Agenda

- Overview of databases
- Working with databases
- Brief introduction to SQL
Why?

- Why should a statistician care about databases?
  - Obvious – data is stored in databases
  - Data often too large – cannot analyze all at once, cannot store entirely in memory
How?

- Software
- R – packages for interacting with database
- ‘Native’ database client software
- We will focus on R, but many real situations require a mix of both
- Many other aspects beyond our scope
- db design, db access control
Overview
Database

- Organized collection of data – usually large
- Example uses – financial records, medical records, inventories
- Ubiquitous – even web sites and the music player in your phone are backed by databases
- Most common type – relational database
Relational database

- Consists of one or more **tables** (similar to a data frame in R)
- columns (variables)
- rows (observations)
- Central principle of database design – **normalization** (reduce redundancy)
Example

- Healthcare provider’s database containing information on
  - physicians
  - patients
Two options
patients and physicians

<table>
<thead>
<tr>
<th>pat_last</th>
<th>pat_first</th>
<th>dr_last</th>
<th>dr_first</th>
<th>dr_office</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doe</td>
<td>John</td>
<td>Jekyll</td>
<td>Henry</td>
<td>123 Main</td>
</tr>
<tr>
<td>Brown</td>
<td>Charlie</td>
<td>House</td>
<td>Gregory</td>
<td>123 Main</td>
</tr>
<tr>
<td>Morgan</td>
<td>Dexter</td>
<td>House</td>
<td>Gregory</td>
<td>123 Main</td>
</tr>
<tr>
<td>Dunphy</td>
<td>Claire</td>
<td>Jekyll</td>
<td>Henry</td>
<td>123 Main</td>
</tr>
</tbody>
</table>
patients and physicians

<table>
<thead>
<tr>
<th>pat_last</th>
<th>pat_first</th>
<th>dr_last</th>
<th>dr_first</th>
<th>dr_office</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doe</td>
<td>John</td>
<td>Jekyll</td>
<td>Henry</td>
<td>123 Main</td>
</tr>
<tr>
<td>Brown</td>
<td>Charlie</td>
<td>House</td>
<td>Gregory</td>
<td>123 Main</td>
</tr>
<tr>
<td>Morgan</td>
<td>Dexter</td>
<td>House</td>
<td>Gregory</td>
<td>123 Main</td>
</tr>
<tr>
<td>Dunphy</td>
<td>Claire</td>
<td>Jekyll</td>
<td>Henry</td>
<td>123 Main</td>
</tr>
</tbody>
</table>

Redundant information.
What happens if Dr Jekyll’s office changes?
<table>
<thead>
<tr>
<th>last_name</th>
<th>first_name</th>
<th>physician_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doe</td>
<td>John</td>
<td>34</td>
</tr>
<tr>
<td>Brown</td>
<td>Charlie</td>
<td>55</td>
</tr>
<tr>
<td>Morgan</td>
<td>Dexter</td>
<td>55</td>
</tr>
<tr>
<td>Dunphy</td>
<td>Claire</td>
<td>34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>physician_id</th>
<th>last_name</th>
<th>first_name</th>
<th>office</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>Jekyll</td>
<td>Henry</td>
<td>123 Main</td>
</tr>
<tr>
<td>55</td>
<td>House</td>
<td>Gregory</td>
<td>123 Main</td>
</tr>
</tbody>
</table>
Keys

• Records are tied together by unique identifiers

• In preceding example, each row of the physicians table is uniquely identified by the column “physician_id” – it is a key
Normalization

- Essential goals
- Reduce redundancy
- Decrease dependencies
- Importance for statisticians?
- Transformations often necessary to get all pieces of data required for analysis
Example

• Manufacturer’s database containing information on
  • products
  • parts
  • suppliers
### parts

<table>
<thead>
<tr>
<th>name</th>
<th>cost</th>
<th>part_id</th>
<th>supplier_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>toggle switch</td>
<td>$3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>round button</td>
<td>$2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>gear</td>
<td>$5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>fly wheel</td>
<td>$3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>glass panel</td>
<td>$7</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

### suppliers

<table>
<thead>
<tr>
<th>supplier_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Widgets, Inc.</td>
</tr>
<tr>
<td>2</td>
<td>Gizmo, Ltd.</td>
</tr>
<tr>
<td>3</td>
<td>Clearly</td>
</tr>
</tbody>
</table>

### products

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Foo</td>
<td>1,3</td>
</tr>
<tr>
<td>2</td>
<td>Bar</td>
<td>2,4</td>
</tr>
<tr>
<td>3</td>
<td>iJunk</td>
<td>1,2,3</td>
</tr>
</tbody>
</table>
Example

- Bank’s database containing information on
  - customers
  - accounts
  - transactions
Database server

- Databases are typically accessed via server-client model
- People often say *database*, but mean *database server*
- Popular software – Oracle, MySQL, Microsoft SQL, SQLite, ...
Server-client model

- Two distinct pieces of software
- One instance of server
- Multiple instances of clients
- Example: web server and web clients
  - Web server provides HTML pages
  - Web client requests pages and renders them
Server-client model

- **Server**
  - Accept connections from clients
  - Process requests and provide results to clients
- **Client**
  - Connect to server
  - Make requests and get results from server
Example

- Healthcare provider (e.g. UPMC)
- Server operated at corporate headquarters
- Clients operated at
  - doctors offices, hospitals, business units
Working with databases
Interaction paradigm

- Connect to database server
- Issue request
- Fetch result
- Repeat
- Disconnect from server
Clients

- Graphical client
  - web interface, point and click
- Native client
  - command line based
- Client library
  - programmatic access – e.g. R packages
Live demo

- Today’s demonstrations will be within SQLite 3 (http://www.sqlite.org)
- Lightweight (in-process) database software
- Database from http://baseball-databank.org
SQLite version 3.7.5
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
List all tables in the database

SQLite version 3.7.5
Enter "help" for instructions
Enter SQL statements terminated with a ";
sqlite> .tables
List all tables in the database

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Table Name</th>
<th>Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllstarFull</td>
<td>FieldingOF</td>
<td>Schools</td>
</tr>
<tr>
<td>Appearances</td>
<td>FieldingPost</td>
<td>SchoolsPlayers</td>
</tr>
<tr>
<td>AwardsManagers</td>
<td>HallOfFame</td>
<td>SeriesPost</td>
</tr>
<tr>
<td>AwardsPlayers</td>
<td>Managers</td>
<td>Teams</td>
</tr>
<tr>
<td>AwardsShareManagers</td>
<td>ManagersHalf</td>
<td>TeamsFranchises</td>
</tr>
<tr>
<td>AwardsSharePlayers</td>
<td>Master</td>
<td>TeamsHalf</td>
</tr>
<tr>
<td>Batting</td>
<td>Pitching</td>
<td>xref_stats</td>
</tr>
<tr>
<td>BattingPost</td>
<td>PitchingPost</td>
<td></td>
</tr>
<tr>
<td>Fielding</td>
<td>Salaries</td>
<td></td>
</tr>
<tr>
<td>sqlite&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQLite version 3.7.5
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
Describe the table ‘Master’

sqlite> .schema Master
Describe the table ‘Master’

```sql
sqlite> .schema Master
CREATE TABLE Master (
  lahanID integer NOT NULL primary key autoincrement,
  playerID varchar(10) NOT NULL default '',
  managerID varchar(10) NOT NULL default '',
  hofID varchar(10) NOT NULL default '',
  birthYear integer default NULL,
  birthMonth integer default NULL,
  birthDay integer default NULL,
  birthCountry varchar(50) default NULL,
  birthState char(2) default NULL,
  birthCity varchar(50) default NULL,
  deathYear integer default NULL,
  deathMonth integer default NULL,
  deathDay integer default NULL,
  deathCountry varchar(50) default NULL,
  deathState char(2) default NULL,
  deathCity varchar(50) default NULL,
  nameFirst varchar(50) default NULL,
  nameLast varchar(50) default NULL,
  nameNote varchar(255) default NULL,
  nameGiven varchar(255) default NULL,
  nameNick varchar(255) default NULL,
  weight integer default NULL,
  height double(4,1) default NULL,
  bats varchar(255) default NULL,
  throws varchar(255) default NULL,
  debut date default NULL,
  finalGame date default NULL,
  college varchar(50) default NULL,
  lahan40ID varchar(9) default NULL,
  lahan45ID varchar(9) default NULL,
  retroID varchar(9) default NULL,
  holtzID varchar(9) default NULL,
  bbrefID varchar(9) default NULL
); sqlite>
```
Describe the table ‘Salaries’

```
sqlite> .schema Salaries
CREATE TABLE `Salaries` (  
  `yearID` integer NOT NULL default '0',
  `teamID` char(3) NOT NULL default '',
  `lgID` char(2) NOT NULL default '',
  `playerID` varchar(9) NOT NULL default '',
  `salary` double(10,2) NOT NULL default '0.00'
);
sqlite>
```
How do we get the top 10 salaries?
How do we get the top 10 salaries and the players’ names?
How do we get the top paid player for each team in 2010?
Introduction to SQL
• **Structured Query Language**

• Declarative language used for communicating with many databases

• Queries performed using single statements, ended by semi-colon.
# Basic Commands

<table>
<thead>
<tr>
<th>Operation</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>List available databases</td>
<td>SHOW DATABASES</td>
</tr>
<tr>
<td>List tables in a database</td>
<td>SHOW TABLES IN database</td>
</tr>
<tr>
<td>List columns in a table</td>
<td>SHOW COLUMNS IN table</td>
</tr>
<tr>
<td>Describe the datatypes of the columns in a table</td>
<td>DESCRIBE table</td>
</tr>
<tr>
<td>Change the default database</td>
<td>USE database</td>
</tr>
</tbody>
</table>
## Basic Commands

<table>
<thead>
<tr>
<th>Operation</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>List available databases</td>
<td>.databases</td>
</tr>
<tr>
<td>List tables in a database</td>
<td>.tables</td>
</tr>
<tr>
<td>List columns in a table</td>
<td></td>
</tr>
<tr>
<td>Describe the datatypes of the columns in a table</td>
<td>.schema table</td>
</tr>
<tr>
<td>Change the default database</td>
<td></td>
</tr>
</tbody>
</table>
SELECT

• Most important command
• Used for selecting subsets of the data
• Single statement – can be quite daunting
• Returns a table
  • names of columns, rows with values
SELECT columns or computations
   FROM table
   WHERE condition
   GROUP BY columns
   HAVING condition
   ORDER BY column [ASC | DESC]
   LIMIT offset,count;

Most clauses are optional
SELECT columns or computations
FROM table
WHERE condition
GROUP BY columns
HAVING condition
ORDER BY column [ASC | DESC]
LIMIT offset, count;

SQL Keywords are case-insensitive
column names are case-sensitive!
Selecting columns

• Columns specified in comma-separated list
• * to specify all columns
**Specific columns**

```sql
SELECT PlayerID, yearID, AB, H FROM Batting;
```

```sql
sqlite> SELECT PlayerID, yearID, AB, H FROM Batting;
playerID|yearID|AB|H
aardsda01|2004|0|0
aardsda01|2006|2|0
aardsda01|2007|0|0
... 
strasz01|2010|20|1
beachbr01|2010|5|1
nifu01|2010|0|0
chapmar01|2010|0|0
igarary01|2010|0|0
sqlite>
```
```sql
SELECT * FROM Salaries;
```

```
sqlite> SELECT * FROM Salaries;
1980|TOR|AL|stiebda01|55000.0
1981|NYA|AL|jacksre01|588000.0
1981|TOR|AL|stiebda01|85000.0
1982|TOR|AL|stiebda01|250000.0
1983|TOR|AL|stiebda01|450000.0
1984|TOR|AL|stiebda01|650000.0
1985|ATL|NL|barkele01|870000.0
1985|ATL|NL|bedrost01|550000.0
1985|ATL|NL|benedbr01|545000.0
1985|ATL|NL|campri01|633333.0
1985|ATL|NL|ceronri01|625000.0
1985|ATL|NL|chambch01|800000.0
...```
All columns, ordered by salary

```sql
SELECT * FROM Salaries ORDER BY Salary;
```

```
sqlite> SELECT * FROM Salaries ORDER BY Salary;
1993|NYA|AL|jamesdi01|0.0
1999|PIT|NL|martija02|0.0
1959|DET|AL|cashno01|7000.0
1993|NYA|AL|silveda01|10900.0
1994|CHA|AL|carych01|50000.0
1997|FLO|NL|penaal01|50000.0
...
2009|NYA|AL|rodrial01|33000000.0
2010|NYA|AL|rodrial01|33000000.0
sqlite>
```
Top 10 salaries

```
SELECT * FROM Salaries ORDER BY Salary DESC LIMIT 10;
```

```
sqlite> SELECT * FROM Salaries ORDER BY Salary DESC LIMIT 10;
2009|NYA|AL|rodrial01|33000000.0
2010|NYA|AL|rodrial01|33000000.0
2008|NYA|AL|rodrial01|28000000.0
2005|NYA|AL|rodrial01|26000000.0
2010|NYA|AL|sabatcc01|24285714.0
2009|LAN|NL|ramirma02|23854494.0
2007|NYA|AL|giambja01|23428571.0
2008|NYA|AL|giambja01|23428571.0
2007|NYA|AL|rodrial01|22708525.0
2010|NYA|AL|jeterde01|22600000.0
sqlite>
```
Selecting rows

- Boolean condition in WHERE clause are evaluated for each row
- Similar to R’s subset() function
### Subset of rows

```sql
SELECT PlayerID,yearID,AB,H FROM Batting WHERE AB > 100 AND H > 0;
```

```sql
sqlite> SELECT PlayerID,yearID,AB,H FROM Batting WHERE AB > 100 AND H > 0;
...
castrst01|2010|463|139
heywaja01|2010|520|144
lucrojo01|2010|277|70
morelmi01|2010|145|37
stantmi03|2010|359|93
worthda01|2010|106|27
navada01|2010|161|39
sqlite>
```
Calculated columns

• Functions of columns (computations) can be used to return **calculated columns**

• Common functions

  • COUNT(), AVG(), SUM(), MIN(), MAX(), VAR_SAMP(), STDDEV_SAMP()
Calculated columns

<table>
<thead>
<tr>
<th>sqlite&gt; SELECT MAX(AB) FROM Batting;</th>
</tr>
</thead>
<tbody>
<tr>
<td>716</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sqlite&gt; SELECT MIN(AB), MAX(AB), AVG(AB) FROM Batting;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 716 155.085227726386</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sqlite&gt; SELECT AB, H, H/AB FROM Batting;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
</tr>
<tr>
<td>2 0 0</td>
</tr>
<tr>
<td>0 0</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0</td>
</tr>
<tr>
<td>468 131 0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Calculated columns

SELECT AB, H, H/CAST(AB AS REAL) FROM Batting;

Here we specify a calculated column “the batting average” (number of hits / number of at bats)

H and AB are both integers, so by default the result is also integer. Casting one of them as a REAL (floating point number) fixes this problem.
## Calculated column

```sql
sqlite> SELECT AB, H, H/CAST(AB AS REAL) FROM Batting;

...  
27|4|0.148148148148148  
65|15|0.230769230769231  
0|0|          
20|1|0.05  
5|1|0.2  
0|0|          
0|0|          
0|0|          
0|0|          
sqlite>
```
To order by the calculated column, we have to rename it
Top 10 batting averages

SELECT PlayerID, yearID, H/CAST(AB AS REAL) AS BatAvg
FROM Batting ORDER BY BatAvg DESC LIMIT 10;

sqlite> SELECT PlayerID, yearID, H/CAST(AB AS REAL) AS BatAvg FROM Batting ORDER BY BatAvg DESC LIMIT 10;
aberal01|1957|1.0
abernte02|1960|1.0
abramge01|1923|1.0
acklefr01|1964|1.0
allismi01|1914|1.0
altroni01|1924|1.0
altroni01|1929|1.0
anderla02|1993|1.0
archeji02|1962|1.0
averyst01|2003|1.0
...
sqlite>
Aggregation

- Use GROUP BY clause to perform calculations on groups – similar to `dply()` in R
Aggregation

sqlite> SELECT playerID, SUM(salary) FROM Salaries GROUP BY playerID
aardsda01|4259750.0
aasedo01|2300000.0
abadan01|327000.0
abbotje01|985000.0
abbotji01|12960500.0
abbotku01|4237000.0
abbotky01|259000.0
abbotpa01|6471000.0
...

Total salary by player
Aggregation

```
sqlite> SELECT playerID, SUM(salary) AS totalSalary FROM Salaries GROUP BY playerID ORDER BY totalSalary DESC LIMIT 10;
rodrial01|264416252.0
jeterde01|205430000.0
ramirma02|204807769.0
bondsba01|188245322.0
johnsra05|175550019.0
sheffga01|168008550.0
maddugr01|153845000.0
griffke02|151703682.0
delgaca01|146299000.0
martipe02|146259585.0
```
Summary

• Relational databases are ubiquitous
• Data divided into many tables
• SQL is the most common method for communicating with databases.
• Use SQL query SELECT to fetch subsets of the data.
• Next: Accessing databases from R