ALTER

Bok, Jong Soon
Jongsoon.bok@gmail.com
www.javaexpert.co.kr

Alter Table

```
ALTER TABLE table_name
| ADD COLUMN col_name column_definition
        [FIRST| AFTER  col_name]
| ADD [CONSTRAINT] PRIMARY KEY
| ADD [CONSTRAINT] UNIQUE [INDEX | KEY ]
| ADD [CONSTRAINT] FOREIGN KEY
| ALTER [COLUMN] col_name {SET DEFAULT value |
                                DROP DEFAULT}
| CHANGE [COLUMN] old_col_name new_col_name column_definition
         [FIRST | AFTER col_name]
| MODIFY [COLUMN] col_name column_definition
         [FIRST | AFTER col_name]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP FOREIGN KEY
| RENAME [TO] new_table_name
```

The **CREATE TABLE** we wrote

```
CREATE TABLE my_contacts
(
    last_name VARCHAR(30),
    first_name VARCHAR(20),
    email VARCHAR(50),
    gender CHAR(1),
    birthday DATE,
    profession VARCHAR(50),
    location VARCHAR(50),
    status VARCHAR(20),
    interests VARCHAR(100),
    seeking VARCHAR(100)
);
```

No primary key here

Could these columns have been more atomic when we set up the table?

Show me the table

```
SHOW CREATE TABLE my_contacts;
```

```
--------------------+
| my_contacts | CREATE TABLE `my_contacts` (
  `last_name` varchar(30) NOT NULL,
  `first_name` varchar(20) NOT NULL,
  `email` varchar(50) NOT NULL,
  `gender` char(1) NOT NULL,
  `birthday` date NOT NULL,
  `profession` varchar(50) NOT NULL,
  `location` varchar(50) NOT NULL,
  `status` varchar(20) NOT NULL,
  `interests` varchar(100) NOT NULL,
  `seeking` varchar(100) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
+-------------+-----------------------------------
```

Time-saving command

The marks around the column names and the table name are called backticks. They show up when we run the SHOW CREATE TABLE command.

Unless we tell the SQL software differently, it assumes all values are NULL by default.

It's a good idea to specify if a column can contain NULL or not when we create our table.

```
CREATE TABLE `my_contacts`
(
    `last_name` varchar(30) default NULL,
    `first_Name` varchar(20) default NULL,
    `email` varchar(50) default NULL,
    `gender` char(1) default NULL,
    `birthday` date default NULL,
    `profession` varchar(50) defaul
    `location` varchar(50) default
    `status` varchar(20) default
    `interests` varchar(100) def
    `seeking` varchar(100) defau
) ENGINE=MyISAM DEFAULT CHARSET=latin
```

Although you could make the code neater (by removing the last line and backticks), you can just copy and paste it to create a table.

Adding a **PRIMARY KEY** to an existing table

```
ALTER TABLE my_contacts
ADD COLUMN contact_id INT
                    NOT NULL
                    AUTO_INCREMENT
                    FIRST,
ADD PRIMARY KEY (contact_id) ;
```

Adding a **PRIMARY KEY** to an existing table (Cont.)

File Edit Window Help Alterations

```
> ALTER TABLE my_contacts
    -> ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (contact_id);

Query OK, 50 rows affected (0.04 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

```
mysql> ALTER TABLE my_contacts
    -> ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (contact_id);
Query OK, 1 row affected (0.04 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

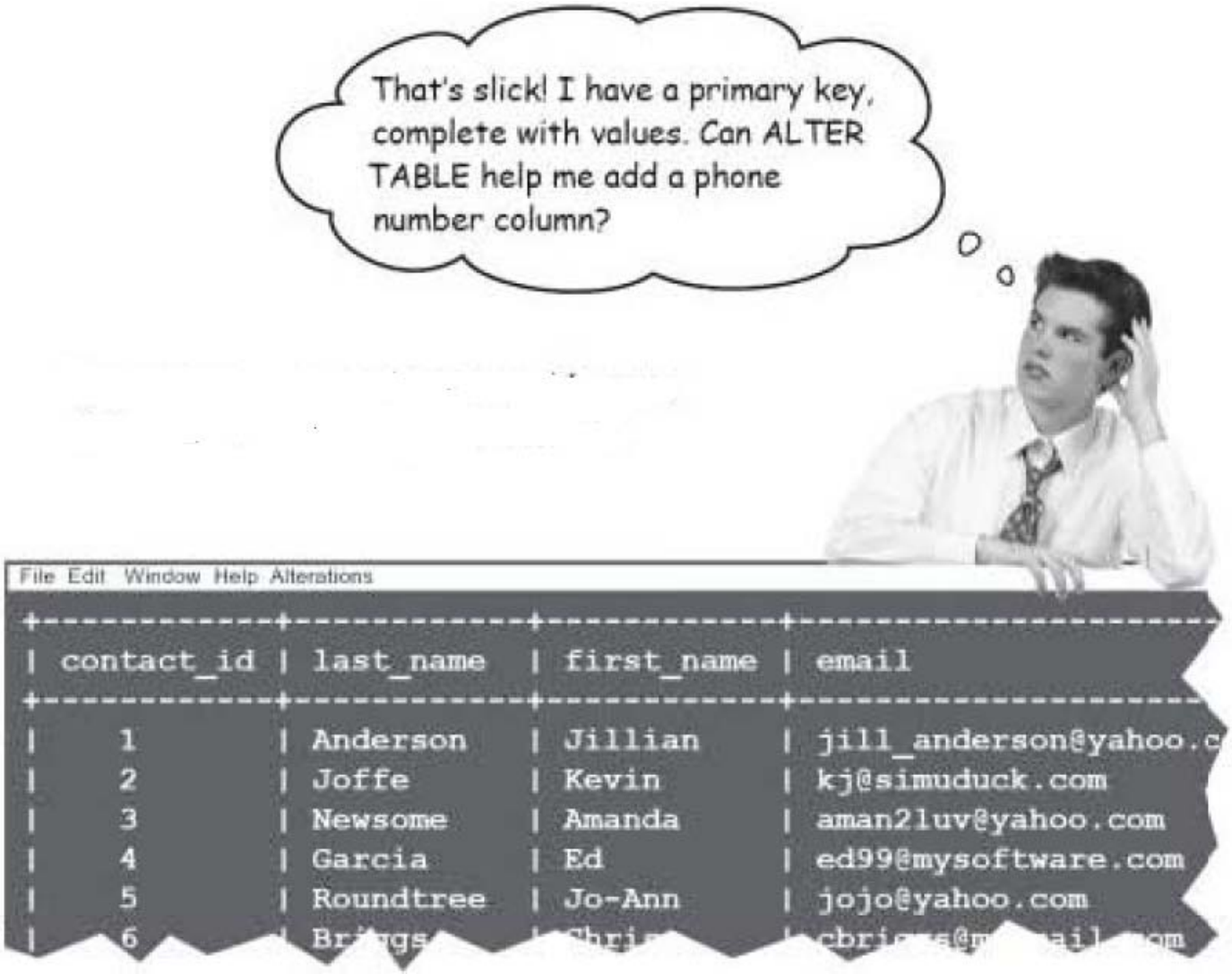
Adding a **PRIMARY KEY** to an existing table (Cont.)

SHOW COLUMNS FROM table_name;

DESC table_name;

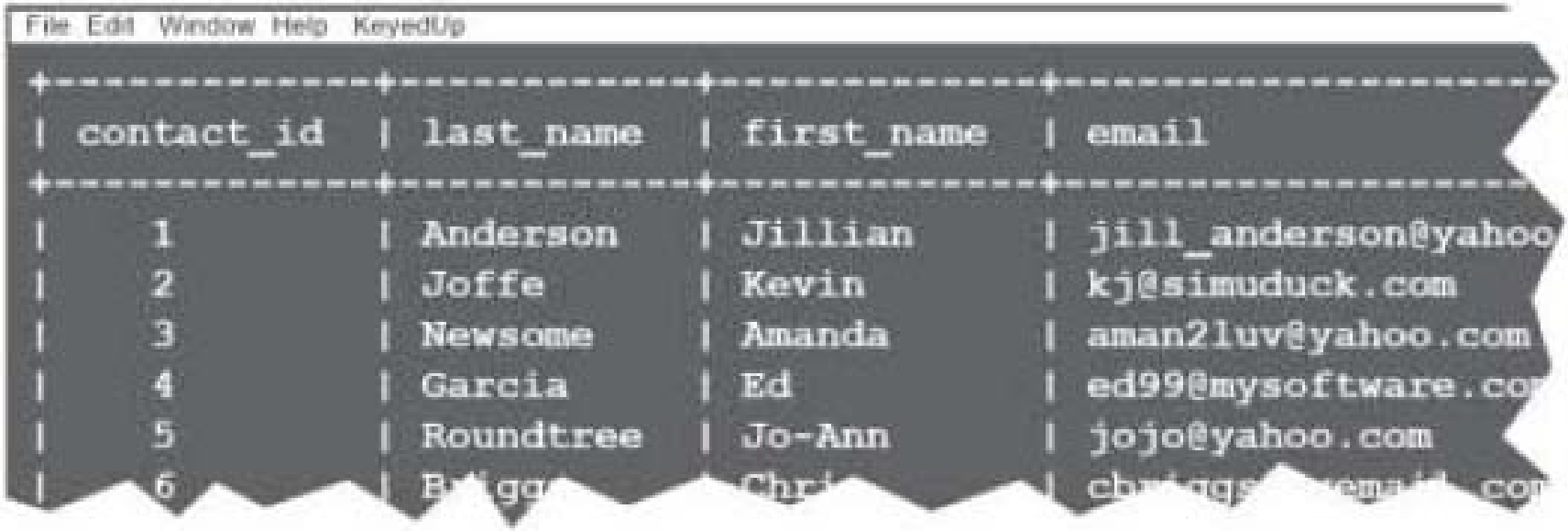
```
mysql> show columns from my_contacts;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| contact_id | int(11)     | NO   | PRI | NULL    | auto_increment |
| last_name  | varchar(30) | NO   |     | NULL    |                |
| first_name | varchar(20) | NO   |     | NULL    |                |
| email      | varchar(50) | NO   |     | NULL    |                |
| gender     | char(1)     | NO   |     | NULL    |                |
| birthday   | date        | NO   |     | NULL    |                |
| profession | varchar(50) | NO   |     | NULL    |                |
| location   | varchar(50) | NO   |     | NULL    |                |
| status     | varchar(20) | NO   |     | NULL    |                |
| interests  | varchar(100)| NO   |     | NULL    |                |
| seeking    | varchar(100)| NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)
```

Adding a **PRIMARY KEY** to an existing table (Cont.)

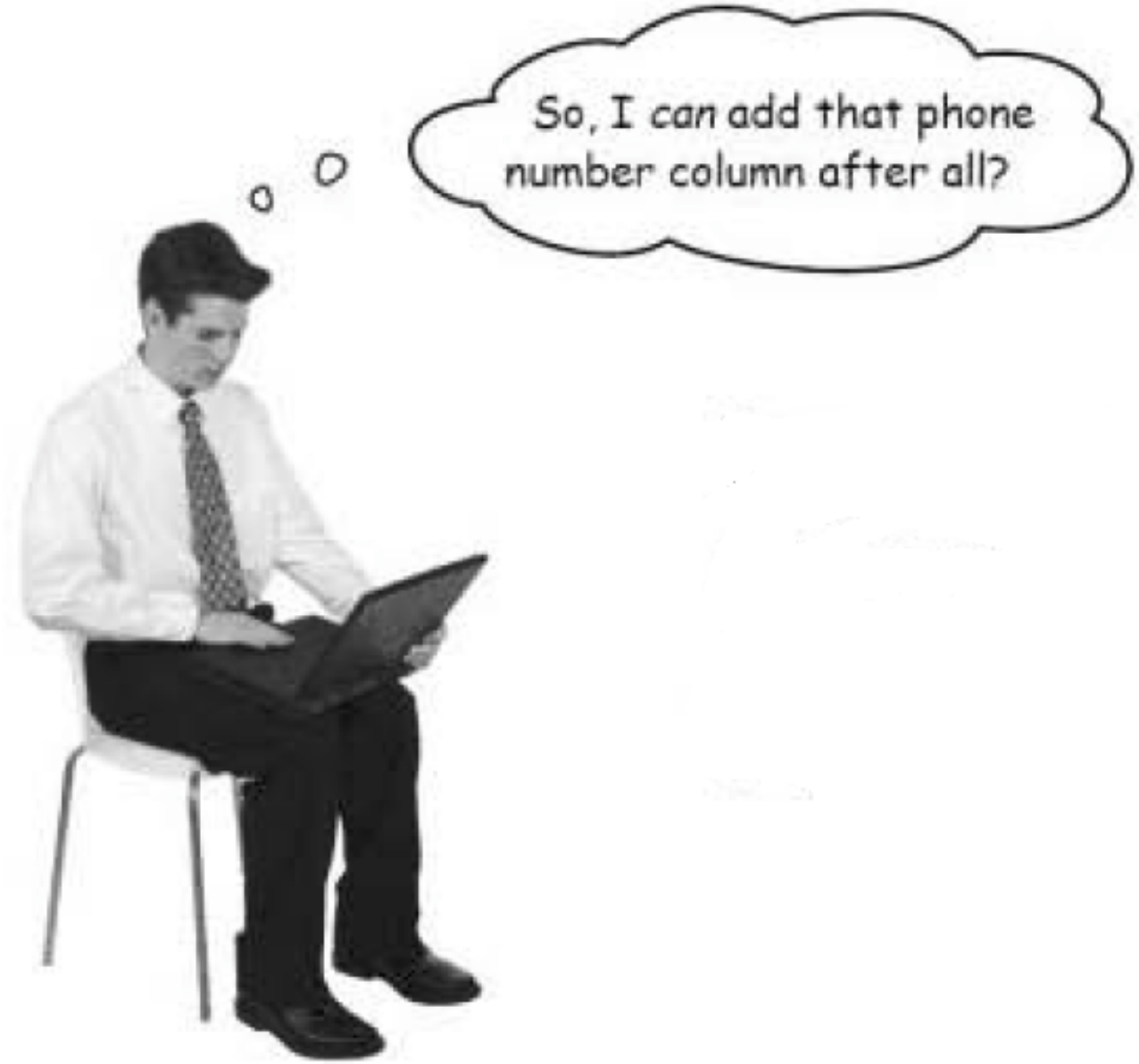


We need to make some changes

- Greg wants to make a few more changes to his table, but he doesn't want to lose any data.



We need to make some changes(Cont.)



- To add a phone column that can hold 10 digits. Location is right after the first_name column.

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
AFTER first_name;
```

Can use the keywords **FIRST** and **AFTER** your_column, but can also use **BEFORE** your_column, **LAST**, **SECOND**, and **THIRD**.

phone	contact_id	last_name	first_name	email

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

+ FIRST ;

contact_id	last_name	first_name	email	phone

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

+ LAST ;

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

+ FIFTH ;

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

+ ;

Can use the keywords **FIRST** and **AFTER** your_column, but can also use **BEFORE** your_column, **LAST**, **SECOND**, and **THIRD**.

contact_id	phone	last_name	first_name	email

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

+ SECOND ;

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

+ BEFORE last_name ;

contact_id	last_name	phone	first_name	email

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

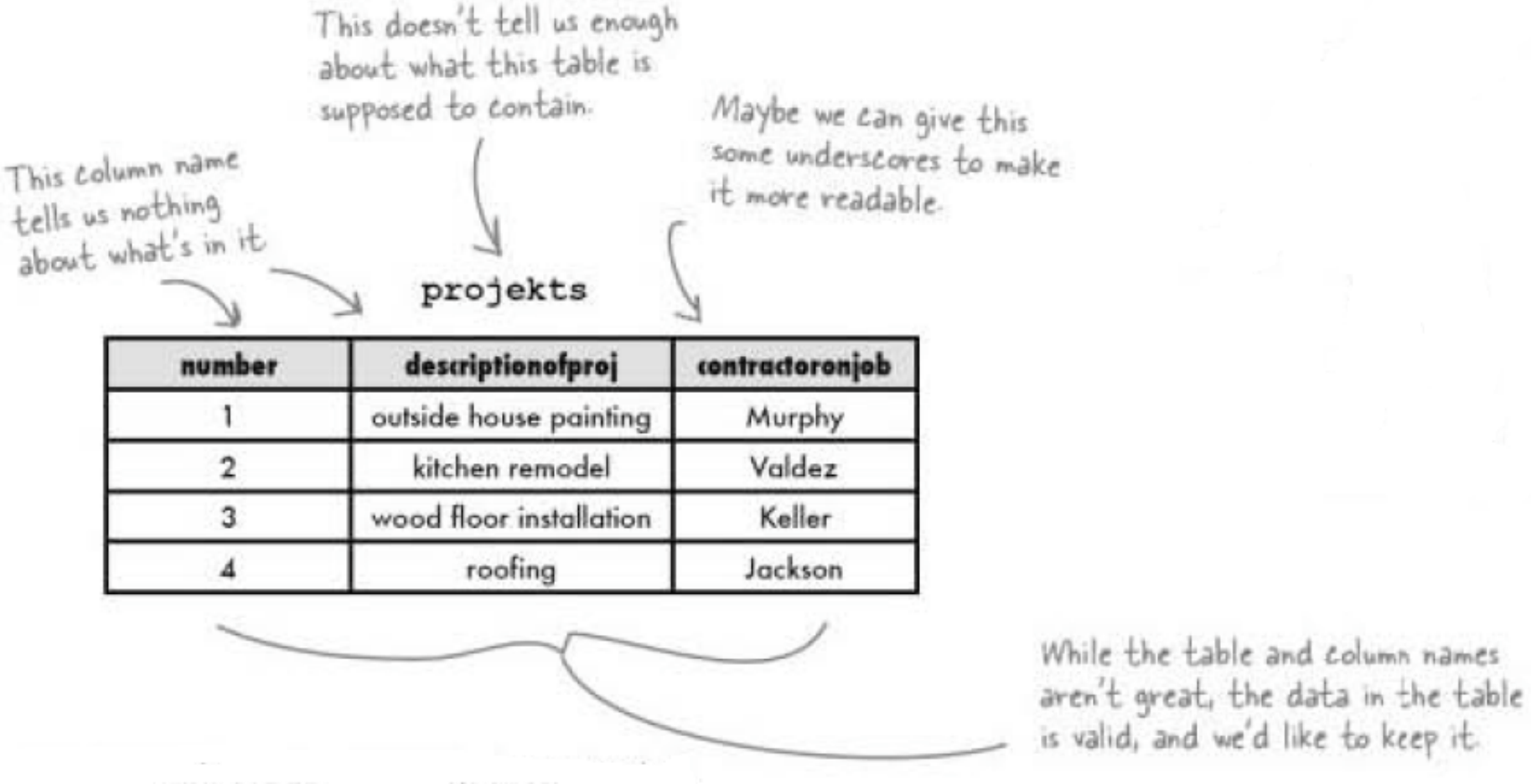
+ AFTER last_name ;

ALTER TABLE rules

- **CHANGE** both the name and data type of an existing column ※
- **MODIFY** the data type or position of an existing column ※
- **ADD** a column to your table – you pick the data type
- **DROP** a column from you table ※

※Possible loss of data may occur, no guarantees offered.

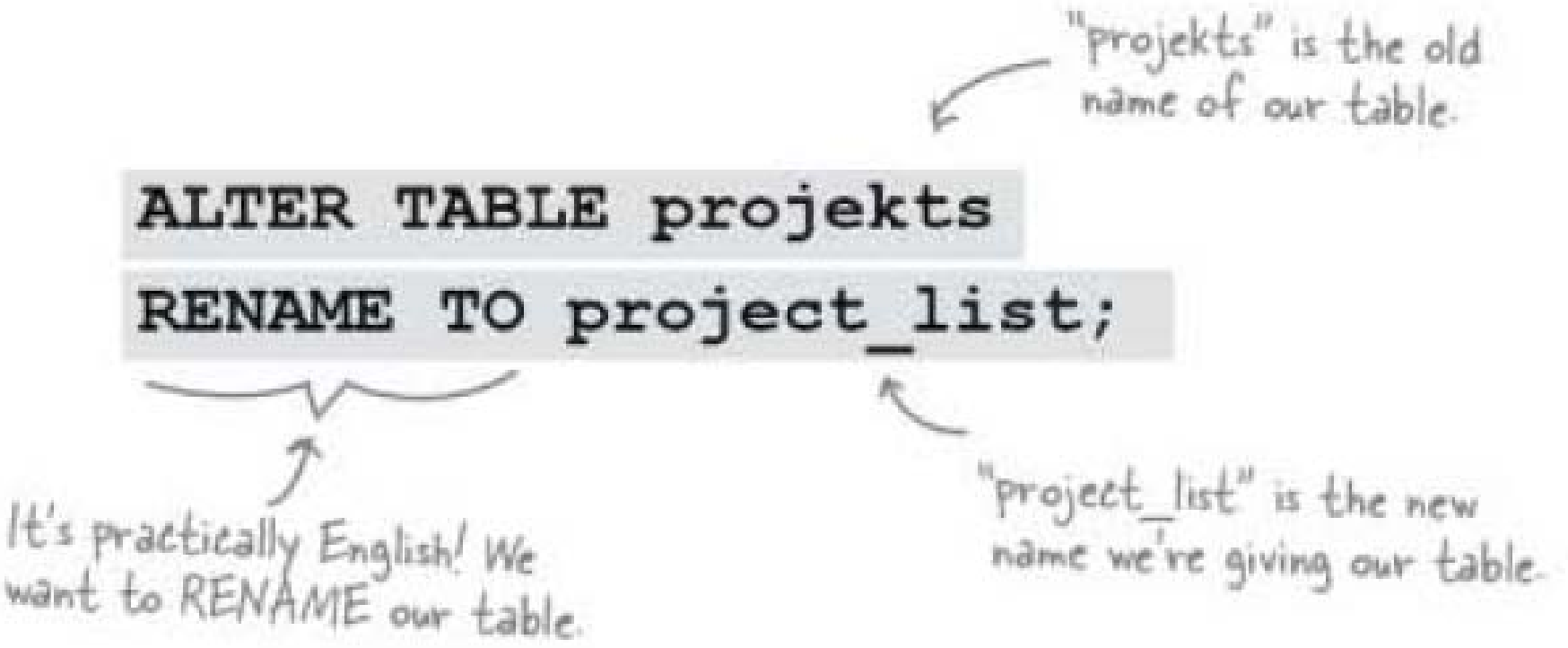
Extreme table makeover



Extreme table makeover (Cont.)

```
mysql> DESCRIBE projekts;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| number             | int(11)     | NO   |     | 0       |       |
| descriptionofproj  | varchar(50) | YES  |     | NULL    |       |
| contractoronjob    | varchar(10) | YES  |     | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Renaming the table




```
mysql> ALTER TABLE projekts
    -> RENAME TO project_list;
Query OK, 0 rows affected (0.00 sec)
```

To find the columns in this sentence that describes how we're going to use our table, then fill in the column names.

Requirements

1. we'll add a primary key with a unique project number in it.
2. We'll need columns to describe each improvement, its start date, estimated cost, and the name of the contracting company working on it, along with their phone number.

proj_id

To find the columns in this sentence that describes how we're going to use our table, then fill in the column names.

Requirements

1. we'll add a primary key with a unique project number in it.
2. We'll need columns to describe each improvement, its start date, estimated cost, and the name of the contracting company working on it, along with their phone number.

proj_id

start_date

proj_desc

con_phone

est_cost

con_name

Retooling our columns

This column will contain a description for each project We'll name it proj_desc

project_list

number	descriptionofproj	contractoronjob

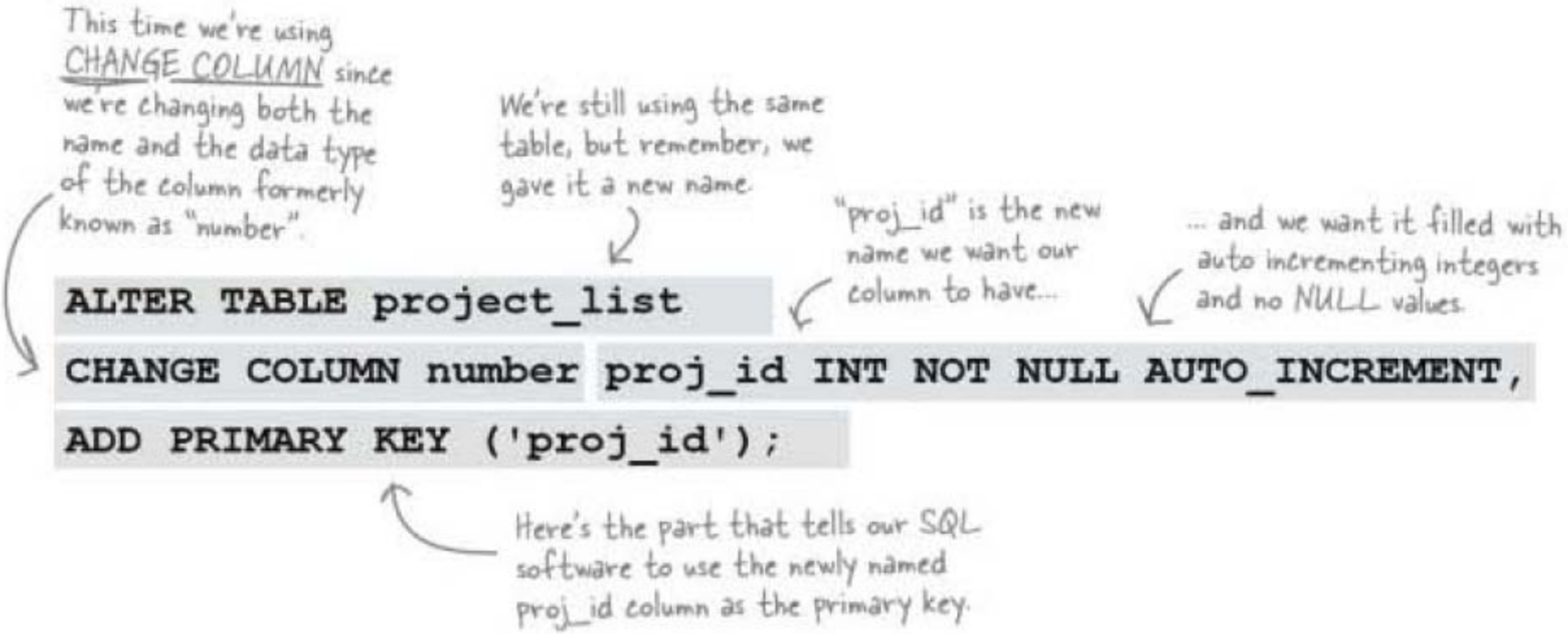
Since it's first in the table, this column will become our proj_id. It'll contain the primary key.

This will hold the name of our contracting company, con_name.

project_list

proj_id	proj_desc	con_name

ALTER and CHANGE




```
mysql> ALTER TABLE project_list
    -> CHANGE COLUMN number proj_id  INT  NOT NULL AUTO_INCREMENT,
    -> ADD PRIMARY KEY (proj_id);
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

Change two columns with one SQL statement


```
mysql> ALTER TABLE project_list
    -> CHANGE COLUMN descriptionofproj proj_desc  VARCHAR(100),
    -> CHANGE COLUMN contractoronjob  con_name   VARCHAR(30);
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

Watching out

If you change the data type to something new, you may lose data.

Changing data type, but name to stay the same

```
ALTER TABLE project_list
MODIFY COLUMN proj_desc VARCHAR(120);
```

The name of the column we're modifying.

The new data type.

And of course you've made sure that the new data type won't cause you to truncate your old data!

To write a single **ALTER TABLE** statement to add in three more columns : a phone number, a start date, and an estimated cost.

```
mysql> DESC project_list;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| proj_id    | int(11)      | NO   | PRI | NULL    | auto_increment |
| proj_desc  | varchar(100) | YES  |     | NULL    |                |
| con_name   | varchar(30)  | YES  |     | NULL    |                |
| con_phone  | varchar(10)  | YES  |     | NULL    |                |
| start_date | date         | YES  |     | NULL    |                |
| est_cost   | decimal(7,2) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

```
ALTER TABLE project_list
ADD COLUMN con_phone VARCHAR(10),
ADD COLUMN start_date  DATE,
ADD COLUMN est_cost DECIMAL(7,2);
```

Quick! DROP that column

- To write the SQL statement to drop the start_date column.

```
ALTER TABLE project_list
DROP COLUMN start_date;
```

Once you've dropped a column, everything that was stored in it is removed too!

To change from before to after.

Before

hooptie

color	year	make	mo	howmuch
silver	1998	Porsche	Boxter	17992.540
NULL	2000	Jaguar	XJ	15995
red	2002	Cadillac	Escalade	40215.9

After

car_table

car_id	VIN	make	model	color	year	price
1	RNKLK66N33G213481	Porsche	Boxter	silver	1998	17992.54
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	15995.00
3	3GYEK63NT2G280668	Cadillac	Escalade	red	2002	40215.90

To change from before to after.

```
ALTER TABLE hooptie
RENAME TO car_table,
ALTER TABLE car_table
ADD COLUMN car_id INT NOT NULL
                        AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (car_id),
ALTER TABLE car_table
ADD COLUMN VIN VARCHAR(16) SECOND,
CHANGE COLUMN mo model VARCHAR(20),
MODIFY COLUMN color AFTER model,
MODIFY COLUMN year SIXTH,
CHANGE COLUMN howmuch price DECIMAL(7,2);
```

Variable cases

- To add **NOT NULL**.

```
mysql> DESC pk_test;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
ALTER TABLE pk_test
CHANGE id  id  INT  NOT NULL;
```

```
mysql> DESC pk_test;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   |     | NULL    |       |
| name  | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Variable cases (Cont.)

- To add **AUTO_INCREMENT**.

```
mysql> DESC pk_test;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   |     | NULL    |       |
| name  | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
ALTER TABLE pk_test
CHANGE id  id  INT  NOT NULL AUTO_INCREMENT;
```

```
mysql> ALTER TABLE pk_test
    -> CHANGE id  id INT NOT NULL AUTO_INCREMENT;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto colum
n and it must be defined as a key
```

Variable cases (Cont.)

- To add **AUTO_INCREMENT**.

```
ALTER TABLE pk_test
CHANGE id  id  INT  NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (id);
```

```
mysql> DESC pk_test;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(10) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

Bullet Points

- Use **CHANGE** when you want to change both the name and the data type of a column.
- Us **MODIFY** when you wish to change only the data type.
- **DROP COLUMN** does just that it drops the named column from the table.
- Use **RENAME** to change the name of your table.
- Can change the order of your columns using **FIRST**, **LAST**, **BEFORE** column_name, **AFTER** column_name, **SECOND**, **THIRD**, **FOURTH**, etc.
- With some RDBMSs, can only change the order of columns in a table when you add them to a table.

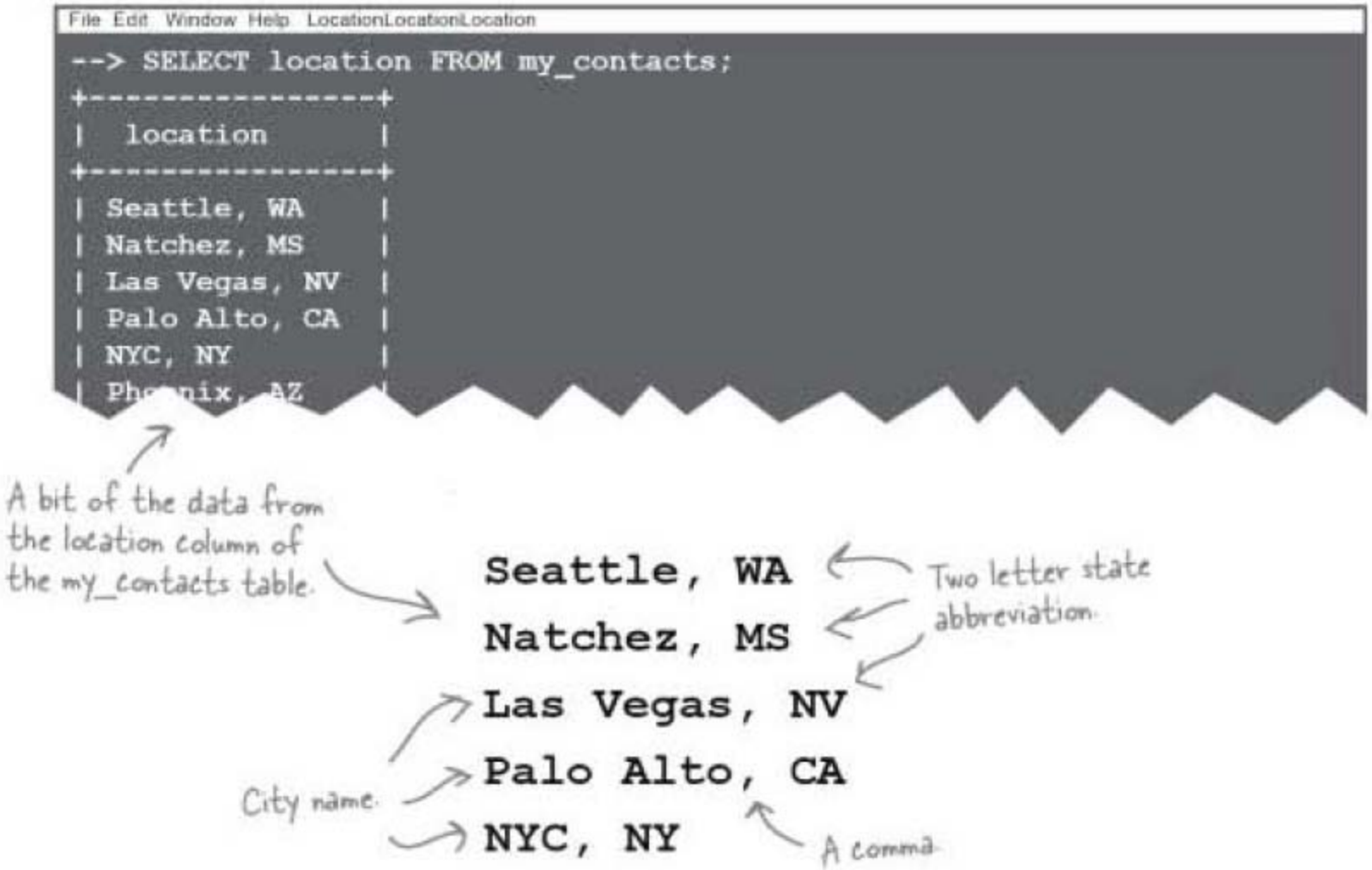
Being atomic columns

```
CREATE TABLE my_contacts
(
    contact_id INT NOT NULL AUTO_INCREMENT
    last_name VARCHAR(30) default NULL,
    first_name VARCHAR(20) default NULL,
    email VARCHAR(50) default NULL,
    gender CHAR(1) default NULL,
    birthday DATE default NULL,
    profession VARCHAR(50) default NULL,
    location VARCHAR(50) default NULL,
    status VARCHAR(20) default NULL,
    interests VARCHAR(100) default NULL,
    seeking VARCHAR(100) default NULL,
    PRIMARY KEY (contact_id)
)
```

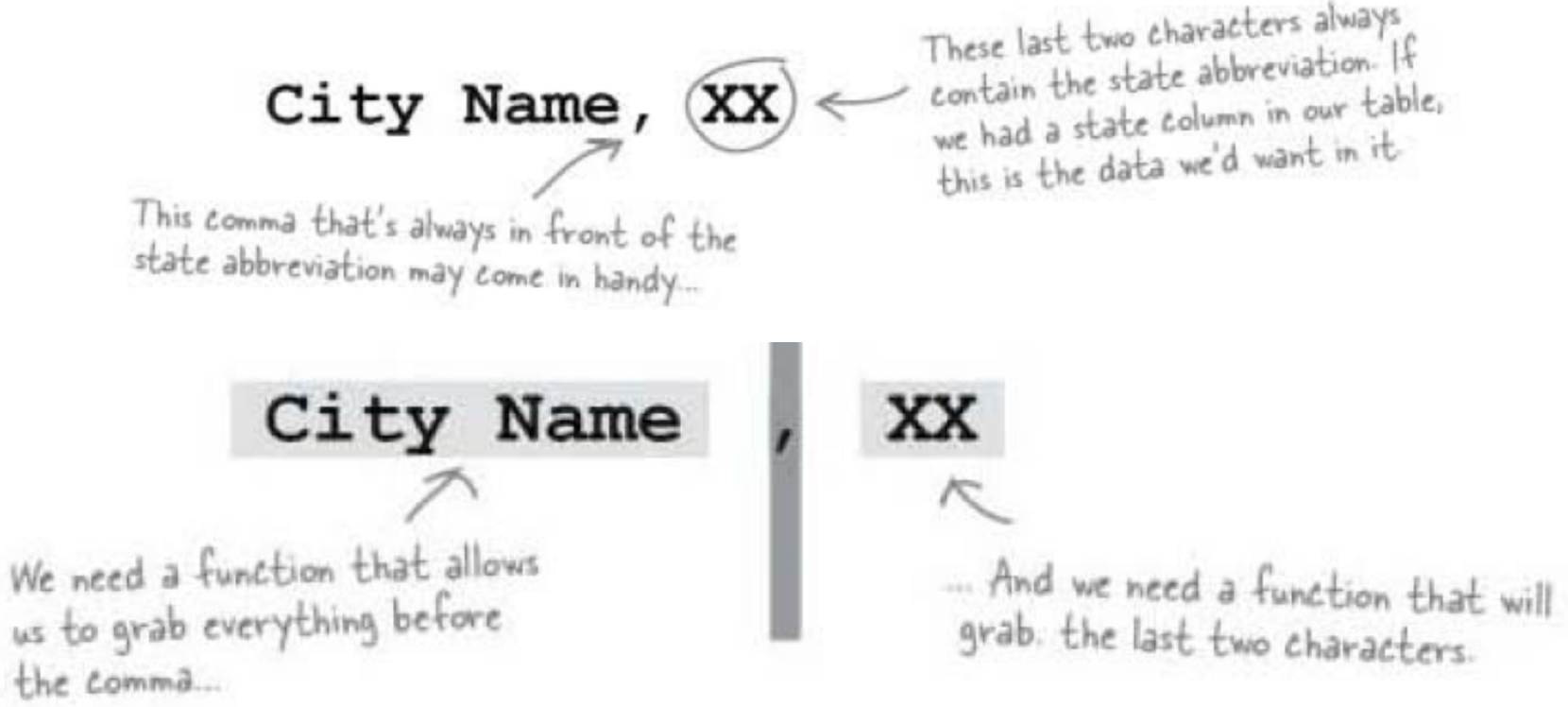
We added these two lines to create and designate our primary key.

These four columns aren't very atomic and could use some tweaking with ALTER TABLE.

Being atomic columns (Cont.)



Look for Pattern



To write an **ALTER TABLE** statement that adds city and state columns to my_contacts.

```
ALTER TABLE my_contacts
ADD COLUMN city VARCHAR(50),
ADD COLUMN state CHAR(2);
```

A few handy string functions

- To **SELECT** the last two characters

```
SELECT RIGHT(location, 2) FROM my_contacts;
```

Start at the RIGHT side of the column. (You can use LEFT in exactly the same way.)

This is the column to use.

This is how many characters to select from the RIGHT side of the column.

- To **SELECT** everything in front of the comma

```
SELECT SUBSTRING_INDEX(location, ',', 1) FROM my_contacts;
```

This grabs part of the column, or substring. It looks for the string in single quotes (in this case, it's a comma) and grabs everything in front of it.

Again, the column name.

Here's the comma the command is looking for.

This is the tricky part. It's "1" because it's looking for the first comma. If it were "2" it would keep going until it found a second comma and grab everything in front of that.

A few handy string functions (Cont.)

- RIGHT(str, len)

```
SELECT RIGHT ('foobarbar', 4);
```

- SUBSTRING_INDEX(str, delim, count)

```
SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
```

- SUBSTRING(str, pos), SUBSTRING(str FROM pos), SUBSTRING(str, pos , len), SUBSTRING(str FROM pos FOR len)

```
SELECT SUBSTRING('Quadratically', 5);
SELECT SUBSTRING('foobarbar'  FROM 4);
SELECT SUBSTRING('Quadratically', 5, 6);
SELECT SUBSTRING('Sakila', -3);
SELECT SUBSTRING('Sakila', -5, 3);
SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
```

A few handy string functions (Cont.)

- UPPER(str), LOWER(str)

```
SELECT UPPER('uSa');
```

- REVERSE(str)

```
SELECT REVERSE('spaGHEtti')
```

- LTRIM(str), RTRIM(str)

```
SELECT LTRIM('   dogfood   ');
```

- LENGTH(str)

```
SELECT LENGTH('San Antonio, TX  ');
```

What's my purpose?

SELECT

LEFT

SUBSTRING_INDEX()

RIGHT

ADD COLUMN

ALTER TABLE

DELETE

INSERT

UPDATE

Take a look at the data in a particular column to find a pattern.

Add new empty columns into our table.

Grab part of the data from a text column.

Put the data we grabbed in step 2 into one of the empty columns.

Use a current column to fill a new column

```
UPDATE my_contacts
SET state = RIGHT(location, 2);
```

Here's the new column for our state data.

And here's the string function that grabs the last two characters from the location column.

It will work without a **WHERE** clause.

How our UPDATE and SET combo works

my_contacts

contact_id	location	city	state
1	Chester, NJ		
2	Katy, TX		
3	San Mateo, CA		

Here's a simplified version of our table.

And here's our SQL statement

```
UPDATE my_contacts
SET state = RIGHT(location, 2);
```

How our UPDATE and SET combo works (Cont.)

First time through →

```
UPDATE my_contacts
SET state = RIGHT('Chester,NJ',2)
```

Takes the first record's location column and operates on it

Second time through →

```
UPDATE my_contacts
SET state = RIGHT('Katy, TX',2)
```

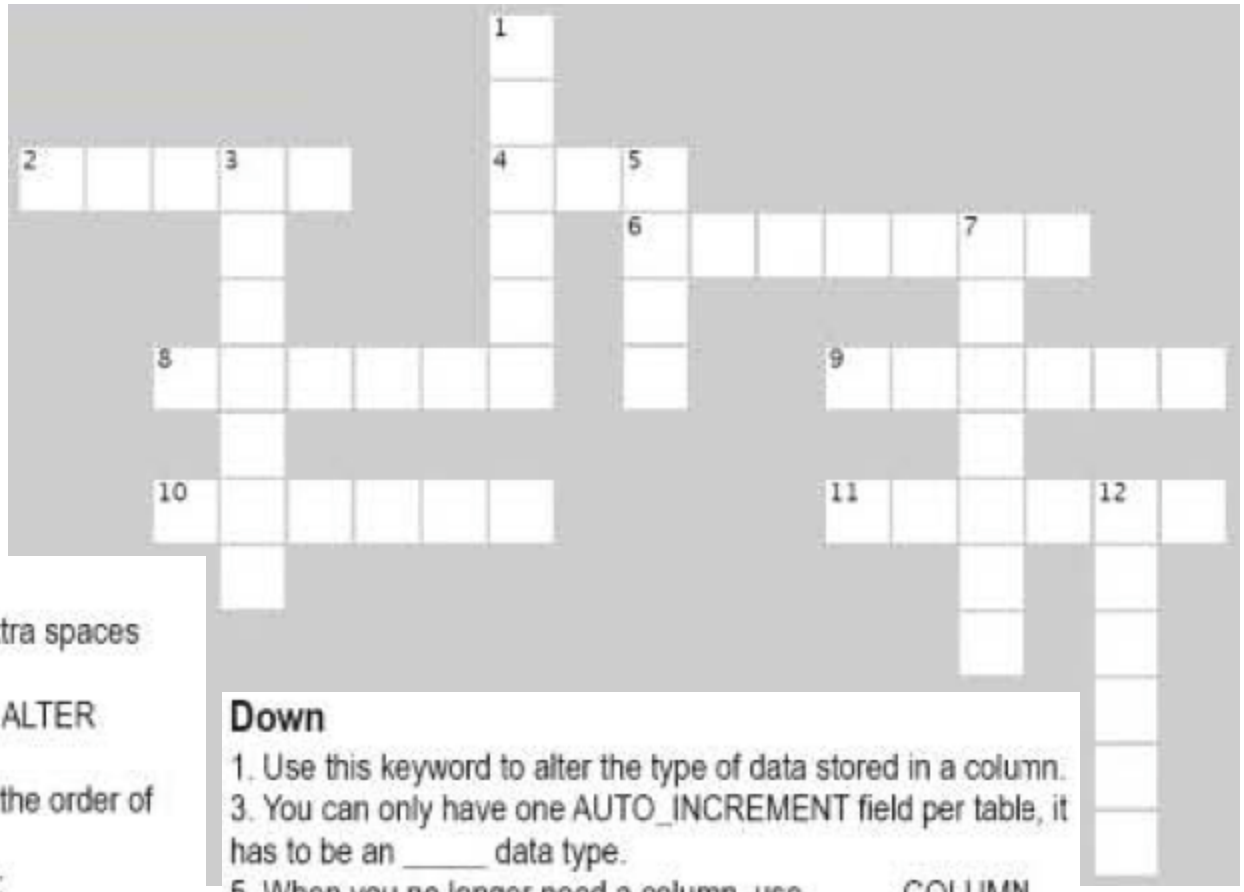
Now the second one

Third and final time through, because there are only three records →

```
UPDATE my_contacts
SET state = RIGHT('San Mateo, CA',2)
```

And finally the third one

Altercross



Across

2. _____(your_string) returns your string with extra spaces removed from before (to the left of) a string.
4. Our table can be given new columns with the ALTER statement and _____ COLUMN clause.
6. _____(your_string) does just that, it reverses the order of letters in your string.
8. ALTER TABLE projekts _____ TO project_list;
9. You can use _____ functions in combination with SELECT, UPDATE, and DELETE.
10. SUBSTRING(your_string, start_position, length) gives you part of your_string, starting at the letter in the start_position. _____ is how much of the string you get back.
11. Use _____ to change the name of your table.

Down

1. Use this keyword to alter the type of data stored in a column.
3. You can only have one AUTO_INCREMENT field per table, it has to be an _____ data type.
5. When you no longer need a column, use _____ COLUMN with ALTER.
7. Values stored in CHAR or VARCHAR columns are known as these.
12. Use this clause with ALTER when you only wish to change the data type.

Review

ALTER TABLE
Lets you change the name of your table and its entire structure while retaining the data inside of it.

ALTER with CHANGE
Lets you change the name and data type of an existing column.

ALTER with MODIFY
Lets you change just the data type of an existing column

ALTER with ADD
Lets you add a column to your table in the order you choose.

ALTER with DROP
Lets you drop a column from you table.

String functions
Lets you modify copies of the contents of string columns when they are returned from a query. The original values remain untouched.