

Lecture / Instructor: Leigh Cotnoir

address 1 :: 22 Tree St.

address 2 :: Apt. B

city :: Chula Vista

state ::

- Select ▼
- Alabama
- Alaska
- Arizona
- Arkansas
- California
- Colorado
- Connecticut
- Delaware

zip :: 91913

port:

Become a Member! ...choose your membership level

☒ Platinum Member:: \$300

☐ Gold Member :: \$200

MYSQL DATABASE STRUCTURE + QUERIES

Database Basics

To start, this is an example of a basic table holding data through the use of **rows** and **columns**. In terms of database terminology, the following apply:


- ▶ Each horizontal **row** represents a “record.”
- ▶ Each vertical **column** represents a “column” (also referred to as a “field”).
- ▶ Each data cell is a field cross-reference of a specific column value in a specific record.
- ▶ Each record has a “**primary key**,” or a **unique id** that no other record has.

 username	l_name	f_name	address1	address2	city	state	zip	email	type	donation
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	CA	91913	gboy@hotmail.com	platinum	300
darth	stewart	ron	9 Cate Pl.	null	San Diego	CA	92182	bigfan@gmail.com	gold	100
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	CA	91941	bandit@yahoo.com	student	25

Database Basics

To start, this is an example of a basic table holding data through the use of **rows** and **columns**. In terms of database terminology, the following apply:

- ▶ Each horizontal row represents a “record.”
- ▶ Each column represents a “column” (yes, it’s still referred to as a column).
- ▶ Each data cell is a “field,” which is a cross-reference of a specific column value in a specific record.
- ▶ Each record has a “primary key,” or a unique id that no other record has.

 username	l_name	f_name	address1	address2	city	state	zip	email	type	donation
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	CA	91913	gboy@hotmail.com	platinum	300
darth	stewart	ron	9 Cate Pl.	null	San Diego	CA	92182	bigfan@gmail.com	gold	100
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	CA	91941	bandit@yahoo.com	student	25

Database Basics

To start, this is an example of a basic table holding data through the use of **rows** and **columns**. In terms of database terminology, the following apply:

- ▶ Each row represents a “record.”
- ▶ Each vertical **column** represents a “**column**” (also referred to as a “**field**”).
- ▶ Each data cell is a “field,” which is a cross-reference of a specific column value in a specific record.
- ▶ Each record has a “primary key,” or a unique id that no other record has.

 username	l_name	f_name	address1	address2	city	state	zip	email	type	donation
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	CA	91913	gboy@hotmail.com	platinum	300
darth	stewart	ron	9 Cate Pl.	null	San Diego	CA	92182	bigfan@gmail.com	gold	100
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	CA	91941	bandit@yahoo.com	student	25

“city” is the name of the column

Database Basics

To start, this is an example of a basic table holding data through the use of **rows** and **columns**. In terms of database terminology, the following apply:

- ▶ Each row represents a “record.”
- ▶ Each column represents a “column” (yes, it’s still referred to as a column).
- ▶ Each data cell is a field cross-reference of a specific column value in a specific record.
- ▶ Each record has a “primary key,” or a unique id that no other record has.

goatboy's city = Chula Vista										
username	l_name	f_name	address1	address2	city	state	zip	email	membership	donation
 goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	CA	91913	gboy@hotmail.com	platinum	300
darth	stewart	ron	9 Cate Pl.	null	San Diego	CA	92182	bigfan@gmail.com	gold	100
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	CA	91941	bandit@yahoo.com	student	25

Database Basics

To start, this is an example of a basic table holding data through the use of **rows** and **columns**. In terms of database terminology, the following apply:

- ▶ Each row represents a “record.”
- ▶ Each column represents a “column” (yes, it’s still referred to as a column).
- ▶ Each data cell is a “field,” which is a cross-reference of a specific column value in a specific record.
- ▶ Each record has a “**primary key**,” or a **unique id** that no other record has.

username	l_name	f_name	address1	address2	city	state	zip	email	type	donation
 goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	CA	91913	gboy@hotmail.com	platinum	300
stewart	ron	9 Cate Pl.	null	San Diego	CA	92182	bigfan@gmail.com	gold	100	
smokey	doe	jane	5571st St.	Unit 219	La Mesa	CA	91941	smokey@yahoo.com	student	25

NOTE: the primary key is what we use to identify the record (row).

NOTE: the primary key is what we use to identify the record (row).

Database Basics

The example below can be considered a “flat-file” database. Here is what makes it so:

- ▶ It is in a single table, with each field separated by a delimiter (commas, tabs, etc.)
- ▶ It is not properly “normalized,” meaning that **all** the data is stored in this **one** table, and that much of the data could be considered redundant. For instance, if we had a lot of records but only 4 levels of donor support (“type”), we would inevitably repeat the 4 types AND the “donation” amounts many, many times. To be more succinct, we could separate this one table into many tables. (*next slide*)

username,	l_name,	f_name,	address1,	address2,	city,	state,	zip,	email,	type,	donation
goatboy,	doe,	john,	22 Tree St.,	Apt. B,	Chula Vista,	CA,	91913,	<u>gboy@hotmail.com,</u>	platinum,	300
darth,	stewart,	ron,	9 Cate Pl.,	null,	San Diego,	CA,	92182,	<u>bigfan@gmail.com,</u>	gold,	100
smokey,	doe,	jane,	557 1st St.,	Unit 245,	La Mesa,	CA,	91941,	<u>bandit@yahoo.com,</u>	student,	25

Database Basics :: Normalization

Normalization is where we try to remove redundancies from the table(s) where possible. This improves speed, efficiency, and data integrity. There are 3 basic “Normal Forms.” You must complete these steps in order to correct normalization.

- ▶ First Normal Form (1NF)

- ✓ ensure the removal of all redundancies in horizontal rows.
- ✓ ensure that columns hold the least amount of data possible

- ▶ Second Normal Form (2NF)

- ✓ removal of redundancies in vertical columns. This means that you must identify columns that repeat their values across multiple rows. If there are too many repeating column values, the column(s) would need to be placed in a separate table(s) and referenced by a key in the original table.


- ▶ Third Normal Form (3NF) -- typically for transactional tables only...

- ✓ Advanced. This often uses “foreign keys” and InnoDB table types. A foreign key is where a column in a table is a dependent “child” of a parent key in another table. We won’t be using this true foreign keys in the course, but we will use the **concept** of foreign keys.

Database Basics :: Normalization : 1NF

Because the previous table example was already normalized down to a 1NF level, let's back it up to see where it might have started without ANY normalization (see below).

- ▶ This table fails the 1NF level because it violates the rule that columns must have as little information as possible. Look at how the address column values contain multiple discreet components that are ganged together as one value.

 username	l_name	f_name	address	email	type	donation
goatboy	doe	john	22 Tree St., Apt. B, Chula Vista, CA, 91913	gboy@hotmail.com	platinum	300
darth	stewart	ron	9 Cate Pl., San Diego, CA, 92182	bigfan@gmail.com	gold	100
smokey	doe	jane	557 1st St., Unit 245, La Mesa, CA 91941	bandit@yahoo.com	student	25

Database Basics :: Normalization : 1NF

Because we created separate columns for discrete pieces of data that were previously in the old “address” field, we now reach 1NF. Notice in this table we have:

- ▶ The least amount of data possible in each column value
- ▶ No redundancies across the horizontal rows/records.

....Okay, so let's go to the next slide to check for 2NF.

 username	l_name	f_name	address1	address2	city	state	zip	email	type	donation
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	CA	91913	gboy@hotmail.com	platinum	300
darth	stewart	ron	9 Cate Pl.	null	San Diego	CA	92182	bigfan@gmail.com	gold	100
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	CA	91941	bandit@yahoo.com	student	25



Database Basics :: Normalization : 2NF


Let's say that in our logic model, each member would have 4 different levels of membership to choose from, and each level has a minimum financial contribution requirement. Well, if we expect to have even 5 member records, we would have a repetition of values in both the "type" AND "donation" columns. This table fails the 2NF level as a result. The "state" column suffers the same fate, too, which we will address as well. To fix it, we'd need to do the following:

- ▶ Put the "type" and "donation" columns into one new table. Leave a "type" column in this member table to reference the new table we are making. Let's see how that looks in the next slide....
- ▶ Also, because the "state" column could have a finite set of values (50 states), any of which could be repeated multiple times across the rows, it can be placed in its own table.

 username	l_name	f_name	address1	address2	city	state	zip	email	type	donation
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	CA	91913	gboy@hotmail.com	platinum	300
darth	stewart	ron	9 Cate Pl.	null	San Diego	CA	92182	bigfan@gmail.com	gold	100
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	CA	91941	bandit@yahoo.com	student	25

Database Basics :: Normalization : 2NF

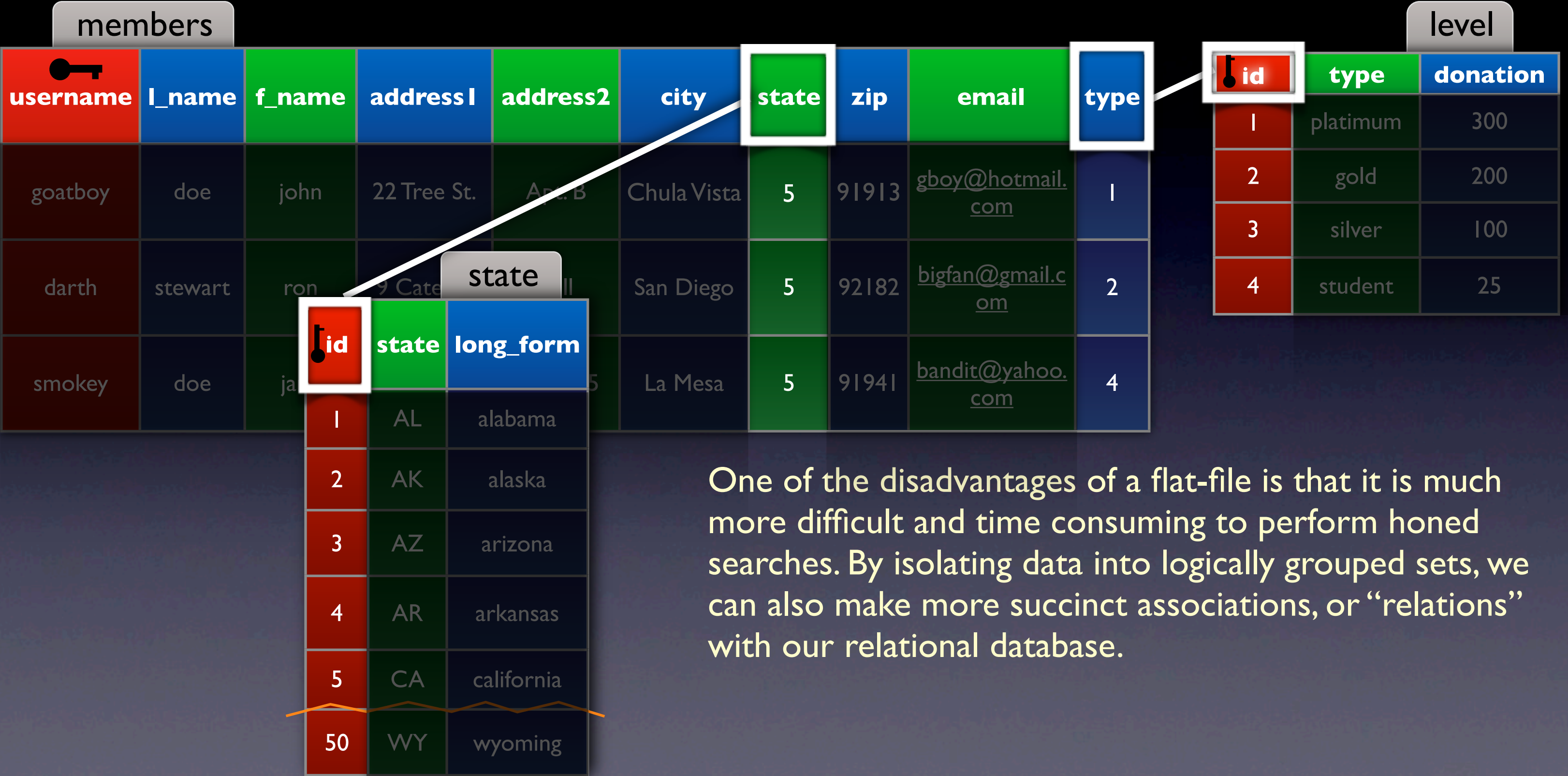
members										level		
 username	l_name	f_name	address1	address2	city	state	zip	email	type	 id	type	donation
goatboy	doe	john	22 Tree St.	Appt. B	Chula Vista	5	91913	gboy@hotmail.com	1	1	platinum	300
darth	stewart	ron	9 Cate	ll	San Diego	5	92182	bigfan@gmail.com	2	2	gold	200
smokey	doe	ja			La Mesa	5	91941	bandit@yahoo.com	4	3	silver	100
										4	student	25

 id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

Advantages of normalizing to 2NF are:

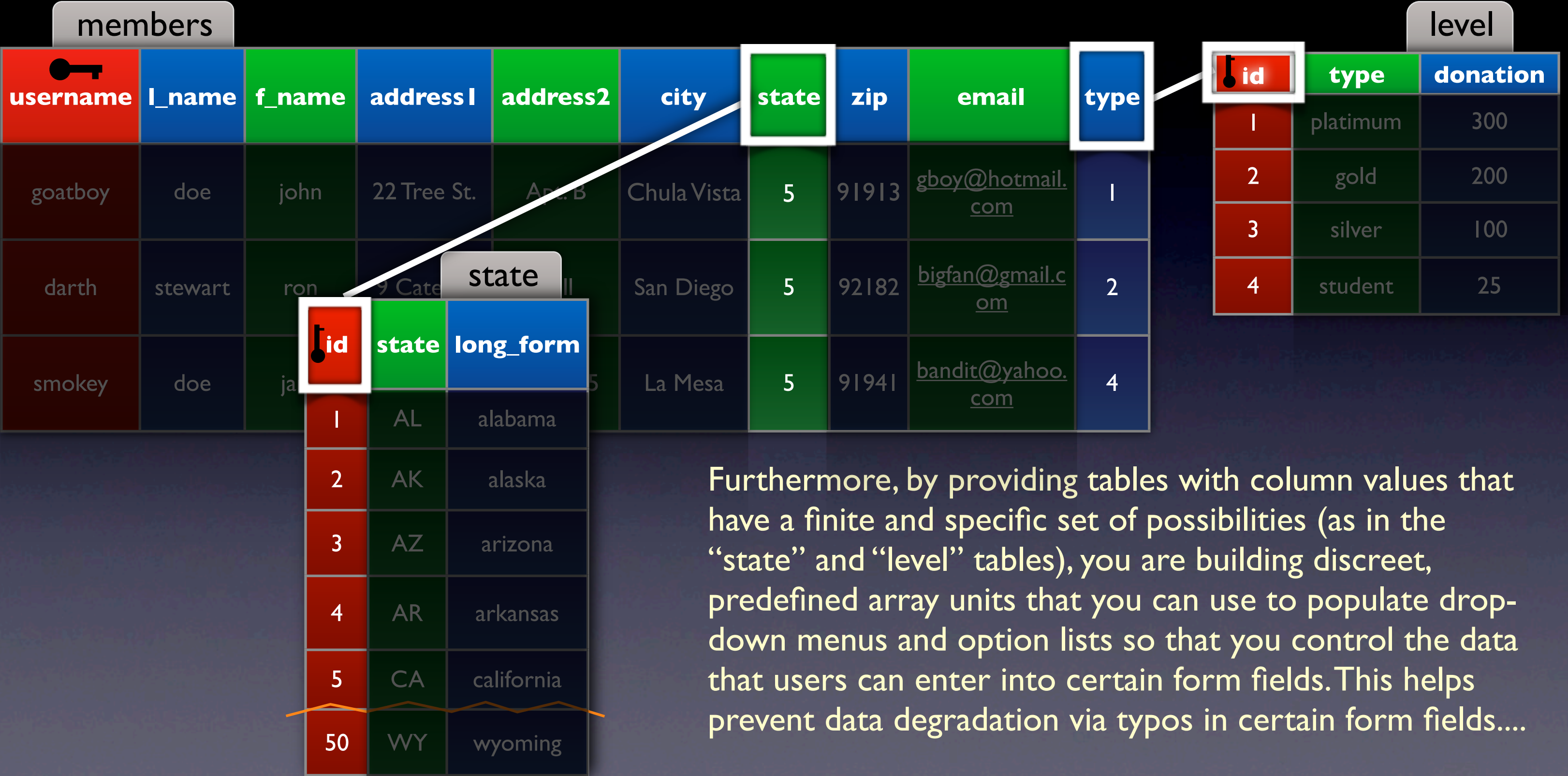
1. greater search abilities
2. mapping more sophisticated associations
3. better data integrity

Database Basics :: Normalization : 2NF



One of the disadvantages of a flat-file is that it is much more difficult and time consuming to perform honed searches. By isolating data into logically grouped sets, we can also make more succinct associations, or “relations” with our relational database.

Database Basics :: Normalization : 2NF



Database Basics :: Normalization : 2NF

members

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	gboy@hotmail.com	1
darth	stewart	ron	9 Cate	ll	San Diego	5	92182	bigfan@gmail.com	2
smokey	doe	ja			La				

level

id	type	donation
1	platinum	300
2	gold	200
3	silver	100
4	student	25

state

id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

Form

address 1 :: 22 Tree St.

address 2 :: Apt. B

city :: Chula Vista

state ::

Select

zip :: 91913

Become a Member! ...choose yo

☒ Platinum Member:: \$300

☐ Gold Member :: \$200

port:

Database Basics :: Normalization : 2NF

members

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	gboy@hotmail.com	1
darth	stewart	ron	9 Cate		San Diego	5	92182	bigfan@gmail.com	2
smokey	doe	jan			La	5			

level

id	type	donation
1	platinum	300
2	gold	200
3	silver	100
4	student	25

state

id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

Form Fields:

address 1 ::

address 2 ::

city ::

state ::

Select ▼

Alabama

Alaska

Arizona

Arkansas

California

Colorado

Connecticut

Delaware

zip ::

Become a Member! ...choose your support:

☒ Platinum Member:: \$300

☐ Gold Member :: \$200

The option select list is being pulled from the “state” table so that users must select from a list of states for form input. This helps keep data consistent.

Database Basics :: Normalization : 2NF

members

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	gboy@hotmail.com	1
darth	stewart	ron	9 Cate	ll	San Diego	5	92182	bigfan@gmail.com	2
smokey	doe	jan			La				

level

id	type	donation
1	platinum	300
2	gold	200
3	silver	100
4	student	25

state

id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

address 1 :: 22 Tree St.

address 2 :: Apt. B

city :: Chula Vista

state ::

Select ▼

AlabamaAlaskaArizonaArkansasCaliforniaColoradoConnecticutDelaware

zip :: 91913

Become a Member! ...choose yo

☒ Platinum Member:: \$300

☐ Gold Member :: \$200

port:

The radio button set of options is being pulled from the “level” table. Values are pulled out as array items and listed through a looping cycle.

Database Basics :: Normalization : 2NF

members										level		
 username	l_name	f_name	address1	address2	city	state	zip	email	type	 id	type	donation
goatboy	doe	john	22 Tree St.	Appt. B	Chula Vista	5	91913	gboy@hotmail.com	1	1	platinum	300
darth	stewart	ron	9 Cate	ll	San Diego	5	92182	bigfan@gmail.com	2	2	gold	200
smokey	doe	ja			La Mesa	5	91941	bandit@yahoo.com	4	3	silver	100
										4	student	25

 id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

Okay, so you might we wondering, “how do we get this data out of the database and onto the webpage?”

We have to run “queries” against the database. Queries are simply commands that manipulate the data in one way or another. The most common query you are likely to run is one using the “SELECT” command. Let’s take a look...

Database Basics :: Normalization : 2NF



Database Basics :: Normalization : 2NF / SELECT

Let's start by grabbing all the column values for every record in the “members” table. You would use this query:

```
SELECT * FROM members;
```

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	<u>gboy@hotmail.com</u>	1
darth	stewart	ron	9 Cate Pl.	null	San Diego	5	92182	<u>bigfan@gmail.com</u>	2
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	5	91941	<u>bandit@yahoo.com</u>	4

The asterisk in the statement is the “wildcard” symbol, meaning “everything.” So, in effect, we are saying:

“Select all columns from the members table.” Above is what the query will return.

Database Basics :: Normalization : 2NF / SELECT

Now let's select specific columns:

```
SELECT username, l_name, f_name FROM members;
```

username	l_name	f_name
goatboy	doe	john
darth	stewart	ron
smokey	doe	jane

This is a more efficient way to query the database if all you really need are these columns. It keeps the server from having to do extensive searches for a bunch of data you don't really care about at that time anyway.

Database Basics :: Normalization : 2NF / WHERE

Now let's select specific columns that match certain criteria using WHERE:

```
SELECT username, l_name, f_name, city FROM members WHERE username='smokey';
```

username	l_name	f_name	city
smokey	doe	jane	La Mesa

A query like this might be run when a user clicks on a link from a list of user names to get more info about the user.

Database Basics :: Normalization : 2NF / LIKE

Let's go back to the full members table so we can see all the records:

```
SELECT * FROM members;
```

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	<u>gboy@hotmail.com</u>	1
darth	stewart	ron	9 Cate Pl.	null	San Diego	5	92182	<u>bigfan@gmail.com</u>	2
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	5	91941	<u>bandit@yahoo.com</u>	4

Now let's do a search matching values to limited criteria with some unknown values, using LIKE:

```
SELECT * FROM members WHERE l_name LIKE 'd%';
```

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	<u>gboy@hotmail.com</u>	1
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	5	91941	<u>bandit@yahoo.com</u>	4

Database Basics :: Normalization : 2NF / LIKE ...%

The MySQL 'LIKE' operator is useful in doing searches where you only know part of what you are looking for. As you saw in the previous slide, the % sign represents the unknown part of the pattern. Look at the examples below of ways we can use this operator to find information:

LIKE uses % to indicate missing pattern (not case-sensitive)

- '%.gif' would search for patterns ending in ".gif"
- 'n%' would search for anything starting with the letter "n"
- '%tra%' searches for anything with the pattern "tra" inside if it at any position. It could return "tramp," "Sinatra," "intravenous" etc. if those words were in the search columns.

Database Basics :: Normalization : 2NF / ORDER BY ... LIMIT

We can also use an ORDER BY command to filter query results in ascending or descending order. Furthermore, we can LIMIT the number of records that are returned in the query. Let's look at the example below:

Reminder: full 'state' table:

id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

```
SELECT long_form  
FROM state  
ORDER BY long_form DESC  
LIMIT 7;
```

Notice that the statement above is separated out on different lines. Like in HTML, MySQL statements can be separated out on different lines before the statement is complete without throwing an error. It knows when to end the statement when it sees the semicolon terminator at the end.

long_form
wyoming
wisconsin
west virginia
washington
virginia
vermont
utah

Database Basics :: Normalization : 2NF / INSERT INTO

Let's go back to the full members table so we can see all the records:

```
SELECT * FROM members;
```

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	gboy@hotmail.com	1
darth	stewart	ron	9 Cate Pl.	null	San Diego	5	92182	bigfan@gmail.com	2
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	5	91941	bandit@yahoo.com	4

Now let's insert a record. Look at the syntax below along with the results, and we'll discuss it in the next slide:

```
INSERT INTO members VALUES ('lcotnoir','cotnoir','leigh','555 Pacific Ocean','#2','San Diego','5','92182','lcotnoir@mail.sdsu.edu','1');
```

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	gboy@hotmail.com	1
darth	stewart	ron	9 Cate Pl.	null	San Diego	5	92182	bigfan@gmail.com	2
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	5	91941	bandit@yahoo.com	4
lcotnoir	cotnoir	leigh	555 Pacific Ocean	#2	San Diego	5	92182	lcotnoir@mail.sdsu.edu	1

Database Basics :: Normalization : 2NF / INSERT INTO

INSERT INTO members **VALUES** ('lcotnoir','cotnoir','leigh','555 Pacific Ocean','#2','San Diego','5','92182','lcotnoir@mail.sdsu.edu','1');

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Apt. B	Chula Vista	5	91913	gboy@hotmail.com	1
darth	stewart	ron	9 Cate Pl.	null	San Diego	5	92182	bigfan@gmail.com	2
smokey	doe	jane	557 1st St.	Unit 245	La Mesa	5	91941	bandit@yahoo.com	4
lcotnoir	cotnoir	leigh	555 Pacific Ocean	#2	San Diego	5	92182	lcotnoir@mail.sdsu.edu	1

This command inserts **NEW** column values into a **NEW** row(s). An example of when you would use this type of query is when someone fills out a web form, whose data you want to store in a database. This could be anything: setting up a user account, buying goods, etc.

It is important that you have a place in the syntax for each field/column, or else the values will be inserted into the wrong columns. The order of the values (see below) is identical to the way that the table's columns are ordered.

You do **NOT** use this command to merely update an existing row. For that, use **UPDATE**.

Database Basics :: Normalization : 2NF / UPDATE, DELETE

Some other important commands are UPDATE and the DELETE statements. The UPDATE command changes values in already-existing records, and the delete command will delete already-existing records. Let's look at some syntax examples:

UPDATE Syntax:

```
UPDATE table_name SET column_name = 'some_value' WHERE some_column = 'some_value';
```

multiple values at once:

```
UPDATE table_name SET columnA = 'some_value', columnB = 'some_value' WHERE some_column = 'some_value';
```

DELETE Syntax:

```
DELETE FROM table_name WHERE some_column = 'some_value';
```

Database Intermediate :: Normalization : 2NF / JOIN

members

username	l_name	f_name	address1	address2	city	state	zip	email	type
goatboy	doe	john	22 Tree St.	Appt. B	Chula Vista	5	91913	gboy@hotmail.com	1
darth	stewart	ron	9 Cate	ll	San Diego	5	92182	bigfan@gmail.com	2
smokey	doe	ja			La Mesa	5	91941	bandit@yahoo.com	4

state



id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming


level

id	type	donation
1	platinum	300
2	gold	200
3	silver	100
4	student	25

Let's go back to our original set of 2NF tables. If we want to be able to access all the values in the "state" table while we are also accessing the values in the "members" table, we can do this by using a **JOIN** in our query. What the JOIN command does is temporarily join multiple tables' columns into one merged table for the life of the query. This join is not permanent, so you have to run the join query each time you want merged results. (next slide)

Database Intermediate :: Normalization : 2NF / JOIN

members										level		
 username	l_name	f_name	address1	address2	city	state	zip	email	type	 id	type	donation
goatboy	doe	john	22 Tree St.	Appt. B	Chula Vista	5	91913	gboy@hotmail.com	1	1	platinum	300
darth	stewart	ron	9 Cate	ll	San Diego	5	92182	bigfan@gmail.com	2	2	gold	200
smokey	doe	ja			La Mesa	5	91941	bandit@yahoo.com	4	3	silver	100
										4	student	25

 id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

Let's get values from all three tables above.

```
SELECT members.l_name, members.f_name,
members.city, members.zip, state.state, level.type,
level.donation
FROM members, state, level
WHERE (members.state = state.id) AND (members.type =
level.id);
```

Database Intermediate :: Normalization : 2NF / JOIN

l_name	f_name	city	state	type	donation
doe	john	Chula Vista	CA	platinum	300
stewart	ron	San Diego	CA	gold	200
doe	jane	La Mesa	CA	student	25

```
SELECT members.l_name, members.f_name,  
members.city, members.zip, state.state,  
level.type, level.donation  
FROM members, state, level  
WHERE (members.state = state.id)  
AND (members.type = level.id);
```

Notice that the new merged table generated by the query does not display those representative numerically keyed values in the 'type' and 'state' columns. Before, the members table held a value of "5" to represent the primary key of CA in the state table. Now you see the state column as 'CA'.

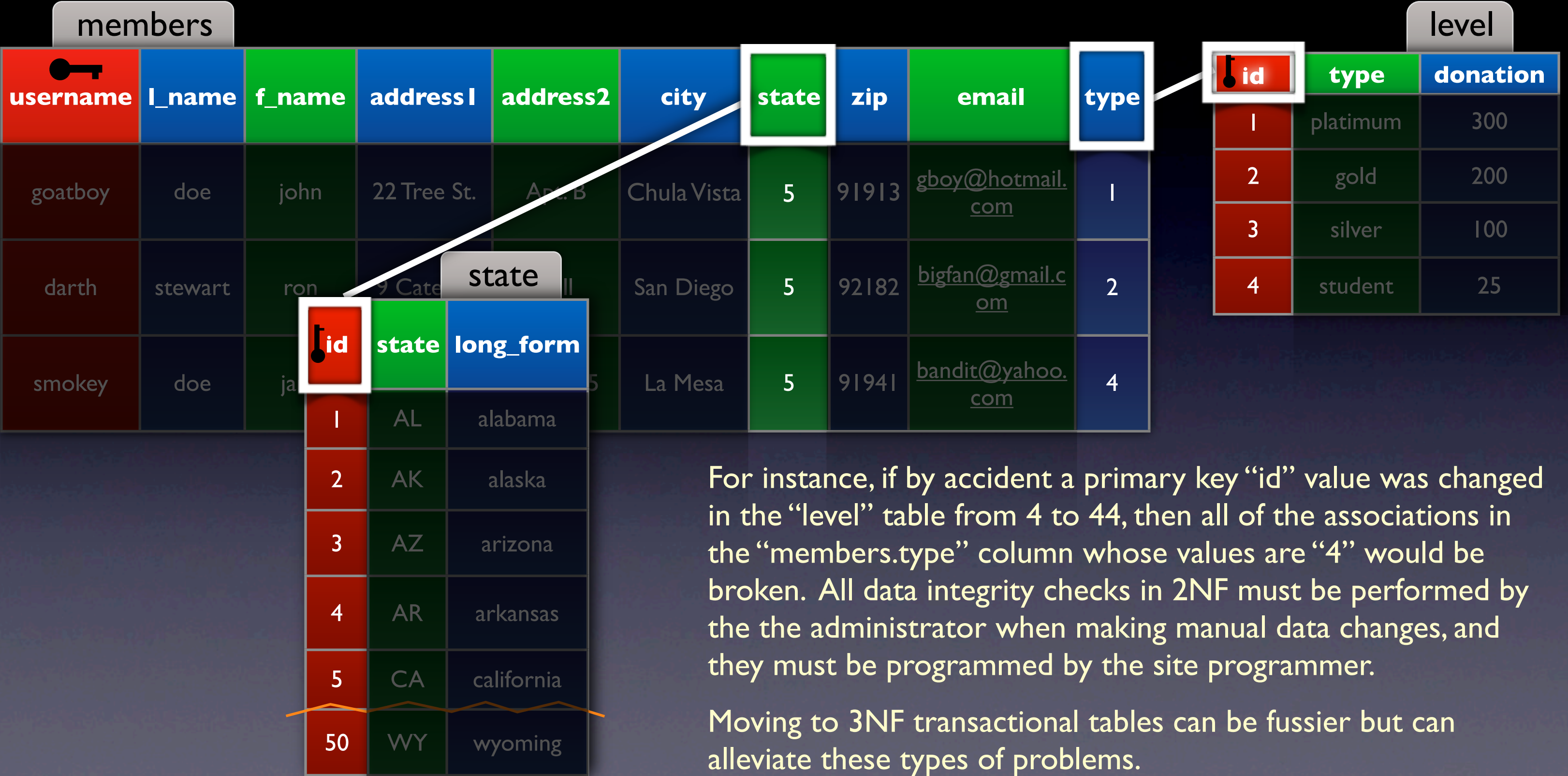
This is because the query grabs the columns it wants to display in the SELECT statement by using the 'table.column' syntax, and the FROM clause helps us finish mapping the tables from which to pull. Lastly, we know how to get exactly the right associations by matching criteria in the WHERE clause. This is where we map the numerically keyed columns from members 'state' and 'type' fields to the state and level tables' primary keys.

Database Advanced :: Normalization : 2NF to 3NF

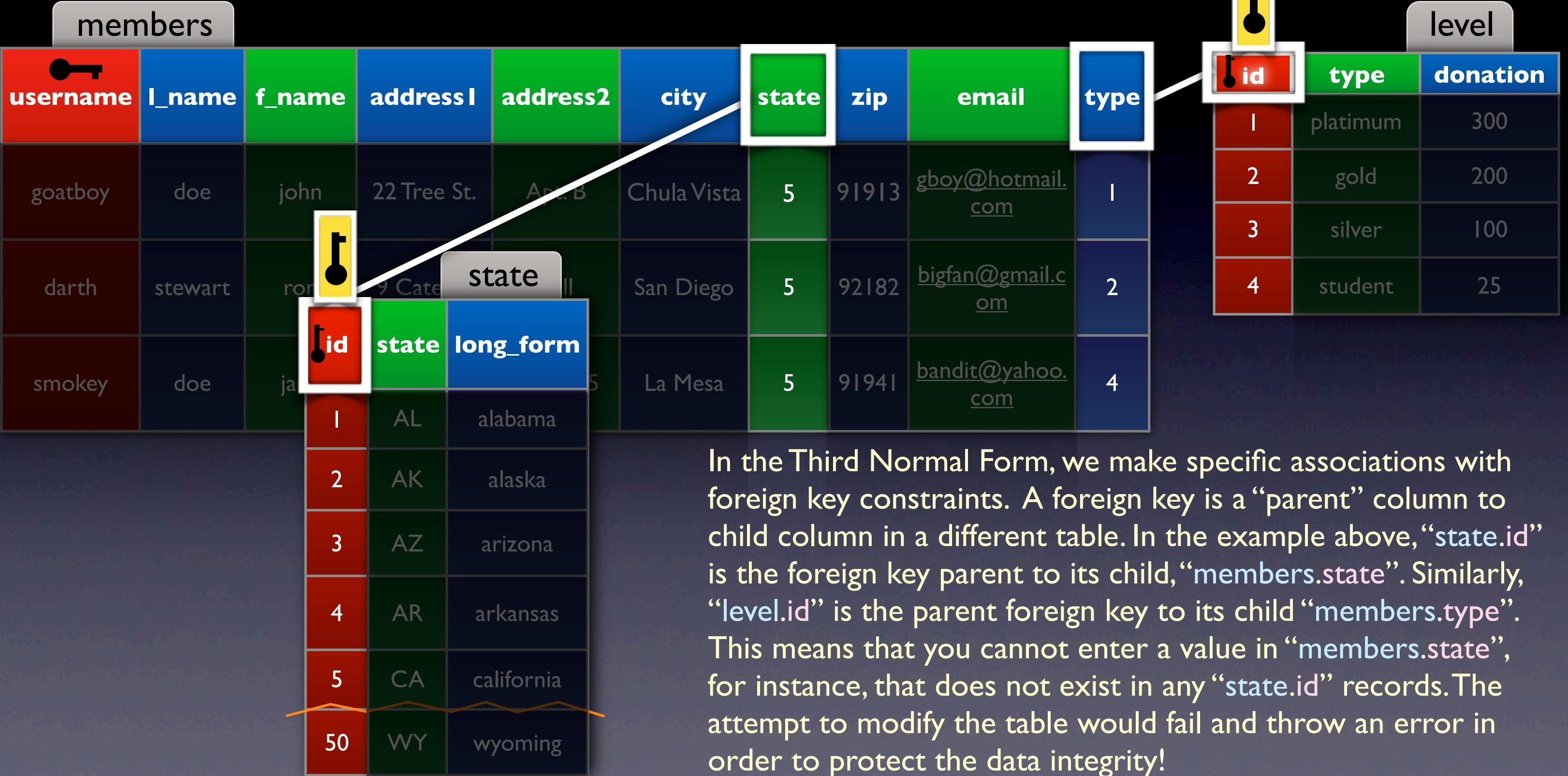


Now that we've discussed joins, let's touch on data integrity. You can see from the previous join example that if any of the keyed data associations were entered incorrectly into the database, your ability to effectively find the data would start to disintegrate. This is the danger of not using transactional InnoDB tables in 3NF form. Nothing is preventing the site administrator from making a data entry mistake that might damage the keyed associations.

Database Advanced :: Normalization : 2NF to 3NF

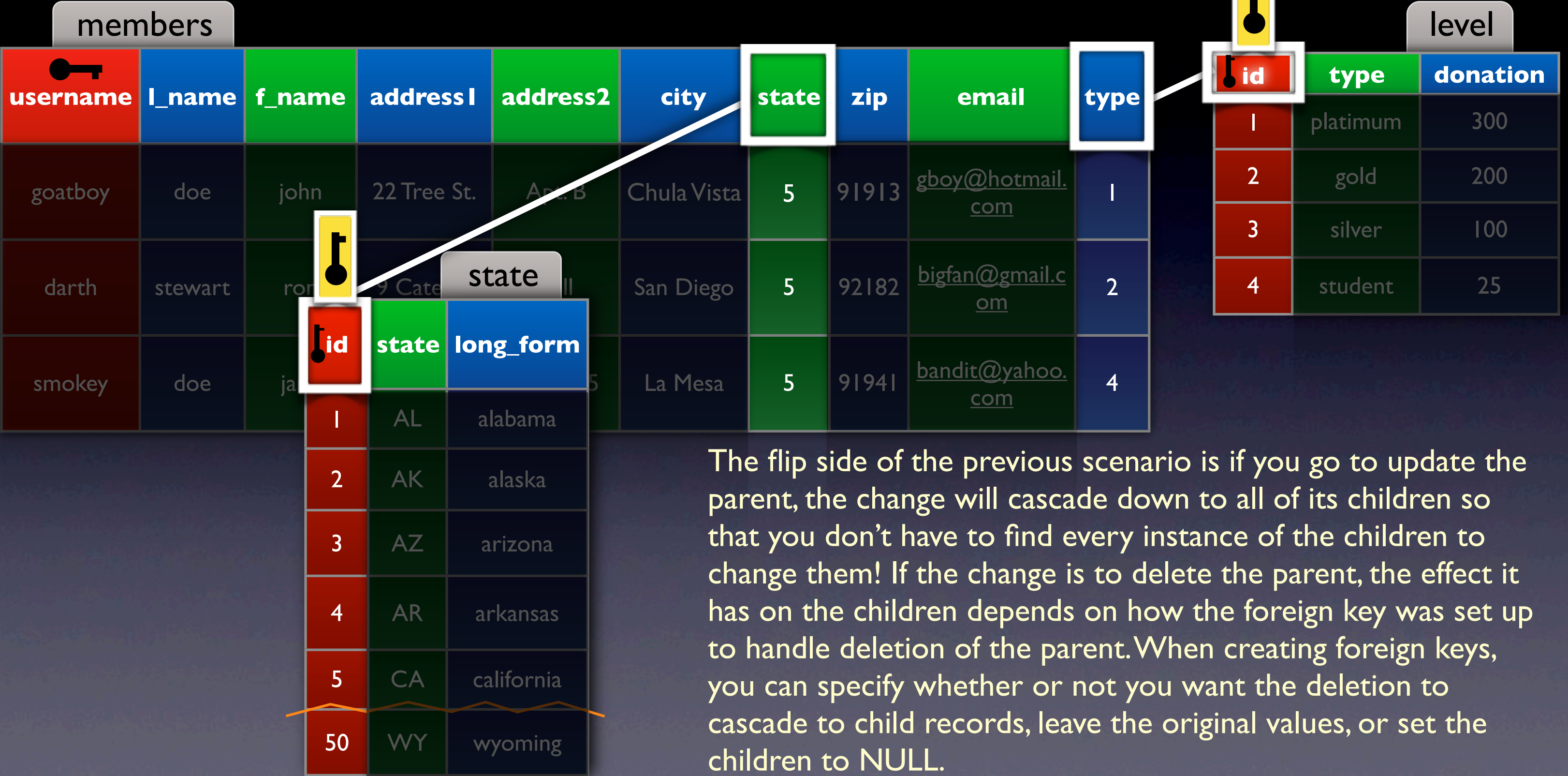


Database Advanced :: Normalization : 3NF / Foreign Keys





In the Third Normal Form, we make specific associations with foreign key constraints. A foreign key is a “parent” column to child column in a different table. In the example above, “state.id” is the foreign key parent to its child, “members.state”. Similarly, “level.id” is the parent foreign key to its child “members.type”. This means that you cannot enter a value in “members.state”, for instance, that does not exist in any “state.id” records. The attempt to modify the table would fail and throw an error in order to protect the data integrity!


Database Advanced :: Normalization : 3NF / Foreign Keys



The flip side of the previous scenario is if you go to update the parent, the change will cascade down to all of its children so that you don't have to find every instance of the children to change them! If the change is to delete the parent, the effect it has on the children depends on how the foreign key was set up to handle deletion of the parent. When creating foreign keys, you can specify whether or not you want the deletion to cascade to child records, leave the original values, or set the children to NULL.

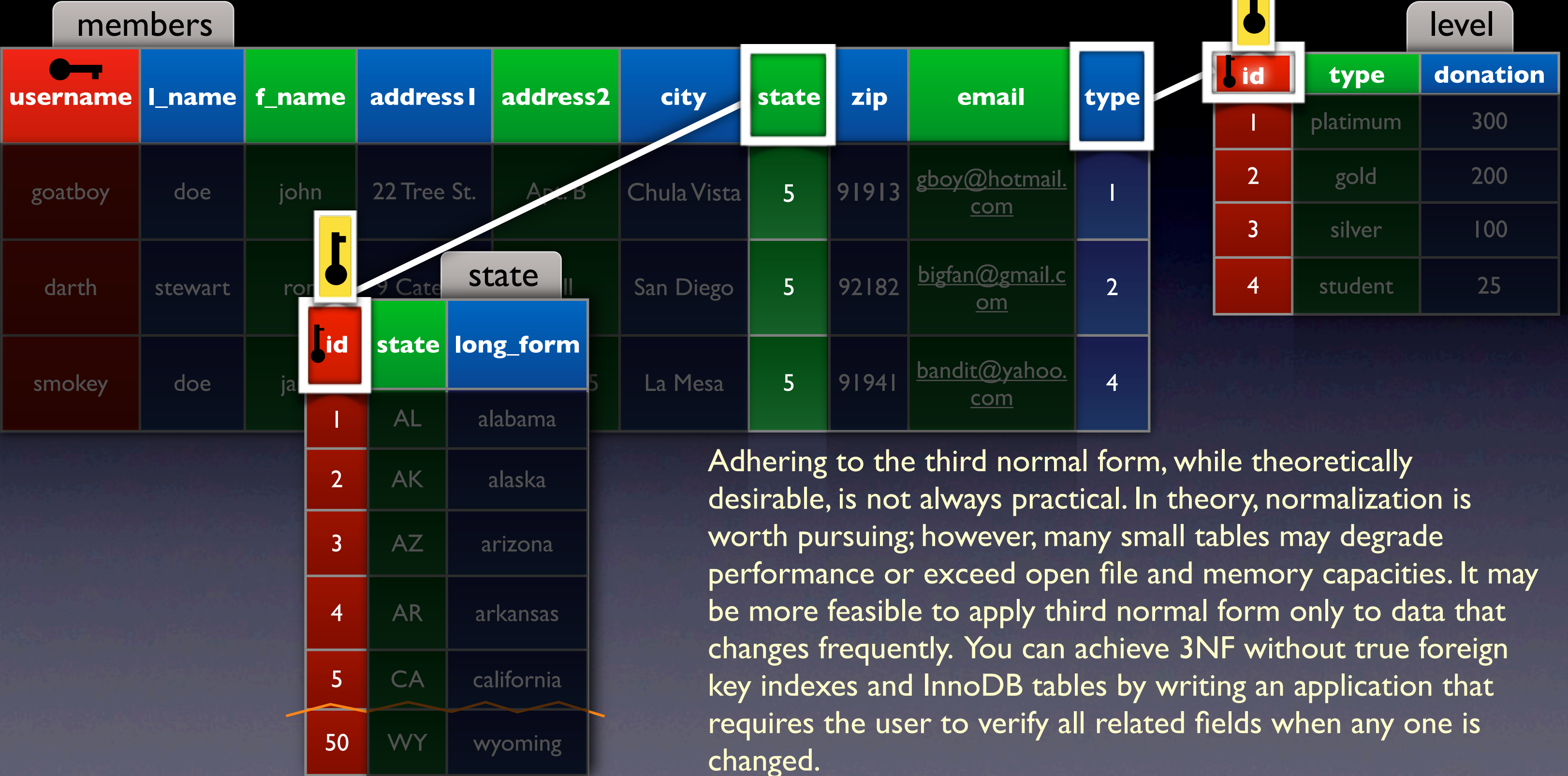
Database Advanced :: Normalization : 3NF / Foreign Keys

members										level		
 username	l_name	f_name	address1	address2	city	state	zip	email	type	 id	type	donation
goatboy	doe	john	22 Tree St.	Appt. B	Chula Vista	5	91913	gboy@hotmail.com	1	1	platinum	300
darth	stewart	ron	9 Cate	Ill	San Diego	5	92182	bigfan@gmail.com	2	2	gold	200
smokey	doe	ja			La Mesa	5	91941	bandit@yahoo.com	4	3	silver	100
										4	student	25

 id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming



Foreign keys only work in transactional tables. A “transaction” means that when one change happens, a second must also occur as a consistency check. MySQL uses MyISAM tables by default (which are NOT transactional and do not support foreign keys), but you can set MySQL up to use InnoDB type tables if you want foreign key support. It is best to design a database with these goals in mind and use the right tables from the beginning. Because foreign keys are advanced, we will not extensively cover them in this class.

Database Advanced :: Normalization : 3NF / Foreign Keys



Adhering to the third normal form, while theoretically desirable, is not always practical. In theory, normalization is worth pursuing; however, many small tables may degrade performance or exceed open file and memory capacities. It may be more feasible to apply third normal form only to data that changes frequently. You can achieve 3NF without true foreign key indexes and InnoDB tables by writing an application that requires the user to verify all related fields when any one is changed.

Database Advanced :: Normalization : 3NF / Foreign Keys

members										level		
 username	l_name	f_name	address1	address2	city	state	zip	email	type	 id	type	donation
goatboy	doe	john	22 Tree St.	Appt. B	Chula Vista	5	91913	gboy@hotmail.com	1	1	platinum	300
darth	stewart	ron	9 Cate	Ill	San Diego	5	92182	bigfan@gmail.com	2	2	gold	200
smokey	doe	ja			La Mesa	5	91941	bandit@yahoo.com	4	3	silver	100
										4	student	25

 id	state	long_form
1	AL	alabama
2	AK	alaska
3	AZ	arizona
4	AR	arkansas
5	CA	california
50	WY	wyoming

Depending on how large the database is and how many tables it requires, building “physical” constraints with foreign key indexes can slow down database performance. Creating actual foreign keys indexes in MySQL means that MySQL polices the queries.

You can still design the tables with conceptual foreign key indexes, but you would use your PHP application to enforce foreign keys based on how queries are written instead of relying on MySQL to enforce the rules.