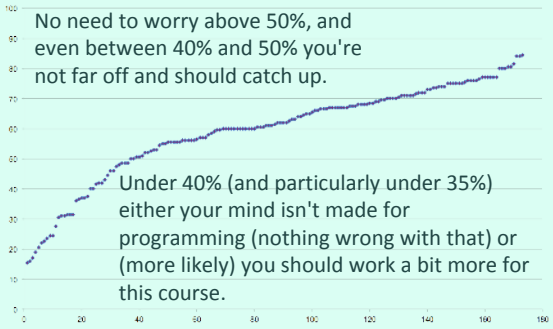
CS209

Computer system design and application

Stéphane Faroult
faroult@sustc.edu.cn

Zhao Yao zhaoy6@sustc.edu.cn
Liu Zijian liuzijian47@163.com
Li Guansong intofor@163.com

MidTerm Exam



Because memory is so cheap these days many people load everything in memory, and save everything when done – or they use databases, which require a different approach.

- Load all data at once
- Work in memory ➡ Collections
- Save everything at once
- Use databases when data needs to be shared or for very large volumes

BIG PROBLEM

When working with databases, the logic is **very** different from when you work with files – few Java developers understand it!

The approach that is the correct one with files is the wrong one with databases, because databases are **systems** that are optimized for data retrieval and processing and work much faster than anything you can do in Java.

In practice, what are mostly files used for today?

Multimedia, Documents — Handled by specialized programs

Parameters

You have seen it (Media and Image in JavaFx, Property files) in many cases all the file reading is performed by a method in a specialized object and eveything is transparent for the developer.

Parameter Files

Specify directories ?
Remote connections?

Java Development Kit

Java Runtime Environment

Parameter files are very important for allowing user to modify some important settings when they just have a .class.

Command-line parameters

javac

Parameter Files

Assign reasonable default values

Replace values found in parameter file

Program

Replace values specified on the command line

Locations for what you need should be in parameter (property) files.

In practice, what are mostly files used for today?

Multimedia, Documents — Handled by specialized programs

Parameters

Data collection and exchange

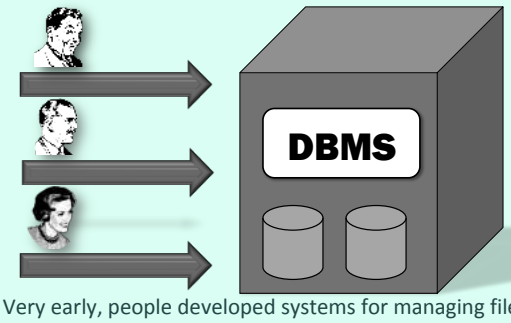
You mostly read and write data for exchanging data between systems – which include database systems.

Time to talk about

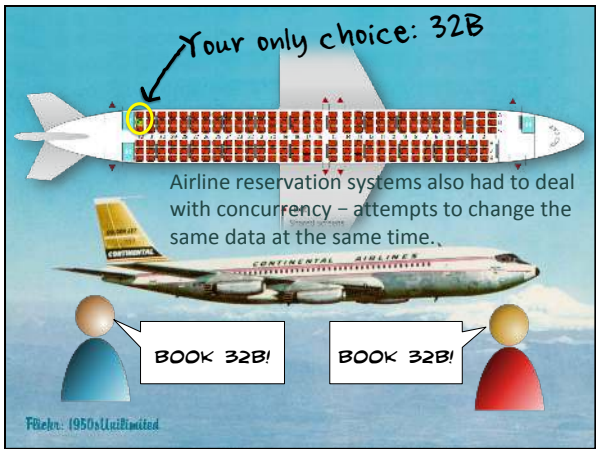
DATABASES

Computers have always been used for processing data – which in the 1960s wasn't as "Big" as today but too big for manual processing.

DBMS means DataBase Management System.



Very early, people developed systems for managing files and retrieving data, so that they need not bother with the lower software layers.



Airline reservation systems also had to deal with concurrency – attempts to change the same data at the same time.

This is what database systems where designed for: checking that changes don't lead to an inconsistent state (two people in the same seat, reservation for a non-existing flight), and also to retrieve data as fast as possible, using a "high-level" language (find this that satisfies those conditions ...) instead of looping on file records and checking each one.

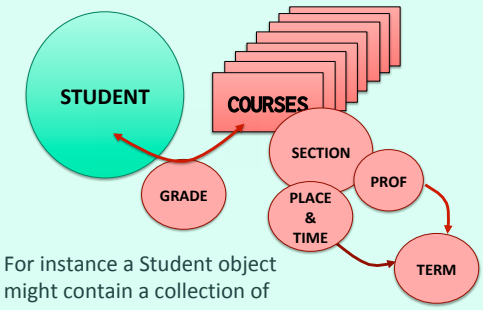
CONTROL changes

RETRIEVE data

Problem of shared data:

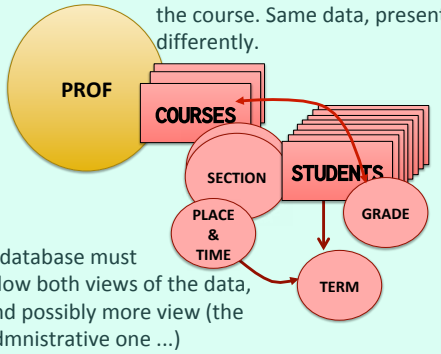
DIFFERENT VIEWS

The problem with shared data is that people have different views of the same things. When you design an object for a single application, it's relatively easy.



For instance a Student object might contain a collection of Course objects, and the professor is an attribute of the course.

For a prof, each course is associated with a collection of students enrolled in the course. Same data, presented differently.



A database must allow both views of the data, and possibly more view (the admnistrative one ...)

EARTHQUAKES

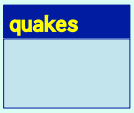
id
UTC date
latitude
longitude
magnitude
depth
region

```
FOX ISLANDS, ALEUTIAN ISLANDS
ANDREANOF ISLANDS, ALEUTIAN IS.
RAT ISLANDS, ALEUTIAN ISLANDS
SOUTH OF ALEUTIAN ISLANDS
NEAR ISLANDS, ALEUTIAN ISLANDS
```

To illustrate the advantages of databases, the region for earthquakes is something hard to process. All these regions refer to the North Pacific area.

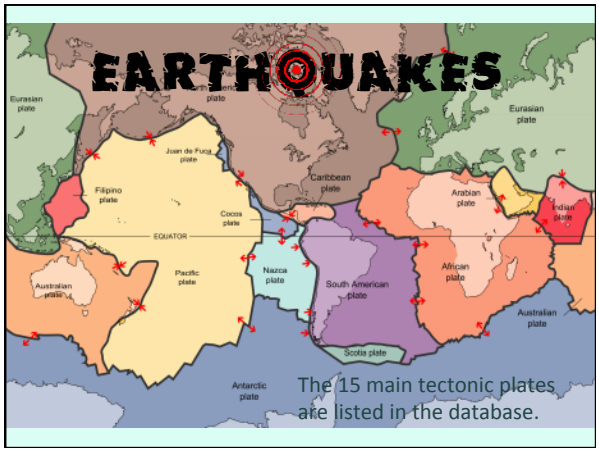
EARTHQUAKES

id
UTC date
latitude
longitude
magnitude
depth
region



You can have one collection of quakes but then filtering by region is difficult. More data has been added to the database supplied to you.

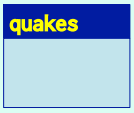
EARTHQUAKES



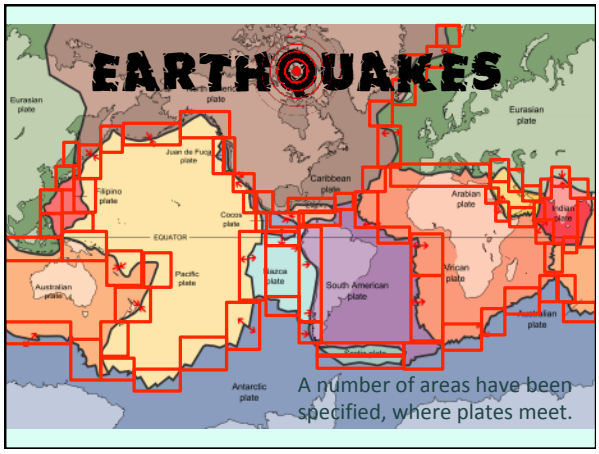
The 15 main tectonic plates are listed in the database.

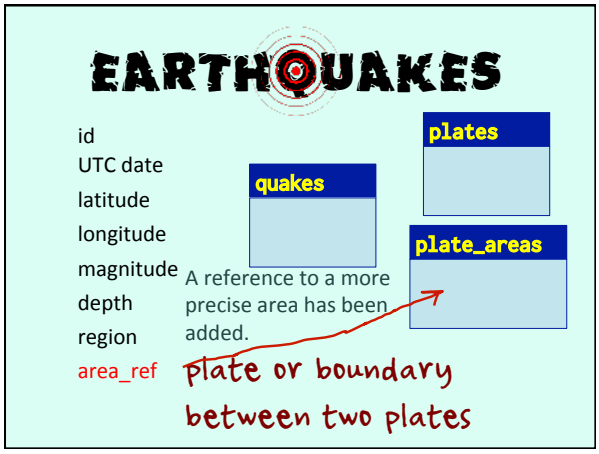
EARTHQUAKES

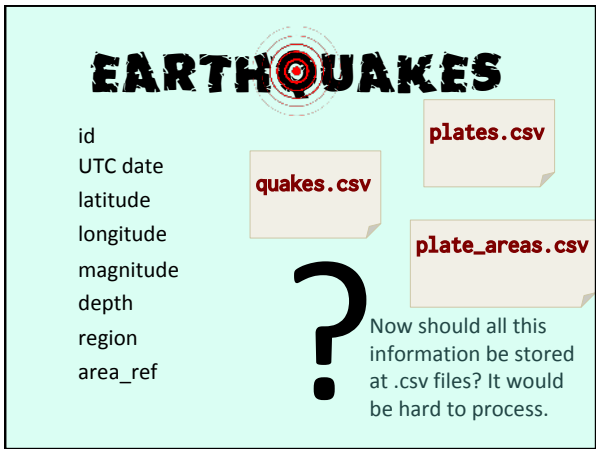
id
UTC date
latitude
longitude
magnitude
depth
region

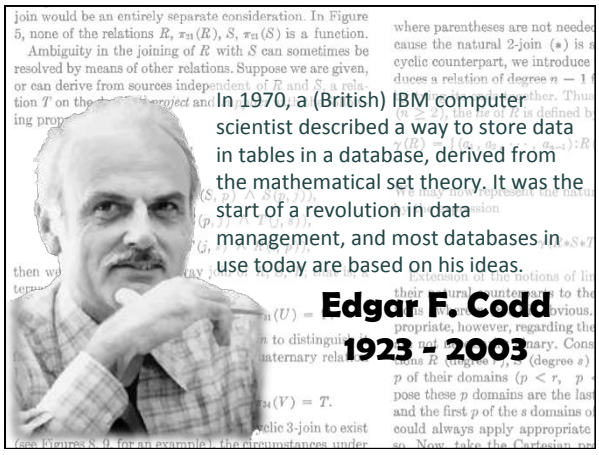


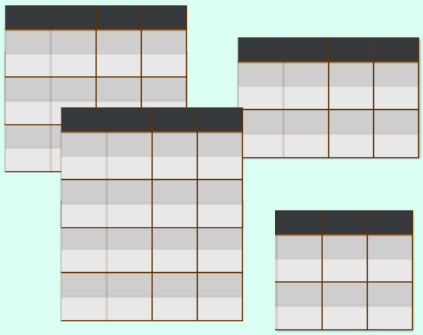
But the interesting areas are where these plates collide – the boundaries between plates (even if earthquakes can happen elsewhere, there are many minor plates)


EARTHQUAKES
A number of areas have been specified, where plates meet.


EARTHQUAKES
id
UTC date
latitude
longitude
magnitude
depth
region
area_ref
quakes
plates
plate_areas
A reference to a more precise area has been added.
plate or boundary between two plates


EARTHQUAKES
id
UTC date
latitude
longitude
magnitude
depth
region
area_ref
quakes.csv
plates.csv
plate_areas.csv
?
Now should all this information be stored at .csv files? It would be hard to process.


In 1970, a (British) IBM computer scientist described a way to store data in tables in a database, derived from the mathematical set theory. It was the start of a revolution in data management, and most databases in use today are based on his ideas.
Edgar F. Codd
1923 - 2003



Codd said that the database should give a tabular ("as tables") view of data, which is quite natural. Each table contains a set.

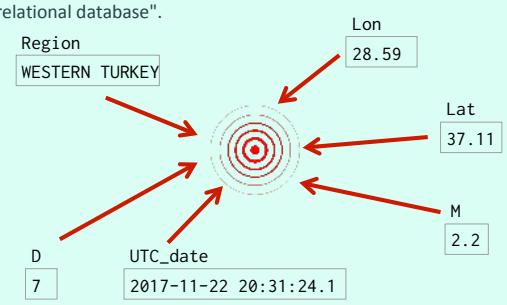
Sets are unordered. If columns are ordered differently, the information isn't changed. Same with rows.

UTC_date	Lat	Lon	D	M	Region
2017-11-22 20:31:24.1	37.11	28.59	7	2.2	WESTERN TURKEY

D	Region	UTC_date	Lon	M	Lat
7	WESTERN TURKEY	2017-11-22 20:31:24.1	28.59	2.2	37.11

But Sets don't allow duplicates. It's important that each different row is stored only once (you define "keys" that allow to specify one particular row. Here, UTC_date, longitude and latitude would be the key)

Because all the attributes are related to a single event, the table is also known as a "relation", hence "relational theory of databases" and "relational database".



Relation

So ...

It's very tempting to equate what we find in a database to what we use in Object-Oriented programming.

Table = Class

Column = Attribute

Row = Object

?

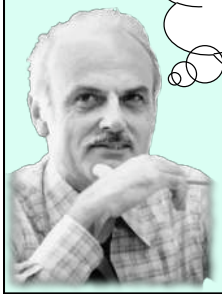
That would be a very big mistake, unfortunately made by many people.

NO!

In object-oriented databases, the focus is on the object.
In a relational database, the focus is on the set.

Codd's big idea was that it's possible to operate on sets to define subsets.

OPERATE
on relations



No references
Link through values

He also insisted on the fact that links between pieces of data in different sets would be established by common data values.

Codd defined several operations, these are the 3 most important ones.

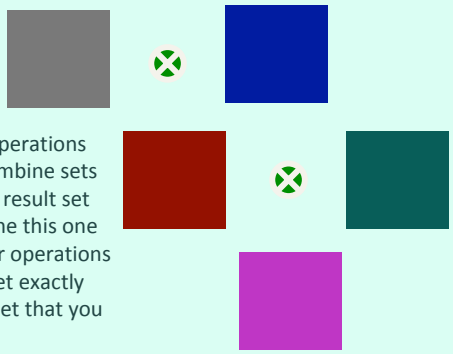
select

project

join

But the great idea, which few people get, is that tables are like variables – instead of containing one value, they contain one set.

Tables
are
variables
(sort of)



And with operations you can combine sets to obtain a result set and combine this one into further operations until you get exactly the result set that you want.

One thing that is very important is that tables should be well designed (they must follow a number of rules, called normal forms). You don't want for instance data to be repeated many times, because it would make changes difficult. You must also know precisely what uniquely identifies a row (it may be all columns, but most often it's one or a few columns)

GOOD MODELLING IS KEY

ENTITIES

"Existing things"

student

prof

course

When you design, you look for "entities", things that exist independently of the others (a course can be on the catalogue without any body taking it and teaching it). You must know what identifies each item: a code, a student/employee id, mail address, phone number can be good identifiers. A name isn't good, as several people can have the same name. Then you have attributes. One entity will be one table.

RELATIONSHIPS

Then you have relationships, that link entities together. If an entity can be linked to only one other entity (for instance Student->Dormitory) it can be an attribute of an entity.

student

prof

course

Session

key = combination of entity keys

Often it will be a table relating identifiers, because a student takes many courses and there are many students in a course.

All this stage of organizing tables is rather difficult and often underrated. If poorly done it can cause many problems.

Distinguishing between what is an attribute and what should be in a relation is often difficult.

NORMALIZATION

What identifies an earthquake?

Date

Latitude

Longitude

Id

I said that date, latitude and longitude identify an earthquake. In the database there is also a sequential number used as identifier. The reason is that you are not supposed to modify an identifier. When an earthquake is first reported, the location isn't always very precised and is sometimes updated later.

Databases are usually associated with a "query language" called SQL, invented in the early 1970s.

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

Don Chamberlin with Ray Boyce (+ 1974)

ACT: In this paper we present the data manipulation facility for a tured English query language (SEQUEL) which can be used for accessing in an integrated relational data base. Without resorting to the conc und variables and quantifiers SEQUEL identifies a set of simple opera- on tabular structures, which can be shown to be of equivalent power t irst order predicate calculus. A SEQUEL user is presented with a cons

SQL was designed to be a very simple language with a syntax close to English that could be used by people who aren't programmers.

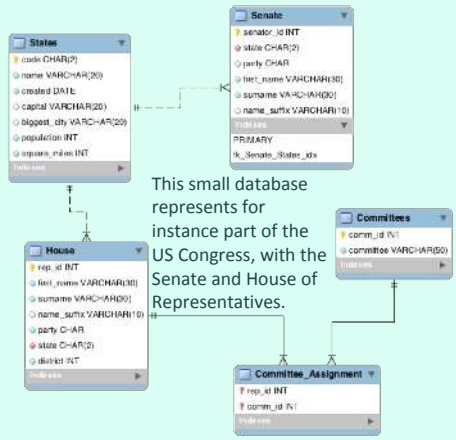
```
select ...
from ...
where ...
```

It became at the same time a major success and a major failure: today only computer programmers use SQL – but almost all of them have to use it, and sometimes very often.

TWO main components

SQL provides commands both for managing tables (creating, dropping, modifying them) and for managing data (inserting new rows, updating or deleting existing ones – plus of course commands for retrieving data that satisfies some criteria).

You can learn SQL syntax in about the same time as you can learn how chess pieces move. As with chess, things however quickly become complicated.



This small database represents for instance part of the US Congress, with the Senate and House of Representatives.

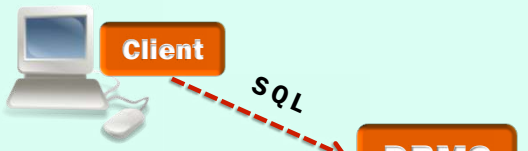
Who participates in the greatest number of House Committees ?

```
select h.first_name, h.surname,
       s.name as state, z.num_of_committees
from (select rep_id,
             count(*) as num_of_committees
      from committee_assignment
      group by rep_id
      having count(*) =
            (select max(num_of_committees)
             from (select rep_id,
                          count(*) as num_of_committees
                   from committee_assignment
                   group by rep_id) x)) z
      join house h
        on h.rep_id = z.rep_id
      join states s
        on s.code = h.state
order by h.surname, h.first_name, s.name
```

Accessing a database from a Java program is easy, even without using some tools that try to write queries for you (usually they write inefficient queries). In the project you'll use Sqlite, which is in fact a local file in which you can create tables that you can query with SQL. It works very much like a real DBMS, for which you connect to a remote server that hosts the database.

Accessing a database from a Java program

Classic case with a REAL database management server



Your program must first connect to the database (which usually required supplying a username and a password) then will send strings that contain an SQL command and will be executed by the server. Some commands will only return success or failure (creating a table, for instance). A command such as an update will be able to tell you how many rows were changed. A query will return you rows, between 0 and ...

Many Java developers have this wrong.

Must SHARE the work between what the database does ...

If you want a sum, don't retrieve all the data and iterate to sum it. Ask for the sum. The DBMS can compute it.

Retrieving JUST what you need

... and what the program does.

Presentation Computations that cannot be performed by the DBMS

Network latency

Don't multiply exchanges between your application and the server. Travelling in networks takes time (it's called latency), even with a lot of bandwidth. Sometimes data centers are pretty far away.

Round-Trips

London — 0.01s — Paris

Dallas — 0.025s — Chicago

Rio de Janeiro — 0.05s — Santiago

Los Angeles — 0.07s — New York

New York — 0.075s — London

Singapore — 0.075s — Tokyo

San Francisco — 0.095s — Tokyo

Hong Kong — 0.12s — Sydney

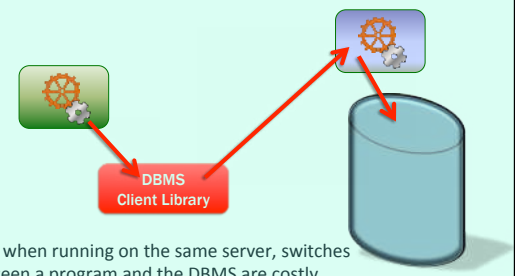
Singapore — 0.18s — Los Angeles

If you want to insert many rows into a remote database, send all the rows to the server and ask it to insert all the rows at once (this is called "batching"); it will do it very fast. If you insert rows one by one, you have to wait each time for the response from the server. The previous slide says that a return trip between Singapore and Tokyo takes 0.075s. If you assume that inserting 10,000 rows takes 0.050s (my machine can do that) doing it as one batch between Singapore and Tokyo would take 0.125s. Inserting one row (tested on my machine): 0.003s. Inserting 10000 rows row by row would be (0.075 + 0.003) * 10000 = 780s, or 13 minutes ...

ONE insert is OK

10,000 inserts HURT

Even when you are close



Even when running on the same server, switches between a program and the DBMS are costly.

Embedded databases

- No server, single-user
- "Connection" same as opening a file
- Everything else like the real thing

The alternative to real database servers are "embedded databases", systems that let you use of file as if it were a database.

"Java DB"

Pure Java

Part of the JDK

Server or embedded

Java is shipped with "Derby", sometimes called "Java DB", which can be used both ways.

www.sqlite.org

Public domain

One file to download

Used in mobile apps – and by Mozilla

Create tables in *SQLite Manager*
(Firefox Plugin)

The most popular embedded database is probably SQLite.

www.sqlite.org

Only for the C programming language

https://github.com/xerial/sqlite-jdbc

You just need a .jar file available from this address (I have also uploaded it to Sakai)

sqlite-jdbc refers to JDBC, which is a protocol allowing to access almost all databases from Java. SQL is often slightly different from database to database, not JDBC calls.

jdbc

java

database

connectivity

Load a specific connection driver (.jar file)

Database-specific connection parameters

Then java methods are the same with every database

SQL is slightly different between databases

Three main Objects

- `Connection` ⟺ Link to database, some special operations
- `PreparedStatement` ⟺ SQL command

There is also a Statement object. A PreparedStatement can be re-executed with different parameters, not a Statement.

Associate parameters

Execute a query

Loop on rows returned

`ResultSet`

Database Access Example

Assuming that the database is hosted by an Oracle server, after connection the following program prompts for a date, and issues a query that computes for each region how many quakes happened since that date. The result is displayed line by line.
The toughest part is the connection.Reflection is used for loading the driver, the name of which depends on the DBMS. Parameters required for the connection also depend on the RDBMS.

Java/JDBC

JDBC


```
import java.util.Properties;
import java.sql.*;
import java.util.Scanner;

class DBExample {
  static Connection    con = null;

  public static void main(String arg[]) {
    Properties info = new Properties();
    String     url = "jdbc:oracle:thin:@localhost:1521:orcl";
    Scanner    input = new Scanner(System.in);

    try {
      Class.forName("oracle.jdbc.OracleDriver");
    } catch(Exception e) {
      System.err.println("Cannot find the driver.");
      System.exit(1);
    }
```

You connect to the database using an url string (not an URL object ...)

Java/JDBC

```
try {
  Class.forName("oracle.jdbc.OracleDriver");
} catch(Exception e) {
  System.err.println("Cannot find the driver.");
  System.exit(1);
}
try {
    System.out.print("Username: ");
    String username = input.nextLine();
    System.out.print("Password: ");
    String password = input.nextLine();
    info.put("user", username);
    info.put("password", password);
    con = DriverManager.getConnection(url, info);
    con.setAutoCommit(false);
    System.out.println("Successfully connected.");
} catch (Exception e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
```

MySQL wants username and password independent parameters, Oracle passes them as Properties. Of course connection can fail.

Java/JDBC

Queries are simple. Placemarkers for parameters in a PreparedStatement are ? signs, implicitly numbered (from 1). Return columns are also numbered from 1. One can often only work with Strings, the DBMS can convert types.

```
System.out.print("Date: ");
String utcDate = input.nextLine();
try {
   PreparedStatement stmt =
      con.prepareStatement("select region, count(*)"
                           + " from quakes"
                           + " where UTC_date >= ?"
                           + " group by region");
   stmt.setString(1, utcDate);
   ResultSet rs = stmt.executeQuery();
   while (rs.next()) {
      System.out.println(rs.getString(1) +
                  "\t" + rs.getString(2));
   }
   rs.close();
```

Java/JDBC

```
        rs.close();
    } catch (Exception e) {
        System.err.println(e.getMessage());
        try {
           con.close();
        } catch (SQLException sqlE) {
          // Ignore
        }
        System.exit(1);
    }
    try {
        con.close();
    } catch (SQLException sqlE) {
        // Ignore
    }
  }
}
```

Queries can fail too, but returning nothing (or updating or deleting or inserting no rows) doesn't throw any exception, because the empty set is a valid set ... However, trying to insert twice the same key for instance will throw an exception.

```
Class.forName("oracle.jdbc.OracleDriver");

Class.forName("com.mysql.jdbc.Driver");

Class.forName("org.postgresql.Driver");

Class.forName("org.sqlite.JDBC");

Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
```

These are JDBC driver names for commonly use Database Management systems (note that for Derby there are two modes, embedded – single user – and not embedded)

And here are connection examples.

```
con = DriverManager.getConnection(url, info);
```

"jdbc:oracle:thin:@hostname:port:dbname"
"jdbc:postgresql://hostname:port/dbname"

user password

```
con = DriverManager.getConnection(url,
                                  username,
                                  password);
```

"jdbc:mysql://hostname:port/dbname"

```
con = DriverManager.getConnection(url);
```

"jdbc:sqlite:filename"

With Sqlite you just provide a filename. It will be created if it doesn't exist already.