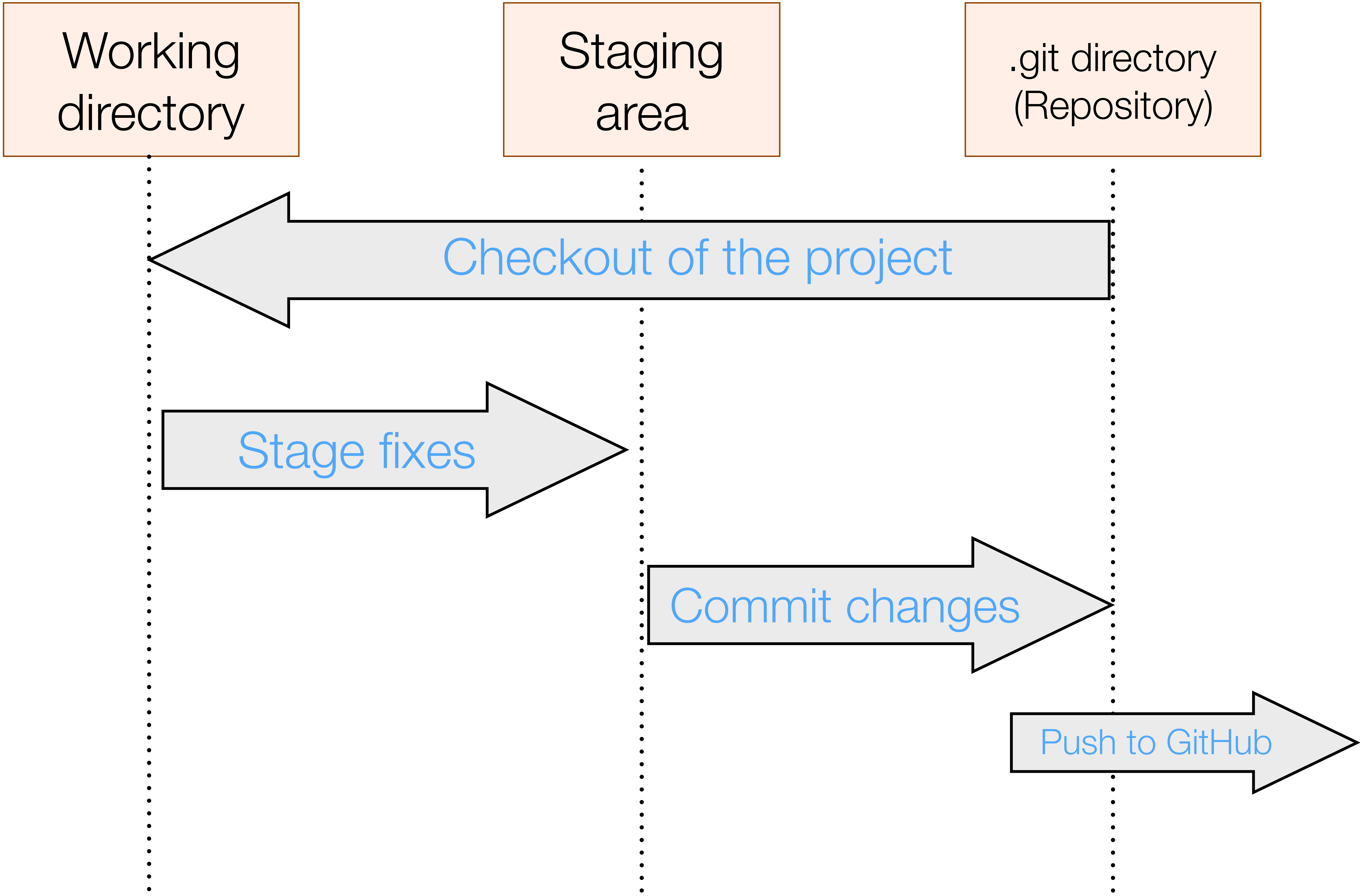
Staging, committing & pushing



Staging, committing & pushing



Staging, committing & pushing



Making a local repository - still!

Add the files

```
> git add README
```

Making a local repository - still!

Add the files

```
> git add README
```

Commit the changes

```
> git commit -m “First commit”
```

Making a local repository - still!

Add the files

```
> git add README
```

Comments! Important!

Commit the changes

```
> git commit -m "First commit"
```

Using git - checking out a project

You need to know the address of the project:

```
https://github.com/jbrinchmann/MyRepo1.git
```

Then you check it out:

```
git clone https://github.com/jbrinchmann/MyRepo1.git
```

You now have this repository in the MyRepo1 directory

To get a new version (if someone has changed something):

```
git pull
```

git - workflow with GitHub

Make GitHub repository

`https://github.com/jbrinchmann`

Check it out

`git clone <address>`

Edit a file on your computer

add the file if new

`git add square.py`

Commit your changes

`git commit -m “Fixed exponent bug”`

Push the changes to the repository

`git push`

Another reason for using it:

You need it for the course!

For you to do:

Check the repository at
https://github.com/jbrinchmann/DDM2017

Here you will find a small document with math/ statistics reminders which I expect you to have read!

Problem sets will also be here (although I will also post them to Blackboard).

SQL - Structured Query Language

What we are doing next - and where it fits in

- The aim is always the data - today we will look a bit at how we can use a database to handle data.
- We will see how we can
  - Create tables in a data-base
  - Find data in a table
  - Combine tables
- There will be some technical detail today - this needs to be learned, but you should always keep in mind what your goal is: To do science with the data.

The problem of creating databases

Keeping track of my observing

Observations:

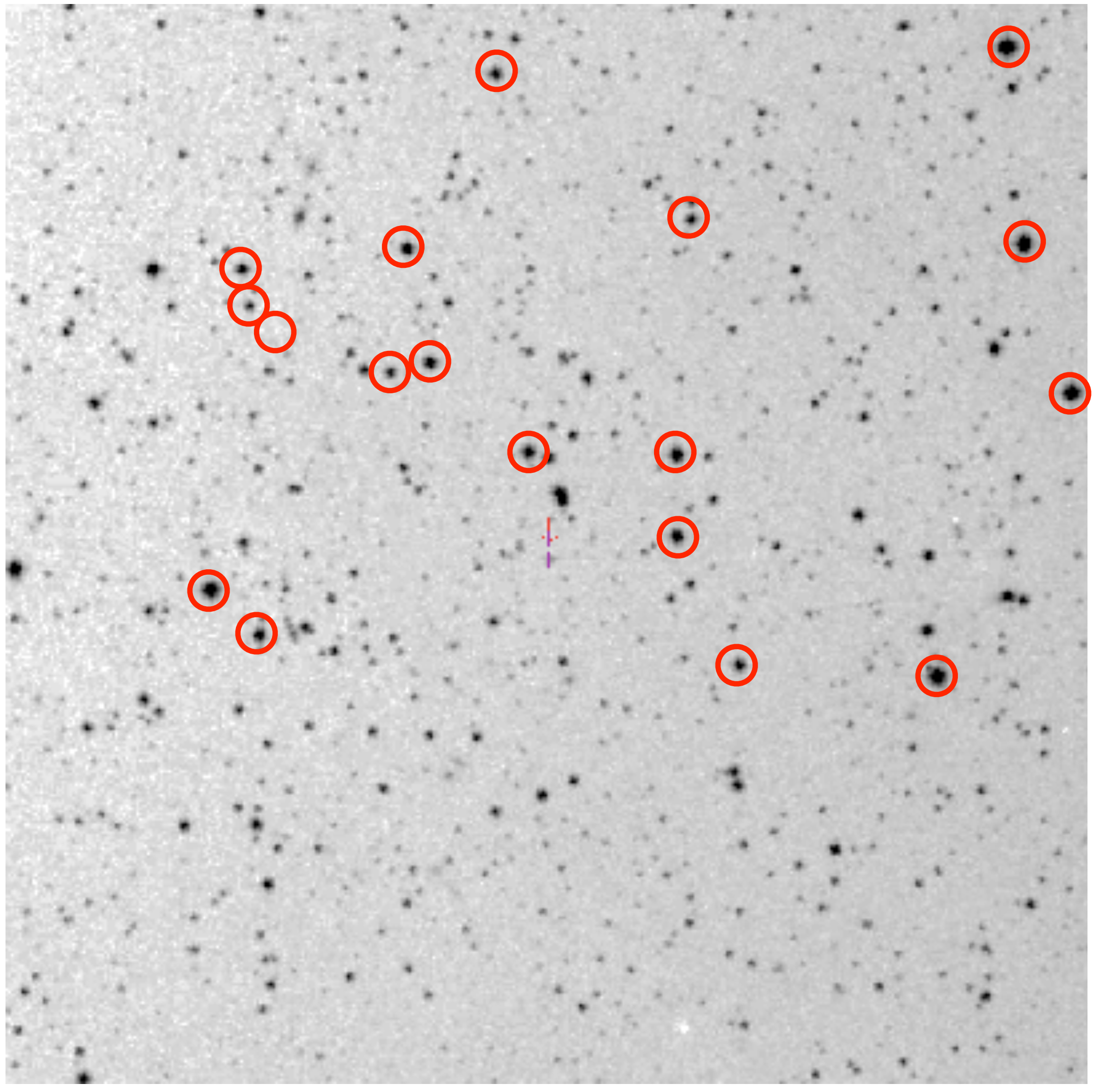
```
# Field       Date           Exptime   Quality   WhereStored
StF-043   92.9885764     23.2        1         /disks/yaeps-1/StF-043.fits
StF-044   97.3323764     30.2        1         /disks/yaeps-1/StF-044.fits
StF-045   93.5532134     29.5        0.5       /disks/yaeps-1/StF-045.fits
```

The problem of creating databases

Keeping track of my observing

Observations:

```
# Field        Date            Exptime    Quality    WhereStored
StF-043   92.9885764      23.2        1          /disks/yaeps-1/StF-043.fits
StF-044   97.3323764      30.2        1          /disks/yaeps-1/StF-044.fits
StF-045   93.5532134      29.5        0.5        /disks/yaeps-1/StF-045.fits
```



Within each field we will detect a number of stars.

The problem of creating databases

Keeping track of my observations:

Observations:

```
# Field       Date          Exptime   Quality   WhereStored
StF-043   92.9885764     23.2        1         /disks/yaeps-1/StF-043.fits
StF-044   97.3323764     30.2        1         /disks/yaeps-1/StF-044.fits
StF-045   93.5532134     29.5        0.5       /disks/yaeps-1/StF-045.fits
```

Stars:

```
# Star      Ra          Dec        g       r
 S1   198.8475000 10.5034722 14.5   15.2
 S2   198.5654167 11.0231944 15.3   15.4
 S5   198.9370833  9.9168889 16.4   15.8
 S7   199.2516667 10.3486944 14.6   14.1
```

If, for each star I keep information about the observations, I can waste a LOT of space. => Relational databases.

Relational databases - an aside

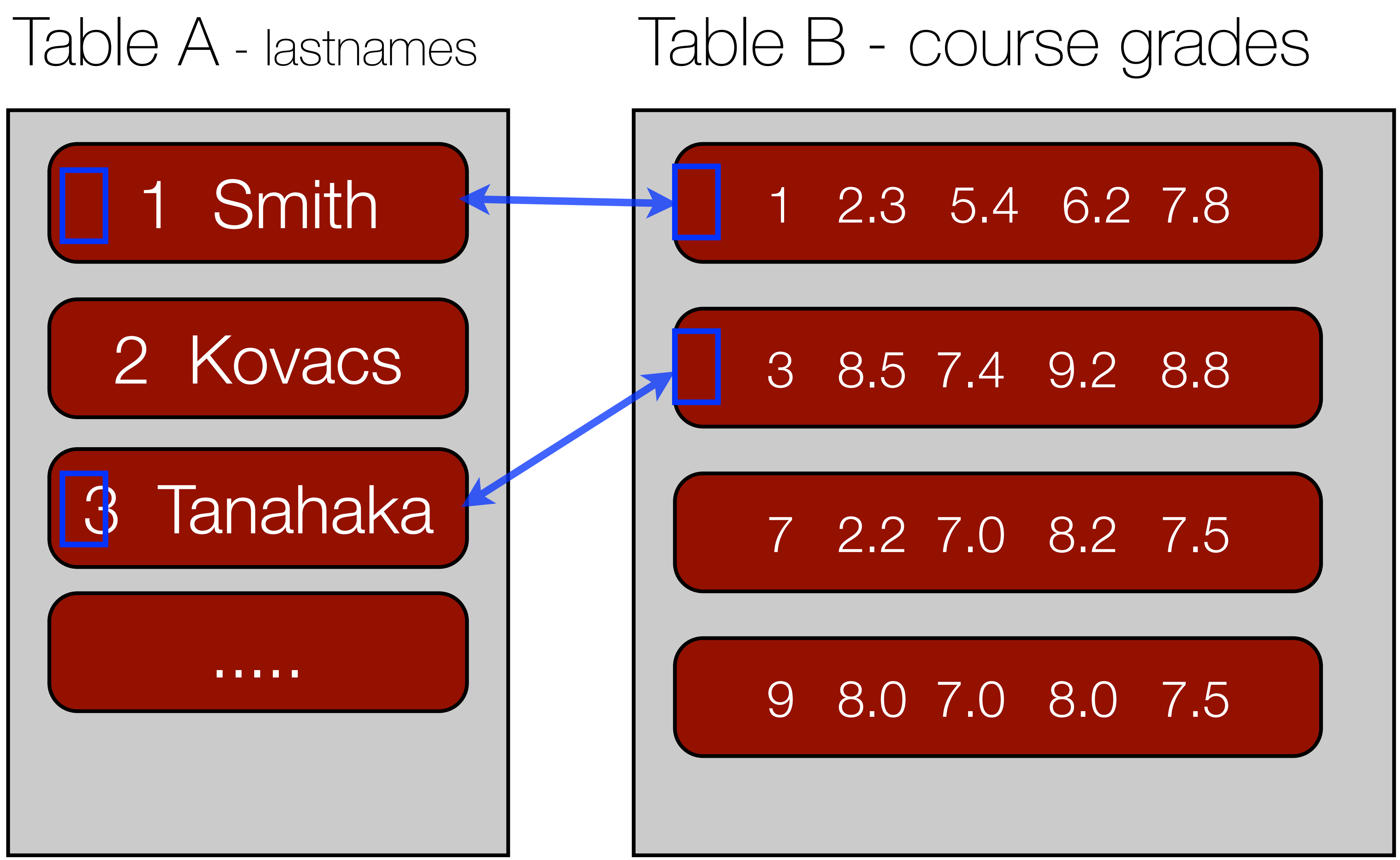
Table A - lastnames

1	Smith
2	Kovacs
3	Tanahaka
.....	

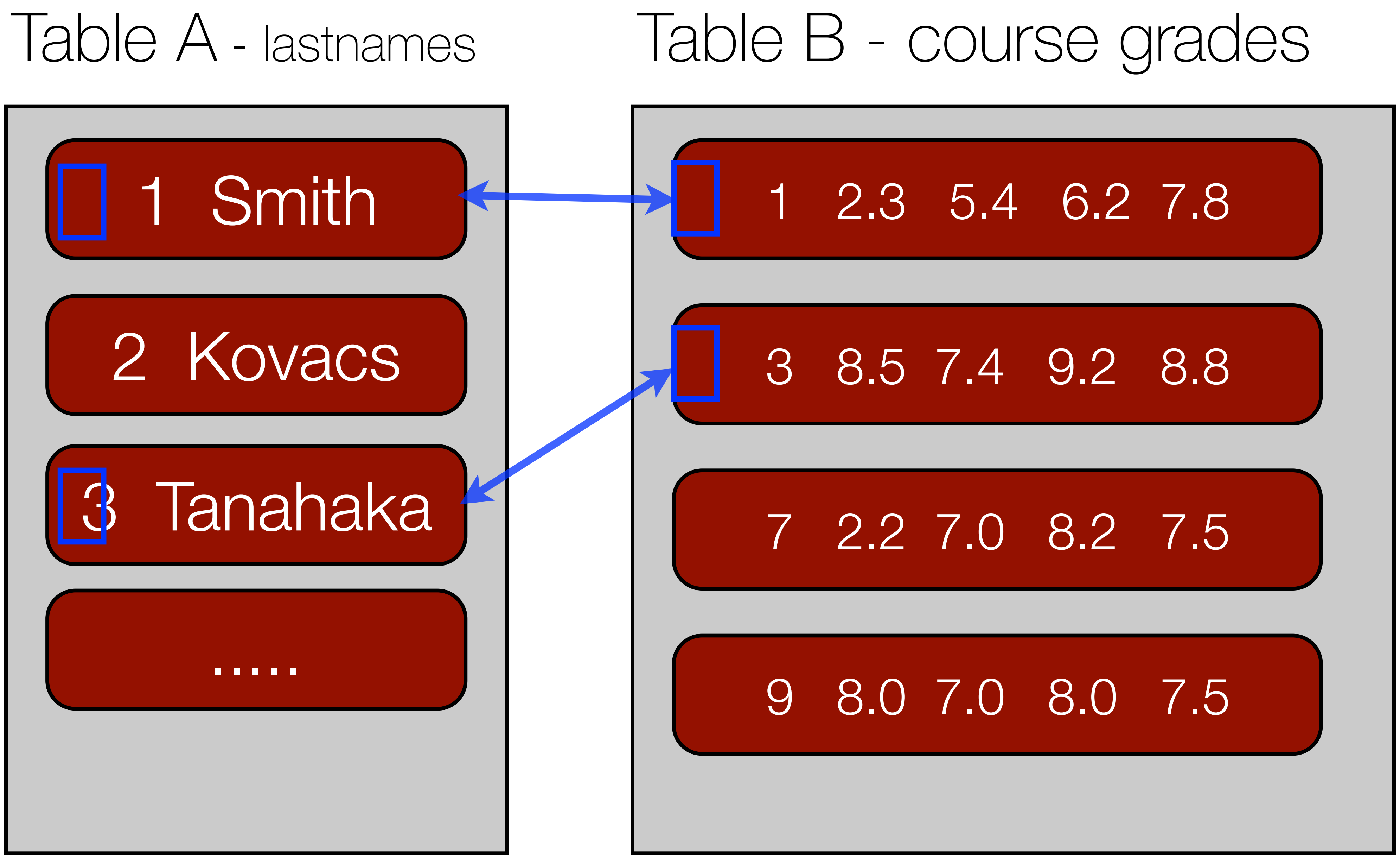
Table B - course grades

1	2.3	5.4	6.2	7.8
3	8.5	7.4	9.2	8.8
7	2.2	7.0	8.2	7.5
9	8.0	7.0	8.0	7.5

Relational databases - an aside



Relational databases - an aside



While a relational database is formally defined with no reference to tables, it is useful to think of it as a collection of tables where (some) rows can be related.

Relational databases - the observing example

Table: Observations

# Field	Date	Exptime	Quality	WhereStored
StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

Table: Stars

# Star	Ra	Dec	g	r
S1	198.8475000	10.5034722	14.5	15.2
S2	198.5654167	11.0231944	15.3	15.4
S5	198.9370833	9.9168889	16.4	15.8
S7	199.2516667	10.3486944	14.6	14.1

How should we link these?

One possibility: Create an ID column

Relational databases - the observing example

Table: Observations

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

Table: Stars

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Relational databases - the observing example

Table: Observations

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

Table: Stars

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Note that there are two IDs in the Stars table:

The ID of each star (StarID) **Primary key**

The ID of the field it was observed id (FieldID) **Foreign key**

Relational databases - the observing example

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

Table Observations

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Table Stars

We would like to ask questions like:

1. Give me all stars brighter than r=14.5
2. How many stars have 0.1 < g-r < 0.4?
3. When did we observe S2?
4. Where is the FITS image stored for star S5?
5. Give me a list of all stars observed on the same FieldID

Creation of databases

We will work with databases in the next practical.

At the end of the lecture notes the details of **creation** of databases are provided. Read through this before the practical but I will not go through in detail today!

Practicalities:
creating a table

We will work with databases in the next practical.

At the end of the lecture notes the details of **creation** of databases are provided. Read through this before the practical but I will not go through in *detail* today!

What is SQL

- It is a declarative language to create database tables, and inserting/updating, deleting and querying information from these tables. It is often said to have two constituent parts:

- Data definition language (DDL): Define database structure (through schemas) and manage data access.

- Data manipulation language (DML): creation, deletion, updating and querying of content.

Choice of database solution

We will make use of **sqlite** as our database solution

Lightweight & convenient

Advantages: Light-weight, no need for complex setup, supports most of SQL. Easy to use for local work. Very widely used (e.g. Firefox, Chrome) and bindings for many languages.

Disadvantages: Not a client-server solution. Not all of SQL is supported and some features (e.g. ALTER TABLE) are only partially available.

Alternatives: MySQL (see Blackboard for notes), Oracle, PostgreSQL, Microsoft SQL Server.

Outline of creation of databases

- Determine the format for each table in your database - its *schema*. Insert this to create your table.

```
CREATE TABLE IF NOT EXISTS Stars (<schema>);
```

- Import data into each table.

```
.separator ,
.import YAEPS.stars-table-sqlite.dat Stars
```

The devil is in the details!

Querying databases - SQL

1. Give me all stars brighter than r=14.5

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

In SQL we do:

```
SELECT *
FROM Stars
WHERE r < 14.5
```

In sqlite:

```
sqlite> SELECT * FROM Stars WHERE r < 14.5;
4,2,S7,199.2516667,10.3486944,14.6,14.1
```

not the prettiest formatting (mysql is nicer) but good enough.

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

```
SELECT *
```

Return all columns for all the rows that matches the constraints. We can also specify specific columns.

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

```
SELECT *
```

Return all columns for all the rows that matches the constraints. We can also specify specific columns.

```
FROM Stars
```

Use the table named Stars for this query.

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

```
SELECT *
```

Return all columns for all the rows that matches the constraints. We can also specify specific columns.

```
FROM Stars
```

Use the table named Stars for this query.

```
WHERE r < 14.5
```

Only return those **rows** that satisfy the criteri(on/a) that we specify.

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r
1	1	S1	198.8475	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

```
SELECT Star, g, r
FROM Stars
```

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r
1	1	S1	198.8475	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

```
SELECT Star, g, r
FROM Stars
```

```
sqlite> SELECT Star, g, r FROM Stars;
S1,14.5,15.2
S2,15.3,15.4
S5,16.4,15.8
S7,14.6,14.1
```

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r
1	1	S1	198.8475	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

```
SELECT Star, g, r
FROM Stars
WHERE r < 14.5
```

But what if I want some combination of columns?

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r	g-r
1	1	S1	198.8475	10.5034722	14.5	15.2	-0.7
1	2	S2	198.5654167	11.0231944	15.3	15.4	-0.1
3	3	S5	198.9370833	9.9168889	16.4	15.8	0.6
2	4	S7	199.2516667	10.3486944	14.6	14.1	0.5

```
SELECT Star, g, r, g-r as gr
FROM Stars
WHERE FieldID = 1
```

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r	g-r
1	1	S1	198.8475	10.5034722	14.5	15.2	-0.7
1	2	S2	198.5654167	11.0231944	15.3	15.4	-0.1
3	3	S5	198.9370833	9.9168889	16.4	15.8	0.6
2	4	S7	199.2516667	10.3486944	14.6	14.1	0.5

```
SELECT Star, g, r, g-r as gr
FROM Stars
WHERE FieldID = 1
```

```
sqlite> SELECT Star, g, r, g-r as gr FROM Stars WHERE FieldID
= 1;
S1,14.5,15.2,-0.6999999999999999
S2,15.3,15.4,-0.0999999999999996
```

Getting a few values - SQL flavours...

When you have very large tables, you often want to try out statements or just get few examples back. This is easy but depends on the SQL flavour you use:

Microsoft SQL (used in SDSS):

```
SELECT TOP 2 r FROM STARS
```

MySQL and sqlite:

```
SELECT r FROM STARS LIMIT 2
```

Oracle (used in AstroWISE):

```
SELECT r FROM STARS WHERE ROWNUM < 2
```

Recall:

Table Observations

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

Table Stars

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Sorting in SQL - ORDER BY

Sorting your output on some variable is simple:

```
SELECT Field, Quality
FROM Observations
ORDER BY Quality
```

What is the first field?

Sorting in SQL - ORDER BY

Sorting your output on some variable is simple:

```
SELECT Field, Quality
FROM Observations
ORDER BY Quality
```

What is the first field?

```
sqlite> SELECT Field, Quality FROM Observations ORDER BY Quality;
StF-051,0.0
StF-045,0.5
StF-048,0.7
StF-043,1.0
StF-044,1.0
StF-046,1.0
StF-047,1.0
StF-049,1.0
StF-050,1.0
```

When you don't need everything

You want to make a histogram of the magnitude distribution of stars in the SDSS. There are **260,562,744** stars - do you need to download them all?

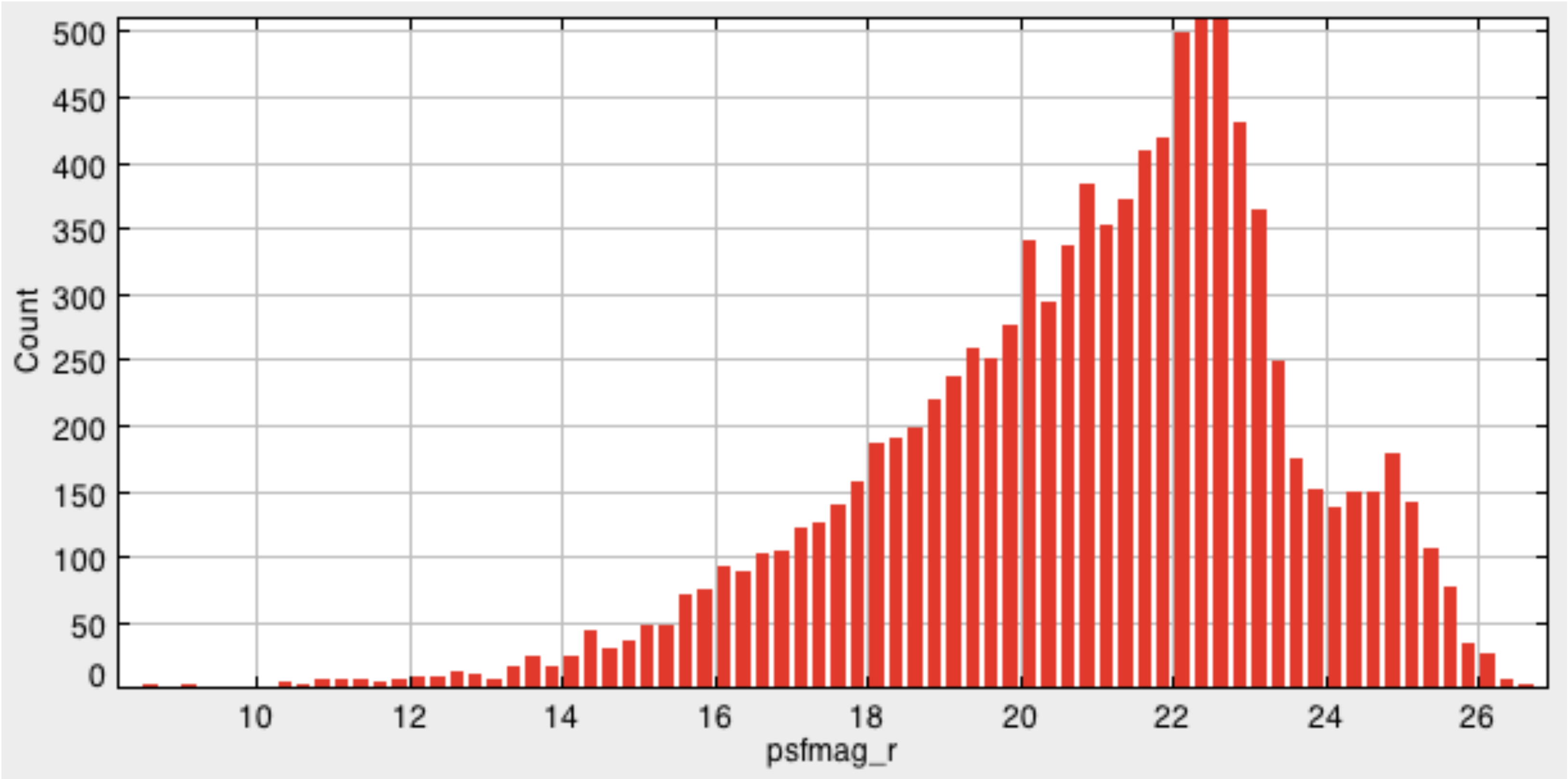
No.

When you don't need everything

You want to make a histogram of the magnitude distribution of stars in the SDSS. There are **260,562,744** stars - do you need to download them all?

No.

But you can if you want - here are the first 10,000. It can take a lot of time though!



Grouping your output

A better solution is sometimes to let the database server do the work. To do that we need to group our output.

Let us look at a simple case - we want to count the number of stars per field.

FieldI	StarI	Star	Ra	Decl	g	r	
1	1	S1	198.8475	0.503472	14.5	15.2	2
1	2	S2	98.565416	1.023194	15.3	15.4	
3	3	S5	98.93708	39.916888	16.4	15.8	1
2	4	S7	99.251666	0.348694	14.6	14.1	1

Grouping your output

Counting the number of stars per field.

FieldID	StarID	Star	Ra	Decl	g	r	
1	1	S1	198.8475	0.503472	14.5	15.2	2
1	2	S2	98.565416	1.023194	15.3	15.4	
3	3	S5	98.937083	9.916888	16.4	15.8	1
2	4	S7	99.251666	0.348694	14.6	14.1	1

The statement we need is: GROUP BY

```
SELECT FieldID, COUNT(*) as NperField
FROM Stars
GROUP BY FieldID
```

Accumulative functions - a side-step

SQL has certain functions that are ‘accumulative’:

COUNT
SUM
AVG
MIN
MAX

```
SELECT COUNT(*) as Nstars
FROM Stars
```

FieldI	StarI	Star	Ra	Decl	g	r
1	1	S1	198.8475	0.503472	14.5	15.2
1	2	S2	98.565416	1.023194	15.3	15.4
3	3	S5	98.937083	9.916888	16.4	15.8
2	4	S7	99.251666	0.348694	14.6	14.1

GROUP BY - operating on accumulative functions

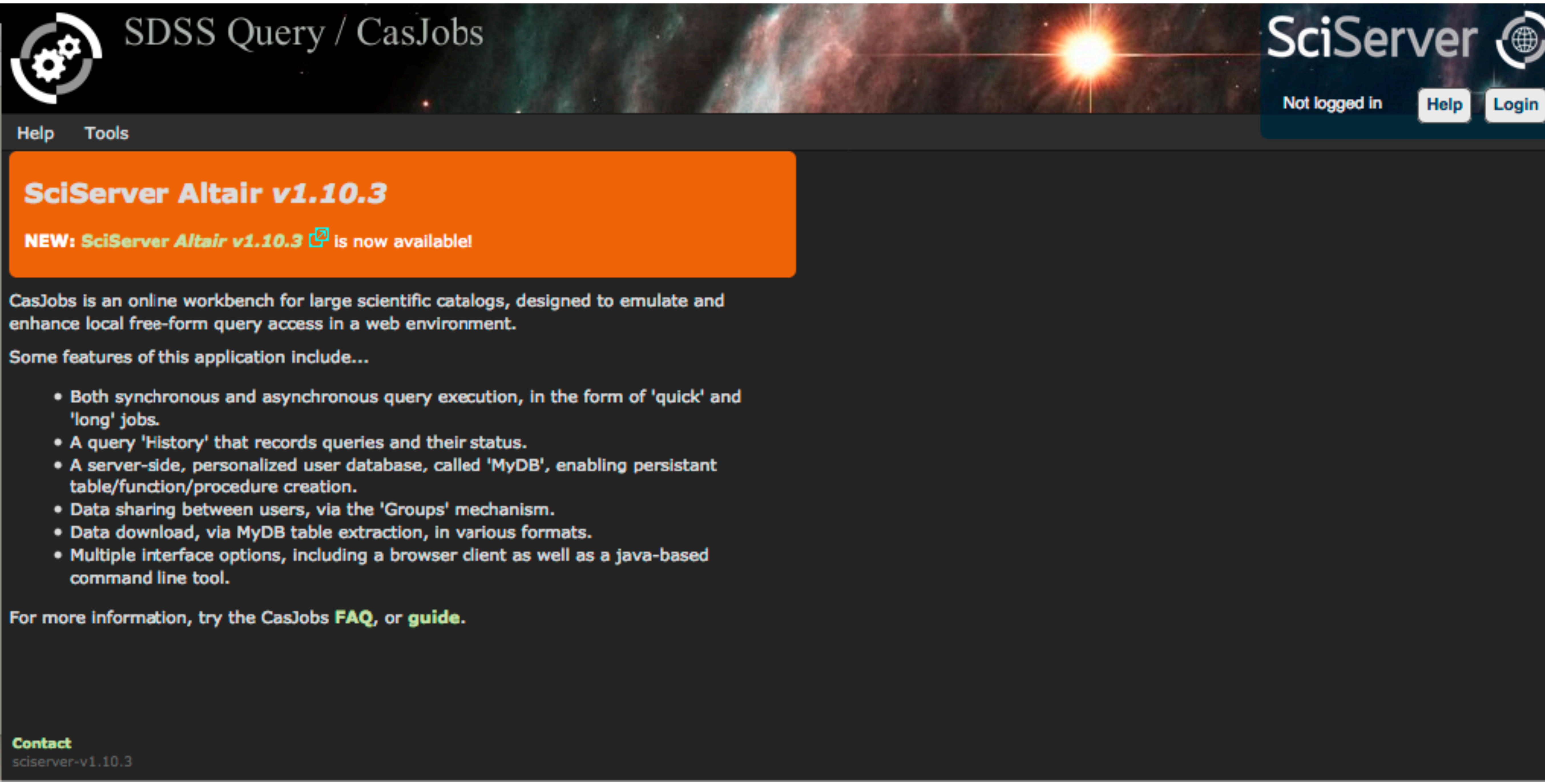
On their own these functions work on everything in one bunch - often not what you want. This is the role of **GROUP BY**.

```
select fieldID, avg(r) as 'mean r'
from stars
group by FieldID;
```

Big data easily: SDSS & SQL

The SDSS database

http://skyserver.sdss.org/casjobs/default.aspx



Create an account here when you can!

Ordering & Groupings

Counting objects in bins:

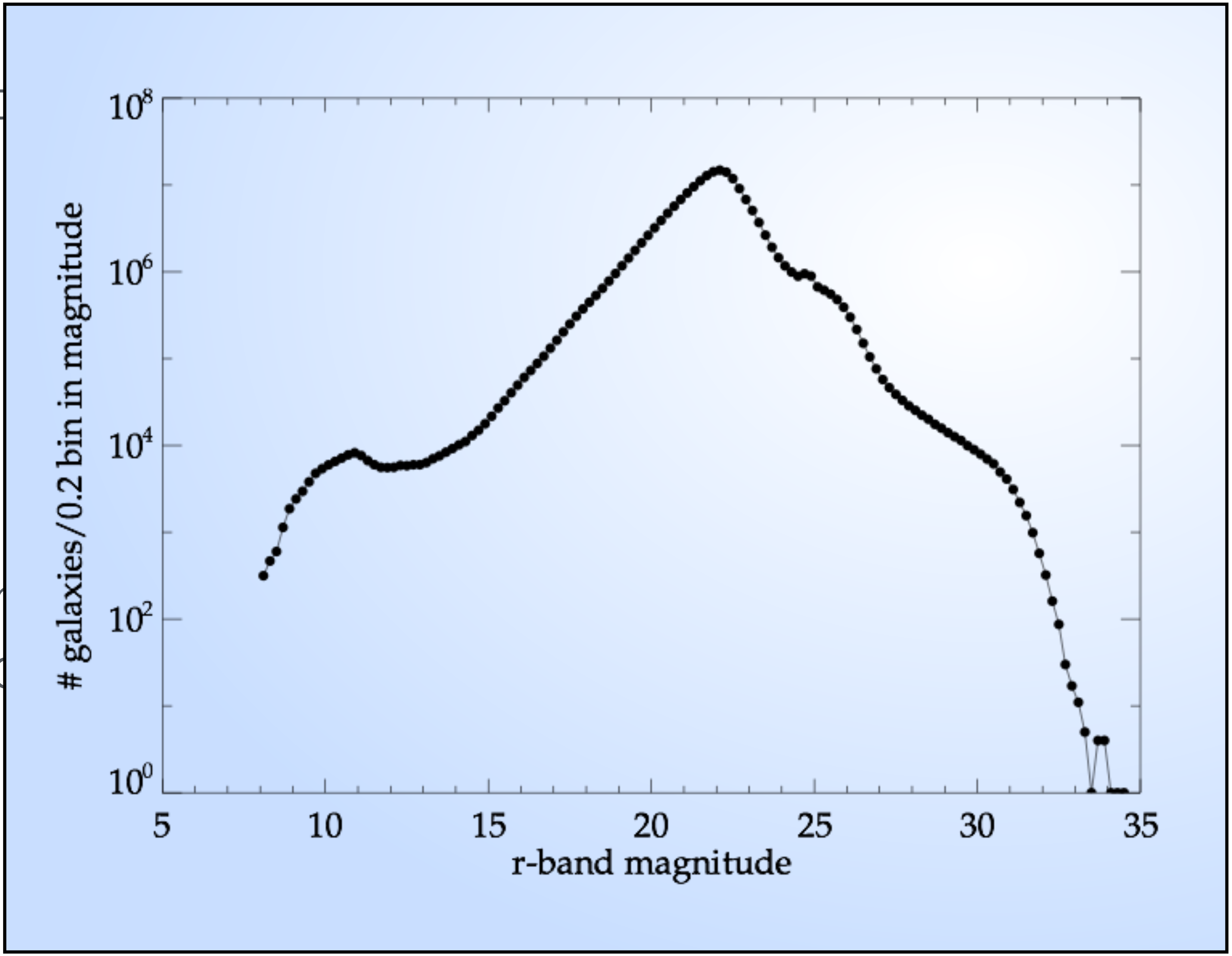
```
SELECT .2*(.5+floor(g.r/.2)) as mag,
       count(*) as num
FROM GALAXY as g
GROUP BY .2*(.5+floor(g.r/.2))
ORDER BY mag
```

In general if you want to make C bins per unit for variable x, you use:

$$\frac{1}{C}\left(0.5 + \lfloor Cx \rfloor\right)$$

Ordering & Groupings

Count



In ger
unit fo

Creating intermediate tables

The result of a SELECT query is another table - we can therefore use this into another query.

As an example we can use this to create a histogram of the first 10,000 stars in the SDSS Star table:

```
select FLOOR(psfMag_r*10+0.5)/10
as rmag, COUNT(*) as N
  FROM
    (select TOP 10000 psfMag_r
    FROM Star) smallcat
GROUP BY (FLOOR(psfMag_r*10+0.5))
ORDER BY rmag
```

Selects from

Now let us go back to our questions:

3. When did we observe S2?

4. Where is the FITS image stored for star S5?

5. Give me a list of all stars observed on the same FieldID

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

In these cases we need to be able to link information between two tables. In SQL we do this using JOINs

First a theoretical view:

Two sets of values: $\{x_i\}$ $\{y_j\}$ (the elements can be vectors/matrices etc)

Possible ways to combine:

Union: $\{x_i, y_j | i=1, n; j=1, m\}$ elements must be the same

Cross-join: $\{(x_i, y_j) | i=1, n; j=1, m\}$ ie. all possible pairs

Left Outer join: $\{(x_i, y_i)$ if y_i exists, (x_i, NULL) otherwise$\}$

Right Outer join: $\{(x_i, y_i)$ if x_i exists, (NULL, y_i) otherwise$\}$

Inner join: $\{(x_i, y_i)$ if y_i exists$\}$

All these are supported in SQL.

UNION

It must make sense to glue the tables together!

```
Select TOP 10
ra, dec
FROM SpecPhoto
WHERE ra > 120
AND DEC < 0
```

Table 1

```
UNION
```

```
Select TOP 10
ra, dec
From SpecPhoto
WHERE ra < 10
AND DEC > 0
```

Table 2

Try it in SDSS!

UNION

It must make sense to glue the tables together!

```
Select TOP 10
ra, dec
FROM SpecPhoto
WHERE ra > 120
AND DEC < 0
```

Table 1

Ra	Dec

```
UNION
```

```
Select TOP 10
ra, dec
From SpecPhoto
WHERE ra < 10
AND DEC > 0
```

Table 2

Try it in SDSS!

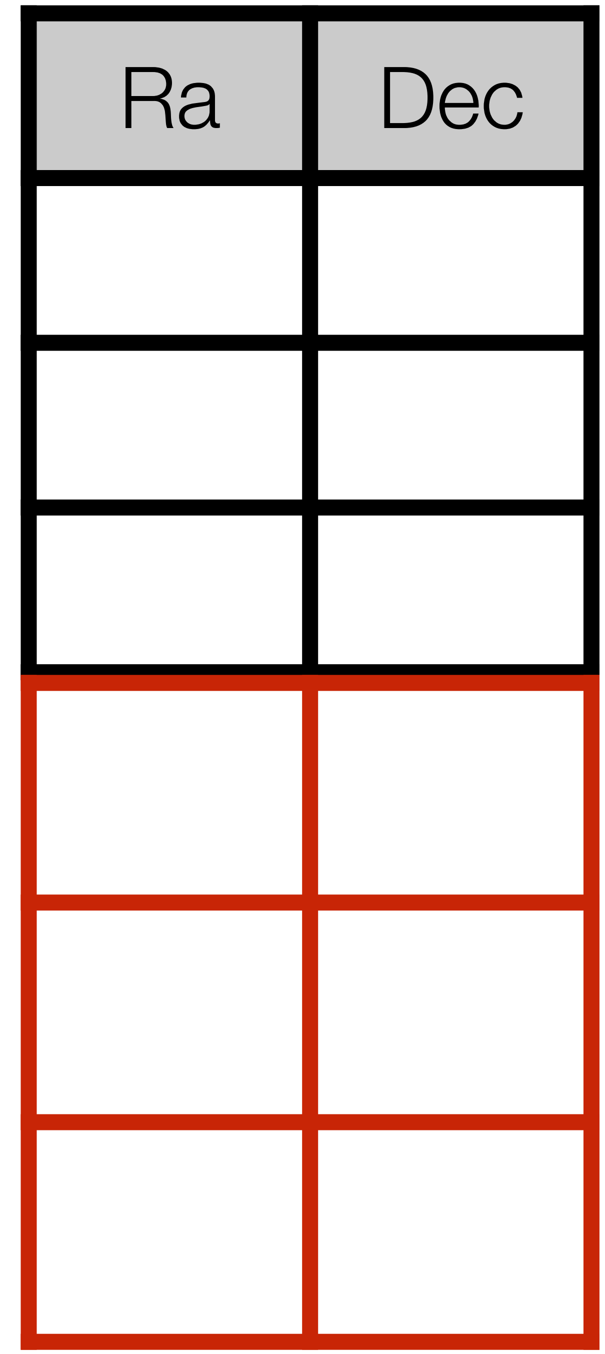
UNION

It must make sense to glue the tables together!

```
Select TOP 10
ra, dec
FROM SpecPhoto
WHERE ra > 120
AND DEC < 0

UNION

Select TOP 10
ra, dec
From SpecPhoto
WHERE ra < 10
AND DEC > 0
```

Table 1

Table 2



Try it in SDSS!

How do you combine?

ID	1
u	19.3
ID	2
u	17.5
ID	3
u	20.5

ID	1
EW(	75Å
ID	3
EW(	0.5Å

How do you combine?

ID	1
u	19.3
ID	2
u	17.5
ID	3
u	20.5

ID	1
EW(	75Å
ID	3
EW(	0.5Å

JOIN

ID	u	EW(Ha)
1	19.3	75Å
3	20.5	0.5Å

How do you combine?

ID	1
u	19.3
ID	2
u	17.5
ID	3
u	20.5

ID	1
EW(	75Å
ID	3
EW(	0.5Å

JOIN

ID	u	EW(Ha)
1	19.3	75Å
3	20.5	0.5Å

```
SELECT P.u, S.z
FROM Photo as P
JOIN  Spectro as S
ON P.ID=S.ID
```

How do you combine?

ID	1
u	19.3
ID	2
u	17.5
ID	3
u	20.5

ID	1
EW(	75Å
ID	3
EW(	0.5Å

JOIN

ID	u	EW(Ha)
1	19.3	75Å
3	20.5	0.5Å

```
SELECT P.u, S.z
FROM Photo as P
JOIN  Spectro as S
ON P.ID=S.ID
```

OR

```
SELECT P.u, S.z
FROM Photo as P,
Spectro as S
WHERE P.ID=S.ID
```

How do you combine?

ID	u	EW(Ha)
1	19.3	75Å
3	20.5	0.5Å

```
SELECT P.u, S.z
FROM Photo as P
JOIN  Spectro as S
ON P.ID=S.ID
```

OR

```
SELECT P.u, S.z
FROM Photo as P,
Spectro as S
WHERE P.ID=S.ID
```

Explicit INNER JOIN

Implicit INNER JOIN

(or *old-style* INNER JOIN)

Explicit vs implicit JOINs

JOIN ... ON a=b
or
WHERE a = b

Mostly up to you - there should be no significant difference between the two.

The main disadvantage of an implicit JOIN is that you have less control of the order things are done if you have more than two tables.

I personally prefer explicit JOINs because they show more clearly what your intention is and if you have a problem because your query runs too slowly, you can more easily figure out the execution order.

ID	u	EW(Ha)
1	19.3	75Å
2	17.5	NULL
3	20.5	0.5Å

But if we want to keep **all** possible pairs we need an **OUTER JOIN**

ID	u	EW(Ha)
1	19.3	75Å
2	17.5	NULL
3	20.5	0.5Å

But if we want to keep **all** possible pairs we need an **OUTER JOIN**

```
SELECT P.u, S.z
FROM Photo as P
LEFT OUTER JOIN  Spectro as S
ON P.ID=S.ID
```

ID	u	EW(Ha)
1	19.3	75Å
2	17.5	NULL
3	20.5	0.5Å

Returning to our questions:

3. When did we observe S2?

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Our link is FieldID in Stars to ID in Observations

Returning to our questions:

3. When did we observe S2?

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Our link is FieldID in Stars to ID in Observations

```
select s.Star, o.Field, o.Date
from
   stars as s
   JOIN Observations as o
   ON s.fieldID = o.ID
Where Star = ‘S2’
```

Returning to our questions:

3. When did we observe S2?

```
select s.Star, o.Field, o.Date
from
  stars as s
  JOIN Observations as o
  ON s.fieldID = o.ID
Where Star = 'S2'
```

We must specify what table to get a quantity from.

JOIN the tables explicitly

Choose our star

Useful: We can do **multiple** stars by changing the WHERE statement to:

Returning to our questions:

3. When did we observe S2?

```
select s.Star, o.Field, o.Date
from
  stars as s
  JOIN Observations as o
  ON s.fieldID = o.ID
Where Star = ‘S2’
```

We must specify what table to get a quantity from.

JOIN the tables explicitly

Choose our star

Useful: We can do **multiple** stars by changing the WHERE statement to:

```
Where Star IN (‘S2’, ‘S1’)
```

The next steps

Read through the final part of the lecture notes & start trying out sqlite3. In the practical we will try out several of these commands on tables you create yourself & also using SDSS.

Creation of databases & using them from Python

Creating a sqlite database

Let us set up a simple database to start with:

```
> sqlite3 DDM15.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .tables
sqlite> .exit
```

This should give you nothing because there are no tables. We need to make one.

Step 2 - inserting a table

Get: YAEPS.stars-table-sqlite.dat and sqlite3-make-stars-table.sql from Blackboard. **Edit** the latter to reflect the location of YAEPS.stars-table-sqlite.dat

When that is done:

```
> sqlite3 DDM15.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .read sqlite3-make-stars-table.sql
sqlite> .tables
Stars
```

Magic?

First we need to create a schema

To do this we need to understand our data:

```
# StarID FieldID  Star      Ra           Dec         g        r
     1         1   S1  198.8475000 10.5034722 14.5   15.2
     2         1   S2  198.5654167 11.0231944 15.3   15.4
     3         3   S5  198.9370833  9.9168889 16.4   15.8
     4         2   S7  199.2516667 10.3486944 14.6   14.1
```

Schema

Column	Type	SQL type	Other
StarID	Integer	INT	PrimaryKey, Unique
FieldID	Integer	INT	ForeignKey
Star	String	varchar(10)	Length < 10
Ra	Real number	DOUBLE	
Dec	Real number	DOUBLE	
g	Real number	DOUBLE	
r	Real number	DOUBLE	

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
    StarID INT,
    FieldID INT,
    Star varchar(10),
    ra DOUBLE,
    decl DOUBLE,
    g FLOAT,
    r FLOAT,
    UNIQUE(StarID),
    PRIMARY KEY(StarID),
    FOREIGN KEY(FieldID) REFERENCES Observations(ID)
    );
```

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
    StarID INT,
    FieldID INT,
    Star varchar(10),
    ra DOUBLE,
    decl DOUBLE,
    g FLOAT,
    r FLOAT,
    UNIQUE(StarID),
    PRIMARY KEY(StarID),
    FOREIGN KEY(FieldID) REFERENCES Observations(ID)
    );
```

Notice the definition of strings

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
    StarID INT,
    FieldID INT,
    Star varchar(10),
    ra DOUBLE,
    decl DOUBLE,
    g FLOAT,
    r FLOAT,
    UNIQUE(StarID),
    PRIMARY KEY(StarID),
    FOREIGN KEY(FieldID) REFERENCES Observations(ID)
    );
```

This indicates columns where each row has to have a unique value

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
    StarID INT,
    FieldID INT,
    Star varchar(10),
    ra DOUBLE,
    decl DOUBLE,
    g FLOAT,
    r FLOAT,
    UNIQUE(StarID),
    PRIMARY KEY(StarID),
    FOREIGN KEY(FieldID) REFERENCES Observations(ID)
    );
```

This can optionally be used to indicate the primary key in this database

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
    StarID INT,
    FieldID INT,
    Star varchar(10),
    ra DOUBLE,
    decl DOUBLE,
    g FLOAT,
    r FLOAT,
    UNIQUE(StarID),
    PRIMARY KEY(StarID),
    FOREIGN KEY(FieldID) REFERENCES Observations(ID)
    );
```

Optional: Use this to indicate foreign keys in the table. This can be useful for clarity at least.

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

well, we need some more (this is database specific:)

```
.separator ,
.import YAEPS.stars-table-sqlite.dat Stars
```

These quick import routines are not part of SQL so vary from database to database (but they are handy!)

Creating databases and tables

well, we need some more (this is database specific:)

```
.separator ,
.import YAEPS.stars-table-sqlite.dat Stars
```

These quick import routines are not part of SQL so vary from database to database (but they are handy!)

In MySQL for instance it would be:

```
LOAD DATA INFILE 'YAEPS.stars-table-sqlite.dat' INTO
TABLE Stars
  FIELDS TERMINATED BY ‘,’;
```

here I can also add IGNORE 1 LINES at the end if I want to skip a header line.

Getting rid of a table & altering it

Everyone makes mistakes. Sometimes the best is to just get rid of a table. It is easy:

```
DROP TABLE Galaxy;
```

Getting rid of a table & altering it

Everyone makes mistakes. Sometimes the best is to just get rid of a table. It is easy:

```
DROP TABLE Galaxy;
```

It is also possible to modify (alter) a table - but not that this does not fully work in sqlite!

```
ALTER TABLE Galaxy ADD rmag  FLOAT
```

Adding column

```
ALTER TABLE Galaxy DROP COLUMN rmag
```

Deleting column

```
ALTER TABLE Galaxy ALTER rmag DOUBLE
```

Changing the type of a column

Putting data into the database - row by row

Adding data one row at a time is done using INSERT:

```
INSERT INTO Galaxy VALUES (1,
12.334, 14.433);
```

GalaxyI ra decl

GalaxyI	ra	decl
1	12.334	14.433

You can also insert only some values but then you have to say what columns they are for:

```
INSERT INTO Galaxy (GalaxyID,
decl) VALUES (2, 17.5);
```

GalaxyI	ra	decl
1	12.334	14.433
2	NULL	17.5

Note: You can also insert multiple rows if you copy from one table to another.

Putting data into the database - in one go

INSERT is quite slow, in part because the database is reorganised after each insert. This is fine for small jobs but not for 100,000s of entries. For this case we use LOAD DATA.

```
LOAD DATA INFILE <filename> INTO TABLE Stars
  FIELDS TERMINATED BY ',' IGNORE 1 LINES;
```

(optional)

will load from a file where each column is separated by a comma (the default is TAB), and rows by newline but it will skip the first line. The column types must match the Table definition - so in this case a file that can be loaded would be:

```
# StarID FieldID Star Ra Dec     g    r
1,1,S1,198.8475000,10.5034722,14.5,15.2
2,1,S2,198.5654167,11.0231944,15.3,15.4
```

This is fine for those situations where your data are already known - for instance if you downloaded a catalogue.

Updating data

Most astronomical data can change (calibration files can be updated, measurement techniques improved etc.)

We often then want to create a new table, but sometimes you want to update a row instead. This is done in SQL using the UPDATE command.

```
UPDATE Galaxy SET ra=11.3 WHERE decl=17.5
```

This is ok, in particular where information is acquired after most of the table is assembled.

Next: A more fancy criterion

2. How many stars have $0.1 < g\text{-}r < 0.4$?

```
SELECT *
FROM Stars
WHERE g-r BETWEEN 0.1 AND 0.4
```

or

```
SELECT *
FROM Stars
WHERE g-r > 0.1
  AND g-r < 0.4
```

Try this now for your newly created table - remember to put a ; at the end of each line!

Reminder:

```
# StarID FieldID  Star      Ra            Dec         g        r
     1         1    S1   198.8475000 10.5034722 14.5    15.2
     2         1    S2   198.5654167 11.0231944 15.3    15.4
     3         3    S5   198.9370833  9.9168889 16.4    15.8
     4         2    S7   199.2516667 10.3486944 14.6    14.1
```

Schema

Column	Type	SQL type	Other
StarID	Integer	INT	PrimaryKey, Unique
FieldID	Integer	INT	ForeignKey
Star	String	varchar(10)	Length < 10
Ra	Real number	DOUBLE	
Dec	Real number	DOUBLE	
g	Real number	DOUBLE	
r	Real number	DOUBLE	

Create a new table - now for the Observations

```
# FieldID  Field       Date       Exptime   Quality         WhereStored
   1       StF-043  92.9885764     23.2       1        /disks/yaeps-1/StF-043.fits
   2       StF-044  97.3323764     30.2       1        /disks/yaeps-1/StF-044.fits
   3       StF-045  93.5532134     29.5       0.5      /disks/yaeps-1/StF-045.fits
```

Column	Type	SQL type	Other
FieldID			
Field			
Date			
Exptime			
Quality			
WhereStored			

In practice:

Get YAEPS.observations-table-sqlite.dat and sqlite3-make-observations-table.sql from Blackboard. **Edit** the latter to reflect the location of YAEPS.observations-table-sqlite.dat

```
> sqlite3 DDM15.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .read sqlite3-make-observations-table.sql
sqlite> .tables
Observations  Stars
```

sqlite from Python

or - what you really want to know

sqlite3 in python - an example

```
import sqlite3 as lite;
```

Load what is necessary

The database must be created first!

```
con = lite.connect(database)
```

Connect to database

```
with con:



    # Get a cursor.
    cur = con.cursor()

    # Execute commands
    cur.execute(command)
```

Use with to gracefully handle exceptions

cursors are used to navigate relational databases and are often needed in programatic access

Building the table in python:

```
# Next, we create a connection to the database.
con = lite.connect(database)


with con:

    # Get a cursor.
    cur = con.cursor()

    # Create the command to create the table. I use a
    # multiline string to ease readability here.
    command = """CREATE TABLE IF NOT EXISTS {0} (StarID INT,
             FieldID INT, Star varchar(10), ra DOUBLE,
             decl DOUBLE, g FLOAT, r FLOAT,
             UNIQUE(StarID), PRIMARY KEY(StarID),
             FOREIGN KEY(FieldID) REFERENCES Observations(ID))""".format(table)

    # Next, actually execute this command.
    cur.execute(command)

    # Now that this is working, let us loop over the table entries
    # and insert these into the table.
    for row in cat:
        command = "INSERT INTO Stars VALUES({0},{1},'{2}',{3},{4},{5},
{6})".format(row[0], row[1], row[2], row[3], row[4], row[5], row[6])
        print command
        cur.execute(command)
```

See Blackboard for the script - now build one for the observations

Using python to query the database:

```
In [1]: import sqlite3 as lite;

In [2]: con = lite.connect('DDM15-python.db')

In [3]: rows = con.execute('SELECT ra, decl FROM Stars')

In [4]: for row in rows:
   ...:     print "Ra={0}  Dec={1}".format(row[0], row[1])
   ...:
Ra=198.8475  Dec=10.5034722
Ra=198.5654167  Dec=11.0231944
Ra=198.9370833  Dec=9.9168889
Ra=199.2516667  Dec=10.3486944
```

As should be clear: The execute statements executes SQL statements in the database and returns a list of results.