

Why do statisticians need to know about databases?

- Databases are in widespread use
- Data are fixed by “client”
- Data are very large and more efficiently stored in a relational database
- Allows data to be manipulated in efficient ways.
- Allows us to do some computations in database, and extract reduced data for further manipulation.

What do statisticians need to know about databases?

- Understand the relevance of databases
- Have familiarity with the basic relational component and map database terminology into statistics terms.
- Use the SELECT command of the Structured Query Language (SQL) to access data.
- Recognize when to (and know how to) do data reduction within the database rather than transferring all the data to a statistical environment.
- Understand the primitive operations of the relational algebra.
- More advanced users: how to design a database for a particular dataset, and know how database are implemented, and queries executed.

Why use a database?

- Supports shared data - reduce number of copies of data
- Easier to update, correct errors and augment with new variables since only one location
- No need to back-up several copies of the data.
- Single maintainer and point of expertise.
- Optimize resource allocation with powerful servers.
- Control concurrent access to data - synchronize access
- Control transactions
- Ensure security and integrity of data
- Share and integrate data from different applications
- Support multiple views of the same data
- Simple to understand: data conceptually represented in 2-dimensional tables
- Simple to build and program applications for use

Pluses and Minuses

Databases Avoid:

- ambiguity
- inconsistency
- redundancy
- wasted space
- effort
- time

Disadvantages

- Harder to exploit application-specific data structures directly.
- Applications must be able to communicate with database.
- User's potentially have to install additional software - once only.

A simple example from Rolland

Question

How would you break this banking table up into multiple tables to reduce the redundancy?

CNo	Name	Address	ANo	Balance	Branch	Address	Mngr
1	Smith, J	101 Elm	201	\$12	Downtown	101 Main St	Reed
2	Smith, D	101 Elm	201	\$12	Downtown	101 Main St	Reed
2	Smith, D	101 Elm	202	\$1000	Downtown	101 Main St	Reed
3	Brown, D	17 Spruce	203	\$117	Downtown	101 Main St	Reed
3	Brown, D	17 Spruce	301	\$10	Suburb	18 Long Ave	Green
3	Brown, D	17 Spruce	302	\$170	Suburb	18 Long Ave	Green

Tables

Here are three separate tables for customer, account, and bank branch information.

Is there any information from the original table not captured?

CNo	Name	Address
1	Smith, J	101 Elm
2	Smith, D	101 Elm
3	Brown, D	17 Spruce

ANo	Balance
201	\$12
202	\$1000
203	\$117
301	\$10
302	\$170

Branch	Address	Mngr
Downtown	101 Main St	Reed
Suburb	18 Long Ave	Green

Customer - Account Relation

What is different about this table, in comparison to the other three?

CNo	ANo
1	201
2	201
2	202
3	203
3	301
3	302

Basic notions of how objects relate to each other

- **Generalization / specialization** - set of objects are associated together in a generalized type (eg. a manager, clerical, technical are specializations of the general object employee) This is a one to one map – each manager maps to one type of employee.
An **is-a** relation
- **Aggregation** - a series of otherwise independent objects that are associated together in an aggregated type (eg. engine, chasis, wheelset are aggregated into car).
An **is-part-of** relation
- **Grouping** - a series of objects that are all "a-type-of" another object. For example, an engine, with a unique engine number, is a type of engine, such as a four cylinder, diesel engine.
A **many-to-one** relation.

Vocabulary - Translation into Statistics terms

The Table is the basic unit of storage.

- Table: Dataframe – Relation
- Row: Case/Observation – Tuple
- Column: Variable – Attribute
- Row ID: Row name – Key
- Column count: Dimension – Degree
- Row count: Size – Cardinality

Attribute Features

- Type - many different types, e.g. tinyint boolean, time stamp, blob, set
- Attributes can have qualifiers governing their entries, e.g. default values, not null, auto increment.
- Keys - unique identifier, may be a combination of attributes
- Missing value - NULL, be careful with boolean and aggregate operations
- Security - restrict permissions at many levels

The SELECT statement

The SELECT command builds a table from the tables in database

```
SELECT column(s) FROM relation(s);
```

- The column(s) may be
 - ▶ Comma-separated list of attribute names, e.g. Name, Addr
 - ▶ * to indicate all columns
 - ▶ aggregate function such as min(Balance)
- The relation(s) is the name of a single relation (table) or a comma separated list of tables
- The statement ends with a semi-colon

WHERE clause

The WHERE clause allows you to identify a subset of tuples to use in building the new relation. The constraint that appears in the WHERE clause will be a logical statement of some sort.

```
SELECT column(s) FROM relation(s) [WHERE constraints];
```

- $X = 3$ will select all those tuples where the attribute `X` is 3.
- $X > 3$ AND $Y < Z$ will select all those tuples where `X` is greater than 3 and the value in attribute `Y` is less than attribute `Z`.
- Letter IN ('A' , 'B' , 'X' , 'Y' , 'Z') will select all those tuples where the attribute `Letter` has one of the five values provided.

Bank Example Queries

The names and addresses of all customers.

Bank Example Queries

The names and addresses of all customers.

```
SELECT Name,Addr FROM Cust;
```

All of the attributes in the Reg table for the accounts belonging to Customer 3.

Bank Example Queries

The names and addresses of all customers.

```
SELECT Name,Addr FROM Cust;
```

All of the attributes in the Reg table for the accounts belonging to Customer 3.

```
SELECT * FROM Reg WHERE CustNo = 3;
```

The account numbers for those accounts that are overdrawn in the Downtown branch.

Bank Example Queries

The names and addresses of all customers.

```
SELECT Name,Addr FROM Cust;
```

All of the attributes in the Reg table for the accounts belonging to Customer 3.

```
SELECT * FROM Reg WHERE CustNo = 3;
```

The account numbers for those accounts that are overdrawn in the Downtown branch.

```
SELECT IDNo FROM Acct  
WHERE Branch = 'Downtown' AND Balance < 0;
```


GROUP BY and HAVING

- Grouping allows you to group rows with a similar value into a single row and operate on them together.
- In addition, the HAVING clause can accompany the GROUP BY clause.
- HAVING restricts the groups that are to be selected. It works similarly to the WHERE clause.
- Note that they are not interchangeable: the WHERE clause modifies the FROM, and the HAVING modifies the GROUP BY.

Examples

The total balance for all accounts belonging to a branch.

Examples

The total balance for all accounts belonging to a branch.

```
SELECT SUM(Balance) FROM Acct GROUP BY Branch;
```

The total overdrawn amount for all overdrawn accounts at each branch, for those branches which have at least \$100 overdrawn.

Examples

The total balance for all accounts belonging to a branch.

```
SELECT SUM(Balance) FROM Acct GROUP BY Branch;
```

The total overdrawn amount for all overdrawn accounts at each branch, for those branches which have at least \$100 overdrawn.

```
SELECT SUM(Balance) FROM Acct WHERE Balance < 0  
      GROUP BY Branch HAVING Min(Balance) < -100;
```

Joining Multiple Tables

- There are several different kinds of joins.
- This is a good time to talk about relational algebra
- with the “join”, every tuple in one table is joined with every tuple in the other table, i.e all possible combinations are formed.
- The WHERE clause also allows you to restrict the joining of the tables. For example, if we only want to join a tuple in Table1 with the tuple in Table2 if the value of X in Table1 matches the value of Y in Table2, then we would say WHERE X=Y.
- If there are variables with the same name in different tables, there needs to be a way to differentiate them. The table name is used as a prefix: TableName.VariableName
- Nicknames can be assigned to a table to make for less typing.

Examples

The name(s) of the owner(s) for every account.

Examples

The name(s) of the owner(s) for every account.

```
SELECT Name, IDNo FROM Cust,Reg WHERE Cust.CustNo = Reg.CustNo;
```

Note that here the two relations Cust and Reg are joined by matching values of the attribute CustNo in the Cust relation with the values of CustNo in the Reg relation.

The customer name, account number, and balance of the account for all accounts.

Examples

The name(s) of the owner(s) for every account.

```
SELECT Name, IDNo FROM Cust,Reg WHERE Cust.CustNo = Reg.CustNo;
```

Note that here the two relations Cust and Reg are joined by matching values of the attribute CustNo in the Cust relation with the values of CustNo in the Reg relation.

The customer name, account number, and balance of the account for all accounts.

```
SELECT C.Name, A.IDNo, A.Balance FROM Cust C, Reg R, Acct A  
WHERE C.CustNo = R.CustNo AND A.IDNo = R.AcctNo;
```

Note this is an example that uses nicknames.

Challenges to teaching this material

- Software: SQLite comes installed on Mac OS X; another option is MySQL
- SQLite -Pros
 - ▶ Stand alone - each person has their own database
 - ▶ No need for a server to be running
 - ▶ Database is one file with all tables in it
 - ▶ Install software once
- SQLite - Cons
 - ▶ Limited SQL commands - currently
 - ▶ Database is not shared
 - ▶ Speed of queries can be slow
- Support: Students can use on laptops, but will need install SQLite.
- “Mistakes”: Students working on a central large database, may tie up the system
- Examples: Finding interesting public databases

- Rolland, *Essentials of Databases*
- Date,
- Celko, *SQL for Smarties*
- Lab: Baseball
 - ▶ Shell script for downloading and converting to SQLite
 - ▶ RSQLite package for extracting data from database using SQL commands and bringing into R variables
 - ▶ Sample code for getting started
 - ▶ Questions to explore data