Database Fundamentals

Applications for Seismology

Robert Casey
IRIS Data Management Center
March 2005

Agenda

- What a database is and why we use them
- Database access methods
- Database design
- Databases in seismology applications
- Introduction to MySQL
- Group exercise installing MySQL
- Setting up a MySQL database

Questions to be covered

- What is a database?
- How do I work with a database?
- Why would I want to use one?
- How can I set up my own database?
- How are databases used in seismology?

Goals of the presentation

- Familiarize group with database concepts
- Provide group with training in installing MySQL
- Preparation for working with Portable Data Collection Centers

What is a database?

- There are many ways we can define it.

What is a database?

- A digital repository for tabular data.
- A persistent data store.
- A collection of data records with pre-defined relationships.
- A software system that provides ready access to data records.
- A software system that manages additions, deletions, and changes to data.

Why use a database?

- Why not use something simple, like files on disk?

Why not use disk files?

- Digital information can be stored in files:
  - Text
  - Pictures
  - Formatted data
- Files are the standard form of disk storage in a computer operating system
- No additional software necessary

Why not use disk files?

- Files can be identified by name
- Files can be placed in directories for organizing them
- Files are simple to
  - Edit
  - Create
  - Copy
- Little training required to access them

There is a downside, though…

- Files may seem easy at first, but they present their own set of problems.

Downside to using files

- Searching for specific information can be difficult
- Filtering through lots of data files for specific items can be a time-consuming, linear process
- Relationships between different pieces of information can be difficult to define
- Non-standard data retrieval methods

Downside to using files

- Editing of files is generally ad-hoc, with no enforced data formatting
- Addition and deletion of records can be difficult to perform safely
- Other data dependencies may be affected by the file editing
- Bulk editing of many data records can be difficult and time consuming

Downside to using files

- Difficult to represent columns of related data in different ways.
- Management of large datasets with deep directory trees can be difficult to traverse.
- Datasets can have excessive redundancies that are troublesome to manage.

Is a database the answer?

- What do we gain from the database approach?

Database benefits

- Data is quick to access and filter.
- Easy to change and update entries.
- Entire records and single values can be updated individually or in bulk.
- Data relationships are established up front.

Database benefits

- Formatting for data fields is controlled
- Large, complex data relationships are easy to manage and traverse
- Most databases use a common access language
- A number of standard software interfaces allow access by applications

Things to watch for

Database issues

- Need database management software
- Some training is required to use
- Large databases require a dedicated administrator to manage
- Setting up a database requires application of up front design principles

Talking to a database

- How do we access one of these?

Talking to a database

- Need a database client…
- …which connects to a database server.
- The server manages the database and provides communications to it.
- The client can be a local or remote program that accesses the server.

Talking to a database

- Communication with the server is generally through a standard query language.
- A query is a command statement that triggers access to the database.
- The query results in returned data records or allows editing of the database contents

Talking to a database

- The most commonly known query language is the Structured Query Language (SQL)
- Most database systems support SQL
- In order to use SQL, we have to first connect to a database.
- Techniques for this vary, but each database tends to have a specific name.

Talking to a database

- An example client connection to a database might be:
  - dbclient myDB
- -where the client starts up and connects to the database named ‘myDB’

Client connection

- Once the client has started up, we can now communicate to that database using SQL.
- You might see a prompt like this:
  - myDB>
- At this point, the database is ready to accept queries.

SQL query

- SQL consists of commands with specific formatting.
- Some allow data reads, while others allow us to add, delete, and edit entries from the database.
- The first SQL command we will introduce is the SELECT query.

SELECT query

- The SELECT statement requires that we specify one or more tables.
- A database table is the fundamental block of information in a database.
- A table consists of rows and columns.
- The columns represent fields of information for a particular thing
- The rows represent an individual item of information

SELECT query

- When we make a SELECT query, we ask for one or more of these columns to be returned from a table.
- Each column has a name and a data type
  - Id INTEGER
  - Employee VARCHAR
  - Date TIMESTAMP

SELECT query

- So, in getting these three fields from a table called Employees, we would type
  - SELECT Id, Employee, Date FROM Employees;
- Take note of the semicolon at the end.
- This is standard for SQL so that multiple lines can be entered before the query is run.

SELECT query

- SELECT Id, Employee, Date FROM Employees;

- SELECT comes first
- A comma-separated list of column names is then listed
- The FROM command indicates that a table name follows
- And finally the table name and a semicolon

SELECT query

- The returned data will be all of the records, or rows, in the table Employees
  - Id Employee Date
  - 1 Jane March 2, 2005
  - 2 John April 25, 2004
  - 3 Susan July 12, 2003

SELECT query

- What if there were thousands of employees?
- What if you wanted to only see employees that had a Date entry before January, 2005?
- You set conditions on which records are returned by using the WHERE clause.

SELECT query

- SELECT Id, Employee, Date FROM Employees WHERE Date < 2005-01-01;

- This example would just return the records

Id	Employee	Date
2	John	April 25, 2004
3	Susan	July 12, 2003

SELECT multiple tables

- Queries can also be performed on multiple tables.
- This operation is called a join.
- You can select some or all of each column in a table and have them displayed as if they were a single table

SELECT multiple tables

- Join operations use the WHERE clause to connect specific fields that relate the two tables together, such as an Id number.
  - SELECT Employee, OfficeNum FROM Employees, Rooms WHERE Employees.id = Rooms.id;

SELECT multiple tables

- The result is that two separate tables are now presented in a unified fashion, linking two different kinds of information together.

Employees			Rooms		
Id	Employee	Date	Id	OfficeNum	Floor
1	Jane	2005-3-2	1	B201	2
2	John	2004-4-25	2	C105	1
3	Susan	2003-7-12	3	M300	3

SELECT multiple tables

- SELECT Employee, OfficeNum FROM Employees, Rooms WHERE Employees.id = Rooms.id;

Employee	OfficeNum
Jane	B201
John	C105
Susan	M300

SELECT multiple conditions

- Multiple conditions can also be specified, with expected results:
- SELECT Employee, OfficeNum FROM Employees, Rooms WHERE Employees.id = Rooms.id AND Date < 2005-01-01 AND Floor < 3;
  - Employee OfficeNum
  - John C105

SELECT query synopsis

- SELECT <col1>,<col2>,<col3>,…
  FROM <table1>,<table2>,….
  WHERE <condition1> <AND|OR>
  <condition2> <AND|OR>
  ……
  <conditionN>
- ;

Other query types

- There is the INSERT command for entering new data records
- INSERT INTO <table_name> (col1,col2,…,colN) VALUES (val1,val2,…valN)
- The column entries must correspond to the value entries that follow the VALUES keyword

Other query types

- The DELETE command allows records to be deleted
- DELETE FROM <table_name> WHERE <condition1> <AND|OR> <condition2> <AND|OR> ….
- Providing the proper conditions is critical to ensuring the right records are deleted
- Try a test SELECT statement first to make sure you are deleting the intended records

Other query types

- The UPDATE command lets you make changes to existing records.
- You change specific fields to new values.
- UPDATE <table_name> SET <col1>=<val1>, <col2>=<val2>,… WHERE <condition1> <AND|OR> <condition2>…….
- Only the records that match the WHERE conditions will be altered.

Query examples

- DELETE FROM Employees WHERE Employee=‘John’;
- INSERT INTO Rooms (Id,OfficeNum,Floor) VALUES (4,’G202’,2);
- UPDATE Employees SET Date=2005-3-10 WHERE Id=1;

Commit and Rollback

- Is there a danger of deleting the wrong record and not being able to recover it?
- Yes, unless you implement a rollback.
- Many databases allow you to indicate whether to auto-commit your changes or leave you to commit the changes manually.
- If you have performed a number of changes and change your mind, you can enact a rollback command.

Commit and Rollback

- A rollback changes all of your SQL transactions back to the point of your previous commit.
- Using the commit command after your transactions locks in the new changes.
- If the client is set to auto-commit, then each and every transaction is automatically put into effect.
- Rollback is not possible in this case.

Commit and Rollback

- Sometimes, rollback is a good option to take if you have programs adding or changing entries to the database.
- If an program error occurs in the middle of a change, you can safely back out of the change.
- This can be a good technique to prevent partial or incomplete updates to the database.

Commit and Rollback

- On a database with multiple clients connected to it, other clients will not see your changes until you issue a commit.
- You can perform relatively safe, isolated changes and only push them to all other clients when you are satisfied.
- If something goes wrong, the rollback results in a reset from those entries, and other clients are not affected in any way.

ACID model

- A good database system conforms to the following rules, summarized by the letters A C I D.
- A - Atomicity
- C - Consistency
- I - Isolation
- D - Durability

ACID model

- A - Atomicity
  – Database modifications must be ‘all or nothing’
  – If one part of a transaction fails, then the entire transaction fails
  – Maintains the atomic nature or wholeness of transactions in the event of a software or hardware failure

ACID model

- C - Consistency
  - Only valid data will be written to the database.
  - If a transaction violates a consistency rule, then an automatic rollback is performed
  - If the consistency rules are met, then the transaction executes
  - The database goes from one consistent state to the next on a successful execution

ACID model

- I - Isolation
  - Multiple transactions occurring at the same time should not impact each other
  - This situation occurs with multiple connecting database clients
  - The database management server should run one or the other client transactions entirely before executing one from another client
  - This prevents one transaction acting on partial data from another

ACID model

- D - Durability
  - Ensures that any transaction committed to the database will not be lost
  - In the event of a software or hardware failure, transactions are preserved through activity logs and database backups
  - Committed transactions can be restored in the event of system failure

Types of client connections

- There are many ways to connect to a database
- Different kinds of clients
- Some are native clients, that provide direct access to the server and provide special command features
- Pretty much every database has some form of native client for performing query operations

Types of client connections

- There are other ways to access the database, ones that allow programs to automatically execute queries on behalf of the user
- These are typically referred to as interfaces.

Interfaces

- Interfaces tend to be database independent in their compatibility
- The same interface can be used on different brands of database
- Software written to use a particular interface can be moved to a different database with little or no modification
- In reality, some changes are necessary, due to slightly different SQL commands and data models

Perl DBI

- Perl DBI is one such interface.
- DBI makes use of a driver to connect to a specific brand of database, called a DBD module. There is a driver for Oracle, a driver for MySQL, and other commonly known databases.
- Developers can write Perl code to query the database and run automated operations on the data.
- A special client does not have to be opened first, the DBI module runs when the Perl script does.

Database connection

- The first thing you do with DBI is make a client connection to the database.
- Here is an example for an Oracle database connection

```
$db = DBI->connect
  (‘DBI:Oracle:seismicdb’,
  ‘user’,’password’) or die “cannot connect”;
```

Database connection

- A successful connection call returns a handle to the database client ($db).
- A failed connection would run the 'die' directive, displaying an error message
- We can now use this handle to make calls to the database.
- The first thing we want to do is pass an SQL statement to the database.
- The proper way to do this with an interface is to prepare the statement before we actually execute it.

Prepared statement

```
$st = $db->prepare("SELECT name, lat, lon
   FROM stations WHERE name =
   'COCO'") or die "prepare failed";
```

- Note that there is no semicolon at the end of the statement
- DBI fills this in for you
- A statement handle is returned for driving the execution of the query

Execute the statement

- The prepared statement sets up the client to make the query, but does not actually execute it until you follow up with the execute command

```
$st->execute();
```

- The execution goes out, and the database will return matching records

Retrieving the records

- Databases typically accommodate iteration through returned database rows
- This means we don't have to load back large sets of data to read them, we can examine them a bit at a time
- In DBI, we read back one row at a time and assign them to an array or to variables

Retrieving the records

```
while ( ($name,$lat,$lon) = $st->fetchrow_array() ) {
    print "name=$name, lat=$lat, lon=$lon\n";
}
```

- Here we loop over each row and print the returned fields to the screen
- The loop exits when no more rows are available

Why we prepare first

- So why go through preparation and execution as separate steps?
- The greatest benefit comes from when the statement is executed many times in succession
- A prepared statement is passed to the database, and in most cases precompiled
- Multiple executions with different aassigned values can occur very quickly

Execution with filled values

- In our prepared statement, we asked for a specific seismic station name

```
$st = $db->prepare("SELECT name, lat, lon
   FROM stations WHERE name =
   'COCO'") or die "prepare failed";
```

Execution with filled values

- However, we can substitute this fixed name with a tag to indicate we will put something there later

```
$st = $db->prepare(“SELECT name, lat, lon
  FROM stations WHERE name = ?”) or
  die “prepare failed”;
```

- Notice the question mark (?)

Execution with filled values

- Now that the statement has been precompiled, we will execute a query with the substitution mark filled in with the station name

```
$st->execute('COCO');
```

Execution with filled values

- The power of this is found when you query for multiple station names in a single run

```
$st->execute('COCO');
        $st->fetchrow_array();
$st->execute('ANMO');
        $st->fetchrow_array();
….etc.
```

Multiple queries

```
@stations = ('COCO','ANMO','MAJO',
   'GNI',…..,'YAK');
foreach $station ( @stations ) {
   $st->execute($station);
   @lat_lon = $st->fetchrow_array();
   print "$lat_lon[0],$lat_lon[1],
   $lat_lon[2]\n";
}
```

- Prepared statements make this very fast and convenient to code

Two prepared statements

- For a single database connection, you can prepare multiple statements in advance

```
$st1 = $db->prepare("SELECT lat,lon
    FROM stations where name = ?");
$st2 = $db->prepare("SELECT name
    FROM stations where lat < ?");
$st1->execute($station);
$st2->execute($lat_max);
```

Closing the connection

- Once all executions are completed, it is good form to close the client connection before the program exits

```
$db->disconnect();
```

- DBI is no longer connected to the server

DBI commit and rollback

- Just as we discussed earlier, we can implement safe client interaction through the use of explicit commit and rollback calls

$st1->prepare("INSERT INTO stations (name,lat,lon) VALUES (?,?,?)");

$result = $st1->execute('ABCDW',34.6,23.2);

DBI commit and rollback

```
if ($result == 0) {
   #failure
   $db->rollback();
} else {
   #success
   $db->commit();
}
```

DBI auto-commit

- DBI also allows auto-commit of statements at execution time
- We set this when we connect to the database

$db = DBI-> connect('DBI:Oracle:seismicdb', {AutoCommit=>1});

- Typically, auto-commit is the default

Other interfaces

- DBI is just one example interface. There are other standard examples in widespread use:
  - JDBC for Java
  - PEAR DB for PHP
  - ODBC for C
  - and many other alternatives…
- All of these follow the same basic approach as what we have shown with DBI

Questions?

Database design

- Now we are going to delve into database design concepts
- Before the SQL queries can occur, tables have to be created and defined
- A database has to be created before we can have tables
- We need to have a plan for organizing and arranging our data before we create the database

Database design

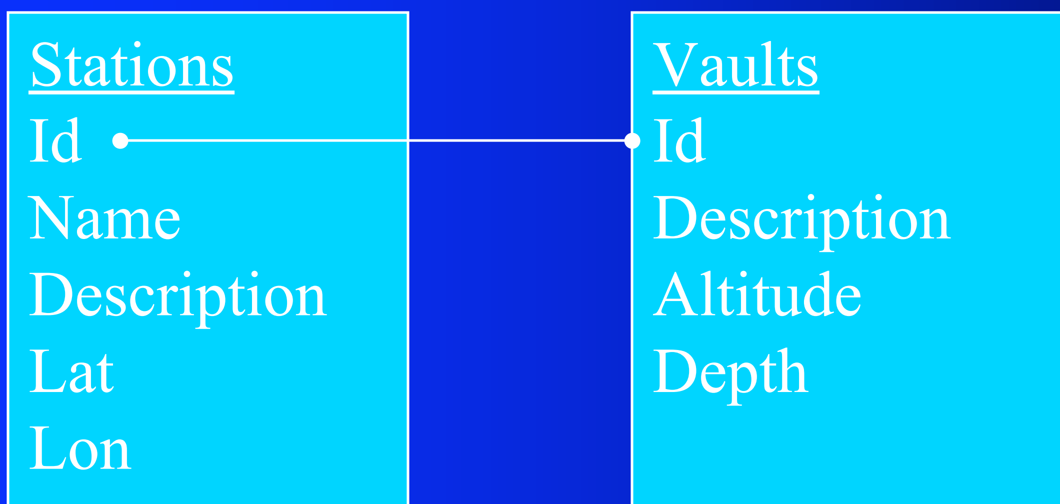
- An effective database results from good planning up front
- You want to have an understanding of the data you will be storing
- You want to know how users will want to access the data
- You will want to know how additions and modifications occur to the data

Database design

- Generally, a database does not consist of just one table
- Many tables are created, and each relate to each other in a certain way
- These relationships allow us to join tables in a query in so that we can retrieve complex representations of data
- These relationships, and the table contents are referred to as a schema.

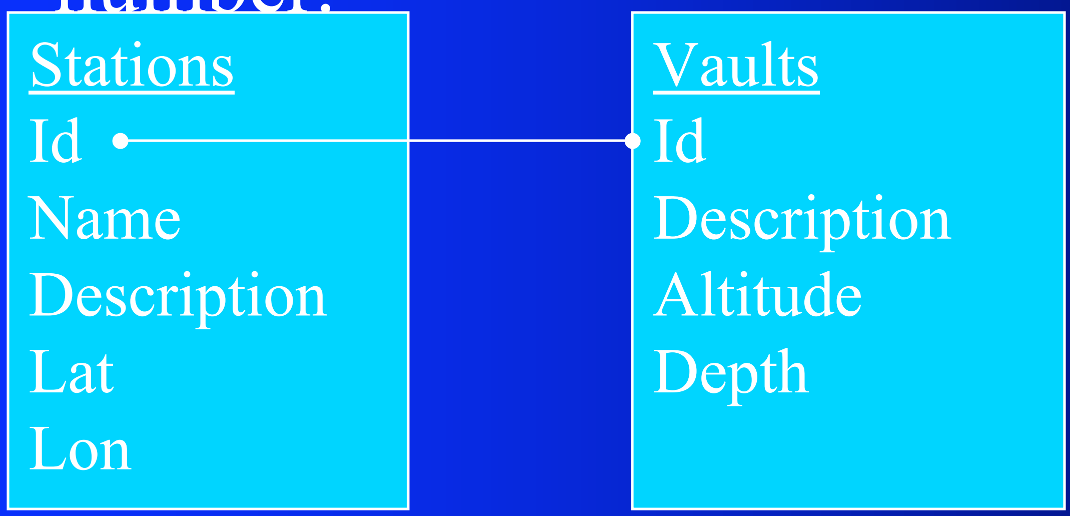
Database schema

- A schema is an illustration or plan showing data relationships



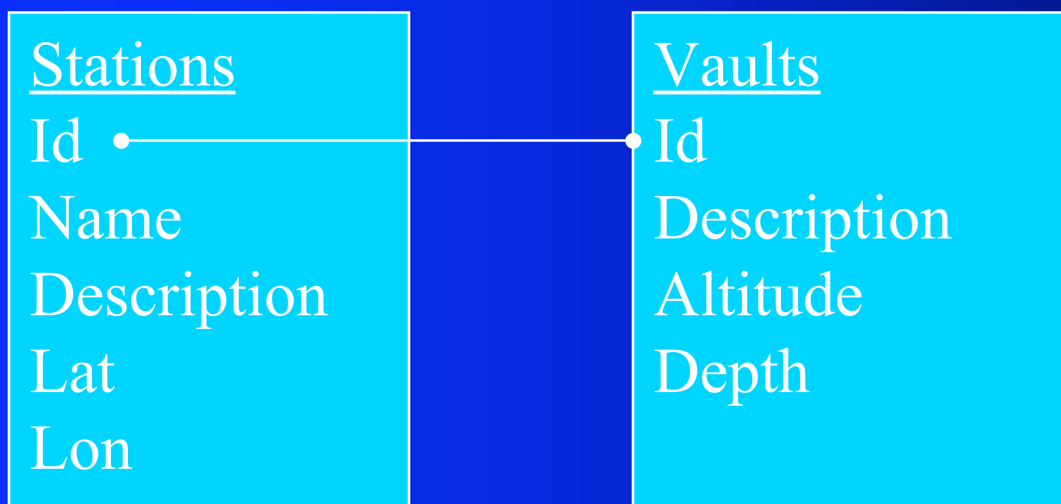
Relationship diagram

- This relationship diagram shows two tables that relate to each other by their Id number.



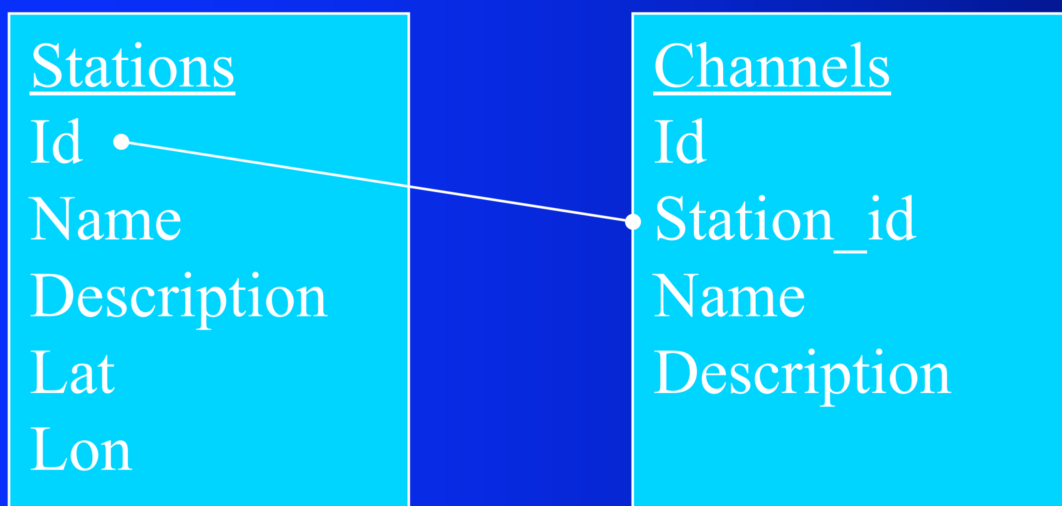
One to One relationship

- This particular example demonstrates a one-to-one relationship



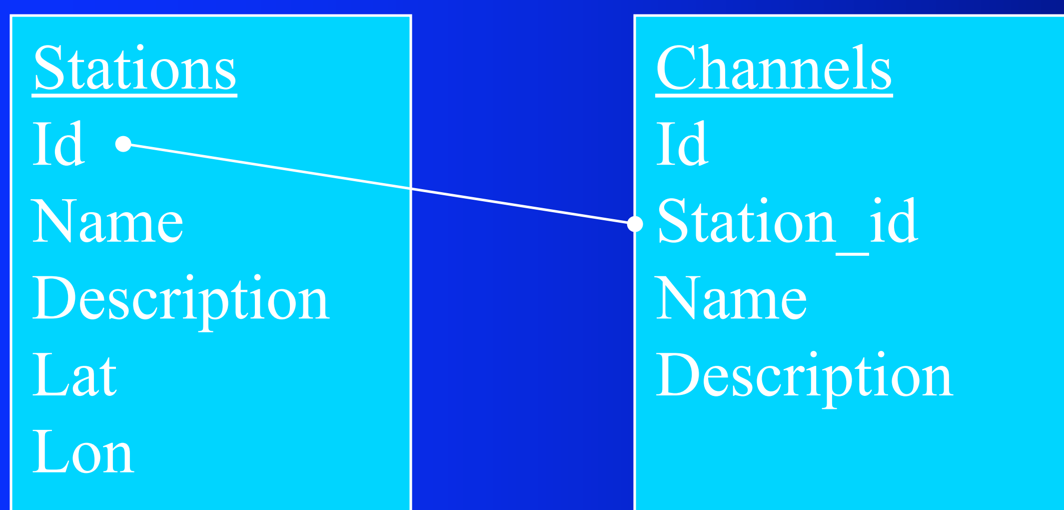
One to Many relationship

- One to many relationships are common and are the basis of most databases



One to Many relationship

- Note the change in fields relating to each other



One to Many relationship

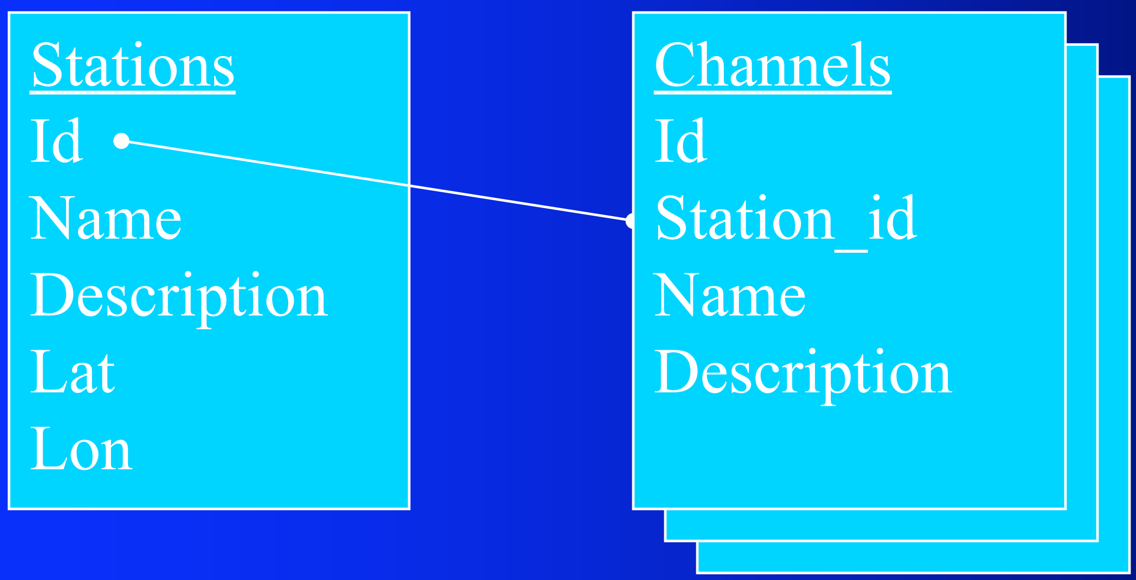
- As you can imagine, there is one station and many channels at that station site.
- Each station record has a unique id number.
- Each channel has a station_id number that references a unique station id number
- Multiple channels can use the same station_id value…they do not have to be unique

Key fields

- These fields that form table relationships are called key fields
- The key field that must be unique is the primary key, which applies to the station Id in this case
- The station_id field in Channels, which does not have to be unique, is called a foreign key

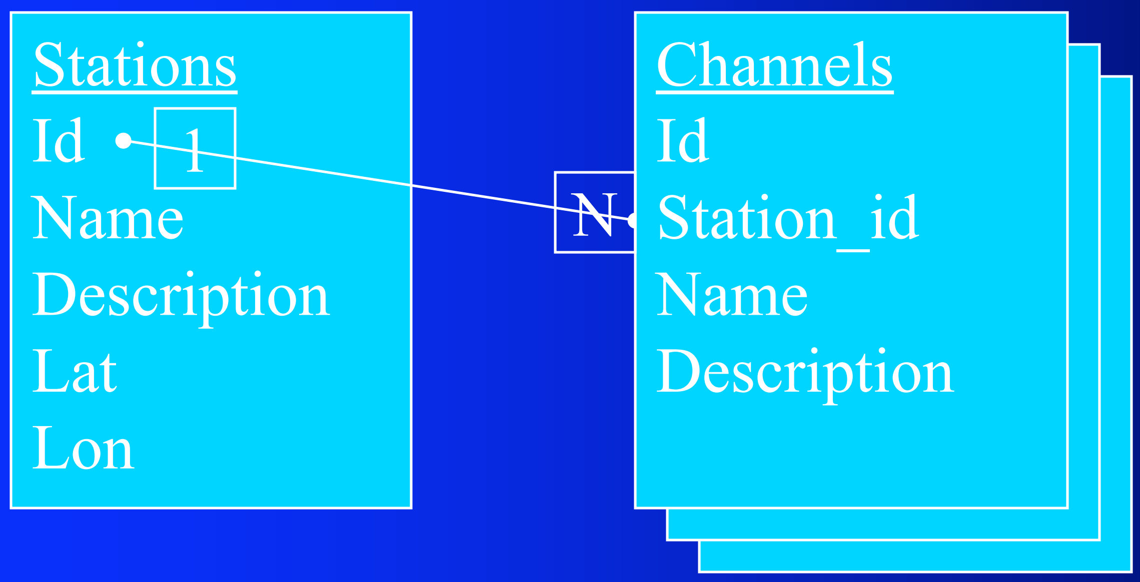
One to Many relationship

- Foreign keys refer back to the primary key in another table, forming the link



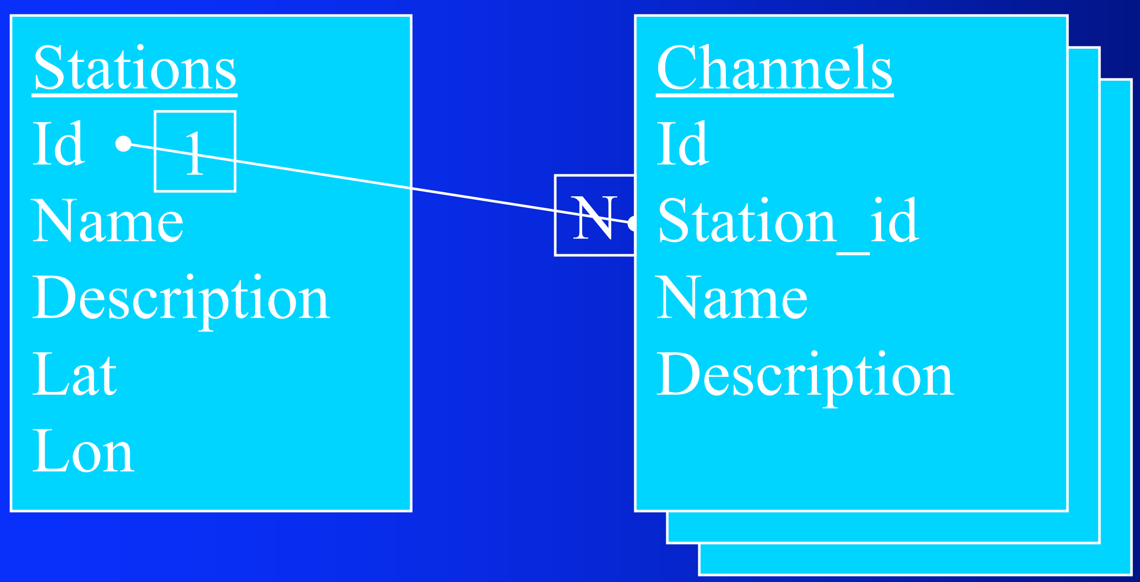
One to Many relationship

- We can signify this with symbols on the link line drawn between the two fields



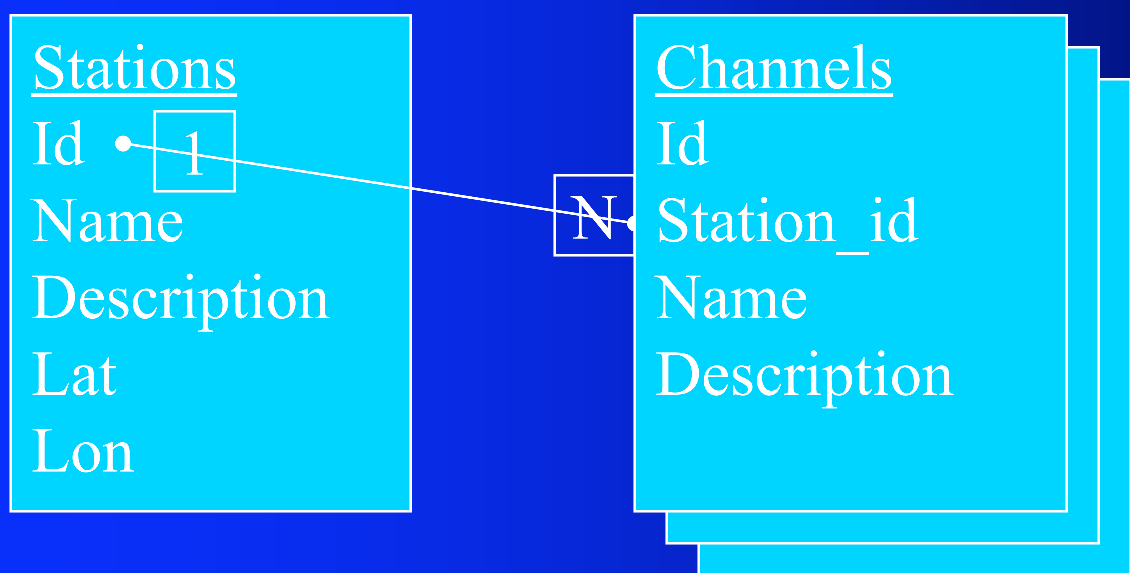
One to Many relationship

- The table on the left, on the 'one' side of the relationship, is called a primary table.



One to Many relationship

- The table on the right, on the 'many' side of the relationship, is called a related table.



Many to Many Relationship

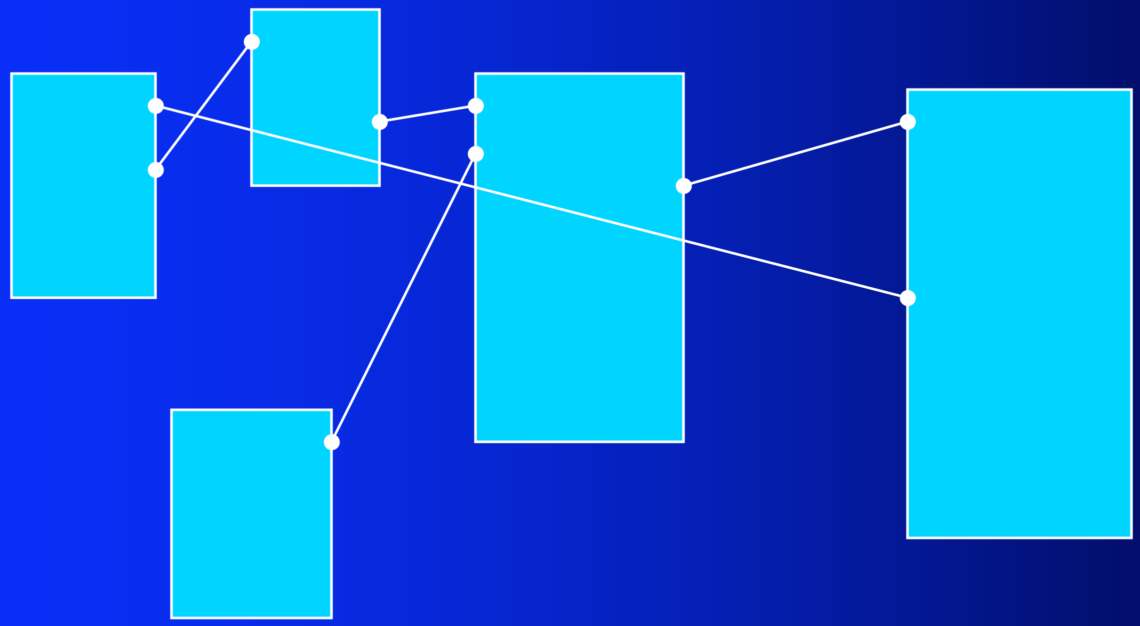
- You can establish relationships between two tables in which many records point to one record in the other table, and also the same case in reverse.
- This is known as a many to many relationship.
- Not many databases support this.
- They are not a recommended approach.

Many to Many Relationship

- The technique to get beyond this is to create a 'bridge' table in between.
- This bridge table will facilitate a many to one relationship to the other two tables, avoiding the many-to-many relationship complications.

Extending to many tables

- A typical database will have many relationships all interconnected, with many keys pointing to each other



Developing your tables

- When starting out with table design, it is good to figure out what your data fields are going to be.
- Become familiar with the entire dataset you are trying to represent, as well as how they will be used.
- Identify how these pieces of data relate to each other.

Developing your tables

- You want to put your data fields into groups that are closely related, such that there are no data redundancies.
- One to one, and one to many relationships should be established between these groups.
- You can then construct tables from these field groups.

Starting our first seismic table

- We will start out with some typical seismic data elements for our first table.
- Next, we will refine our datasets to adhere to good design principles.
- First, let’s get information on seismic stations, their channels, and their instruments.

Stations table

- Station name
- Station latitude
- Station longitude
- Station altitude
- Station depth
- Station Operator
- Number of channels
- Channel name
- Channel sample rate
- Channel gain
- Channel instrument type
- Number of instrument stages

Example table

Stations

name	lat	lon	alt	dep	operator	num_chan	chan	samp_rate	gain	inst	stages
ABC	32.5	102.6	1033	0	Joe	3	BHE,BHN,BHZ	20,20,20	5.1E+03	CMG3	4
CDE	-5.3	73.5	744	56	Sally	6	LHE,LHN,LHZ, BHE,BHN,BHZ	1,1,1, 20,20,20	7.2E+08	320	3

…..

…..

…..

Many repetitions

- As you can see in this first example, there are fields with multiple values in them.
- It is difficult to work with multiple entries in one field, especially when they are of arbitrary or varying length.

Repeating groups

- We could put them as individual entries in fields like chan1, chan2, chan3, etc….
- This is referred to as a repeating group.

Sta	chan1	chan2	chan3	chan4
ABC	BHE	BHN	BHZ	null

Repeating groups

- Repeating groups tend to result in ‘sparse’ tables with a lot of empty space.
- This technique also does not make it easy to expand the number of related entries because you soon run out of fields, or have to add more to the table.

Need a better approach

- We want to have one value per field in each row.
- We might have to repeat some values in rows in order to achieve this, such as redundant station name entries.
- Still, this would offer a solution that allows us to add as many new entries as we need.

Normalization

- This attempt to break up the data into rows with unique values is called normalization.
- Properly normalized tables means that we can find a non-repeating set of values in one place.
- This makes adding and deleting data easier and less prone to error.

Normalization

- Normalization has about 5 levels of application, each one more stringent than the last.
- We are only concerned with the first three.
- The first level that we are going to apply is called the First Normal Form.
- A table in first normal form has no multiple field values and no repeating groups of the same type of field.

First normal form

Stations

name	lat	lon	alt	dep	operator	num_chan	chan	samp_rate	gain	inst	stages
ABC	32.5	102.6	1033	0	Joe	3	BHE	20	5.1E+03	CMG3	4
ABC	32.5	102.6	1033	0	Joe	3	BHN	20	5.1E+03	CMG3	4
ABC	32.5	102.6	1033	0	Joe	3	BHZ	20	5.1E+03	CMG3	4
CDE	-5.3	73.5	744	56	Sally	6	LHE	1	7.2E+08	320LP	3
CDE	-5.3	73.5	744	56	Sally	6	LHN	1	7.2E+08	320LP	3
CDE	-5.3	73.5	744	56	Sally	6	LHZ	1	7.2E+08	320LP	3
CDE	-5.3	73.5	744	56	Sally	6	BHE	20	7.2E+08	320BB	3
CDE	-5.3	73.5	744	56	Sally	6	BHN	20	7.2E+08	320BB	3
CDE	-5.3	73.5	744	56	Sally	6	BHZ	20	7.2E+08	320BB	3

Lots of redundancy

- Even as we have eliminated the groups of values in a field, we have created a lot of rows with redundant values.
- For each channel we have put in a record, the corresponding station name is repeated.
- Also, the operator name is repeated. What would happen if the operator changed to a new person?
- We would have to edit several fields.

Functional dependency

- What we notice is that the values for **latitude**, **longitude**, **depth**, and **altitude** directly determine the station name.
- The station name is therefore the determinant field.
- If a field value can only be one possible value, based on the determinant field, then it is considered functionally dependent.

Functional dependency

- With a given station name, we can determine the latitude
- With the station name, we know the Operator

- With the station name, do we know the channel?

Functional dependency

- Since there are multiple possible channel values for a station name, we say that the channel is <u>not</u> functionally dependent
- We want to arrange our data so that in a given table, there are only functionally dependent fields in addition to the fields for the primary key.

Second normal form

- Applying the Second Normal Form to a table means that non-key fields must be functionally dependent on the key field.
- The station name is the determinant in the table, as well as the key field.

Second normal form

- Latitude, longitude, altitude, and depth can stay.
- We will also allow the operator to stay, provided there is only one per station.
- However, we must move the channel and its related fields to a new table.
- The new table will be called ‘Channels’.

Second normal form

Stations

name	lat	lon	alt	dep	operator	num_chan
ABC	32.5	102.6	1033	0	Joe	3
CDE	-5.3	73.5	744	56	Sally	6

Channels

Station	chan	samp_rate	gain	inst	stages
ABC	BHE	20	5.1E+03	CMG3	4
ABC	BHN	20	5.1E+03	CMG3	4
ABC	BHZ	20	5.1E+03	CMG3	4
CDE	LHE	1	7.2E+08	320LP	3
CDE	LHN	1	7.2E+08	320LP	3
CDE	LHZ	1	7.2E+08	320LP	3
CDE	BHE	20	7.2E+08	320BB	3
CDE	BHN	20	7.2E+08	320BB	3
CDE	BHZ	20	7.2E+08	320BB	3

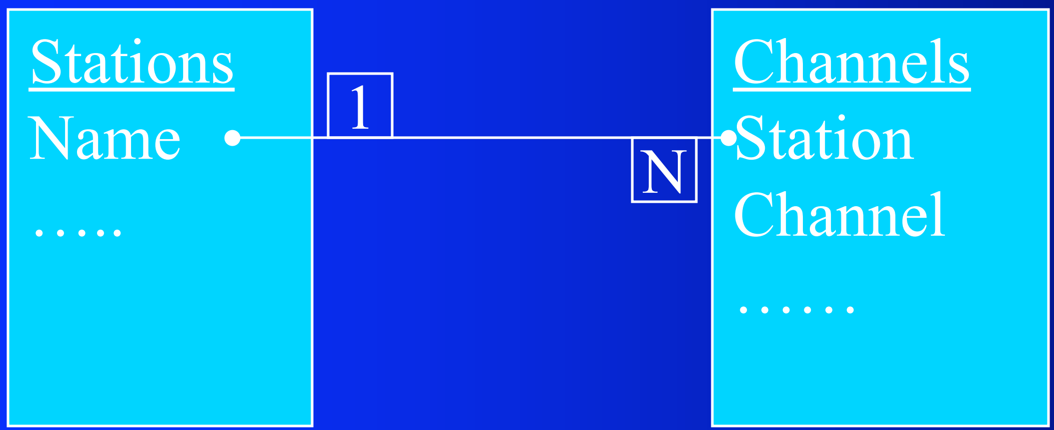
Key fields

- Note in the new table layout, we had to add a field to the Channels table called 'Station'.
- In splitting the fields into two tables, we risked breaking their association with each other.
- We need to create a foreign key in the Channels table.

Foreign key

- A foreign key is simply a primary key field moved to another table.
- The foreign key can use a different field name.
- The value, however, needs to be identical.
- The foreign key refers back to the primary key when relating the two tables.

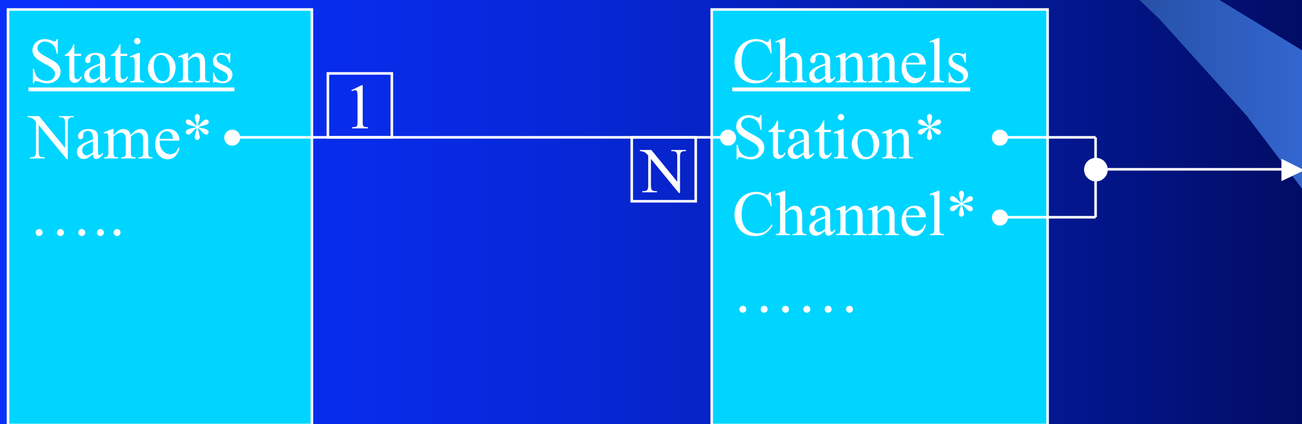
Key fields



Key fields

- You may have noticed that the Channels table doesn’t have just one primary key field.
- The primary key in this case must be more than just the channel name, because BHZ appears more than once in the ‘Channels’ table.
- However, the <u>combination</u> of **station** and **channel** names forms a unique identifier.

Key fields



- The stars signify key fields.

Multiple field key

- The primary key of a table can consist of any number of fields.
- Conditions:
  - the combination of fields is unique
  - there are no nulls in any of the fields
  - values contained in these fields shouldn't change very much

Candidate key

- It is possible to have different combinations of fields be the primary key.
- This depends on the other tables being related to.
- Any fields that can be in a primary key group is called a candidate key.

Reviewing our tables

Stations

name	lat	lon	alt	dep	operator	num_chan
ABC	32.5	102.6	1033	0	Joe	3
CDE	-5.3	73.5	744	56	Sally	6

Channels

Station	chan	samp_rate	gain	inst	stages
ABC	BHE	20	5.1E+03	CMG3	4
ABC	BHN	20	5.1E+03	CMG3	4
ABC	BHZ	20	5.1E+03	CMG3	4
CDE	LHE	1	7.2E+08	320LP	3
CDE	LHN	1	7.2E+08	320LP	3
CDE	LHZ	1	7.2E+08	320LP	3
CDE	BHE	20	7.2E+08	320BB	3
CDE	BHN	20	7.2E+08	320BB	3
CDE	BHZ	20	7.2E+08	320BB	3

More redundancies?

Channels

Station	chan	samp_rate	gain	inst	stages
ABC	BHE	20	5.1E+03	CMG3	4
ABC	BHN	20	5.1E+03	CMG3	4
ABC	BHZ	20	5.1E+03	CMG3	4
CDE	LHE	1	7.2E+08	320LP	3
CDE	LHN	1	7.2E+08	320LP	3
CDE	LHZ	1	7.2E+08	320LP	3
CDE	BHE	20	7.2E+08	320BB	3
CDE	BHN	20	7.2E+08	320BB	3
CDE	BHZ	20	7.2E+08	320BB	3

More redundancies?

Channels

Station	chan	samp_rate	gain	inst	stages
ABC	BHE	20	5.1E+03	CMG3	4
ABC	BHN	20	5.1E+03	CMG3	4
ABC	BHZ	20	5.1E+03	CMG3	4

- There are fields such as sample rate, gain, instrument name, and number of stages that repeat.

(Names and values are only for illustration)

Instrument as determinant

Channels

Station	chan	samp_rate	gain	inst	stages
ABC	BHE	20	5.1E+03	CMG3	4
ABC	BHN	20	5.1E+03	CMG3	4
ABC	BHZ	20	5.1E+03	CMG3	4

- We might say that the instrument (CMG3) determines the gain, the sample rate, and the number of stages.
- The instrument name is not a primary key.

Transitive dependency

- …is a functional dependency where
  - a non-key field is determined by the value in another non-key field
  - that field is not a candidate key
- We might use this to further refine our database tables.

Third Normal Form

- A table is in third normal form if
  - The table is in Second Normal Form
  - There are no transitive dependencies
- To make this third normal form, we need to create a new table, with the instrument name as the primary key.

New instrument table

Channels

Station	chan	inst
ABC	BHE	CMG3
ABC	BHN	CMG3
ABC	BHZ	CMG3

Instruments

Name	samp_rate	gain	stages
CMG3	20	5.1E+03	4

What we have accomplished

- Cleaner tables
- Minimal data redundancies
- Room to grow (scalability)
- Ease of editing field values

Normalization Review

- 1NF – No repeating groups.
- 2NF – Non-key fields are functionally dependent on the entire primary key.
- 3NF – No transitive dependencies.

Guidelines for Normalization

- For database normalization, good rules to follow are:
  – Look for repeating values
  – Look for fields that relate to each other
  – Determine the 'parent' field for the group
  – Make more tables and break fields into smaller functional groups

Guidelines for Primary Keys

- Use as few fields as possible.
- Make sure the field or fields provides a unique identity.
- Avoid many-to-many relationships.
- If multiple key fields gets difficult to manage, use a unique ID number as the primary key instead.

End of Part 1