Introduction to SQL

The Basics

Jon Flanders
@jonflanders

pluralsight
hardcore developer training

What is SQL?

- Structured Query Language
- A special-purpose programming language
  - Its purpose is to manipulate relational databases
  - Declarative language
  - Contains both a data definition syntax as well as a data manipulation syntax
- It’s both an ANSI and ISO standard
  - In this course, I’ll stick to the standards-based SQL–hopefully without the standards-based biased toward complex concepts

What is a Database?

- A container to help organize data in a constructive way
- Useful when you have some threshold amount of data
  - Think 500 Excel spreadsheets instead of 5
- Putting all the data in one place makes it easy to
  - Query the data
  - Update the data
  - Insert new data
  - Delete old data
- One source of truth rather than many sources of truth
- There are many different types of databases
  - Relational
  - Object-oriented
  - Document-based
- Some large number of databases in use today are classified as using the “relational model”
  - SQL is a language built for this model

What Does “Relational” Mean?

- The relational model is the database model on which SQL is based
  - It is a way to describe data and the relationships between data entities
- The pure Relational Model is math all the way down
  - Originally was based on relational algebra and tuple relational calculus
- SQL has either “watered-down” or “improved” the “pure” relational model over time
  - But we still use the term relational to describe the system in use today

Tables, Columns, and Rows

- In a relational database, data is stored in a construct known as a Table
  - A Table has a name and a collection of Columns
- Each Column also has a name
  - Each Column also has a restriction that restricts the size and category of data that can be stored in that Column
  - Also each Column can be required or not-required
- Each row contains data for at least all the required columns
- Rows can be retrieved by asking questions of the data
  - The questions will be related to the values in columns (e.g. “Who are all my contacts that have a phone number that starts with 310?”)
  - Asking such questions with SQL is known as querying

Keys

- Another essential piece of a relational model is keys
- Each Table should/must have one column that can uniquely identify a row
  - This column is known as the Primary Key
- If more than one table uses the same primary key, you can then merge those two tables together
  - More than likely you will want to merge a subset of each table
- A table can also have a column that is classified as a Foreign Key
  - A Foreign Key links that table to another Table's Primary Key
  - This also allows rows from each table to be linked together
- Sometimes keys are "natural"
  - Like an ISBN number for a book – unique across all published books on the planet
- Sometime you have to "invent" keys
  - Usually an integer is sufficient
  - Often a database will add this data in each row automatically for you – often called an "identity" column

Example: A Database for Contacts

- Why not just have one table?
- Imagine a simple Contacts database
  - Just first name, last name, and email address

First Name	Last Name	Email
Jon	Flanders	jon@...
Fritz	Onion	fritz@...

- What if Jon has more than one email address?
- Tempting to just add a column
  - Now we are just limited to two emails
  - Something isn’t right here
  - Limits the questions we can potentially ask this data in the future

First Name	Last Name	Email1	Email2
Jon	Flanders	jon@...	jon2@...
Fritz	Onion	fritz@...	null

A Better Solution

Key	First Name	Last Name
1	Jon	Flanders
2	Fritz	Onion

Key	Person Key	Email
1	2	fritz@...
2	1	jon@...
3	1	jon2@...

- Now we can ask better questions
- “What are all of Jon’s email addresses?”
- “How many email addresses does each person have?”

- This is a design process in database design known as Normalization
- Much more to normalization
- Concentrate on what questions can be asked – you’ll do pretty well

SQL Statement

- A valid SQL Statement is made up of an actionable set of valid words
  - Some of the words are defined by SQL, some are defined by you
  - SQL is english-based so it is “readable”
  - Most people think of “query” when they think of SQL but it can do much more than just query
  - Most parts of a SQL Statement can be broken down into “clauses”
- A valid SQL Statement has a semi-colon (;) at the end of the statement
  - You can put multiple statements together, separated by semi-colons
- SQL is not case-sensitive
  - Most people write SQL Statements with SQL-specific words in all caps, and user-defined words in lower case
- Comments
  - Comments are good for documentation of SQL Statements saved for re-use
  - -- is for single-line comments
  - /* is for multi-line comments*/

Commands

- SQL Statements all start with a command
  - Generally the command is a verb
- SELECT is an example

```
SELECT  VALUES
FROM TABLENAME   ;
```

Commands

- SQL Statements all start with a command
  - Generally the command is a verb
- SELECT is an example

```
SELECT  MyColumnName,‘Constant’
FROM MyTableName ;
```

After the Command

- What comes next is highly dependent on the command itself
- For example – after the SELECT command comes
  - A definition of the set you want returned – a list of columns, and/or a wildcard (*) to indicate all columns, and/or a set of literal values. This is not optional
  - A FROM clause which contains one or more table names. This is actually optional (if the set contains only literal values)
- There is much more to using SELECT that will be covered later in this course – stay tuned

Naming Things

- Naming things is also important for being able to use a database successfully
  - Different people have different ideas on the right way to name things in a database
- I will use a few simple rules in this course
  - Table names will be singular: a Table name describes what a row of data in that table “is” (e.g. user, email, phone)
  - Column names will never be repeated inside of a particular database – this is to keep things clear when looking at names of columns
- Names in SQL are “scoped”
  - A Database will have a name
  - Tables inside of a database will have a name, but a Table name should be referred to using its “full” name, which includes the Database name
  - A period is used to separate each part of the name
  - Database.Table
  - Columns are also scoped: Table.Column

Creating Things with SQL

- There is a whole set of SQL commands that relate to creating and modifying constructs in a database

```
--this is not ANSI SQL
--but is supported by most vendors
CREATE DATABASE Contacts;
USE DATABASE Contact;
```

```
CREATE TABLE Contacts.User (...);
```

Data Types (not exhaustive)

- Each Column has a restriction on the type of data that can be stored inside of it

Data Type	Value Space
CHARACTER	Can hold N character values – set to N statically
CHARACTER VARYING	Can hold N character values – set to N dynamically – storage can be less than N
BINARY	Hexadecimal data
SMALLINT	-2^15 (-32,768) to 2^15-1 (32,767)
INTEGER	-2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)
BIGINT	-2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)
BOOLEAN	TRUE or FALSE
DATE	YEAR, MONTH and DAY in the format YYYY-MM-DD
TIME	HOUR, MINUTE and SECOND in the format HH:MM:SS[.sF] where F is the fractional part of the SECOND value
TIMESTAMP	Both DATE and TIME

RDMS

- Relational Database Management System
  - More than just the Database
  - Also includes tools and applications to help manage larger-scale database installations
- Most extend the ANSI SQL language with vendor-specific extensions
- Most are proprietary
  - Oracle has PL/SQL
  - SQL Server has T-SQL
- Most ANSI SQL will work with any RDMS
  - Notice the key term here “most”

SQL in This Course

- In this course I am using ANSI SQL
  - When you use your RDMS, some details could be different
  - I'll try to point as many of those differences out that I can
- Why I am using MySQL?
  - Free and open-source
  - Runs on all major platforms: Windows, OSX, Linux
  - Has an ANSI operation mode that enforces ANSI compliance (most RDMS's do not have this feature)

Summary

- SQL is a powerful declarative language that you can grasp by understanding a few basic concepts

Introduction to SQL

Using SELECT

Jon Flanders
@jonflanders

pluralsight
hardcore developer training

What you will learn...

- How to effectively use the SELECT command to ask interesting questions of your data

Questions

- **We store data in databases for a reason – because that data might be important to us later**
  - Important to our business or important to us personally
- **Querying data is all about asking questions of that data**
  - Pretend that the data is actually another human that knows everything that is stored in the data
  - All queries could be postulated in English
  - Understanding the mapping between English and SQL can help you learn SQL much faster

Anatomy of a Query

- From the first module, you should already know that all queries start with the SELECT command
- After SELECT there is only one required thing – the "select list"
- The select list is a list of items you want in the set
- The select list determines the shape of the result set
- If the select list only contains a set of variables, that can be the end of the query
  - Variables are pieces of data that aren't stored in a table – instead they are declared locally in the SQL Statement itself

```
SELECT 'Hello','World';
```

The Select List

- **More often, the select list contains a list of columns from a table you want to query**
  - If you are querying columns from a table, you will need a FROM clause after the select list
  - After every column comes a comma, except the last column in the list

```
SELECT <COLUMN_NAME>, <COLUMN_NAME>
    FROM <TABLE_NAME>;
```

```
SELECT person_first_name, person_last_name
    FROM person;
```

Just Give Me all the Columns!

- **If you want all the columns from a table, you can use the wildcard character (*)**
  - This is useful, but inefficient and dangerous
  - Inefficient, because now the database has to look up all the columns for you
  - Dangerous, because the order of columns might change later and break code that depended on the order of the columns
  - People will yell at you if you give them SQL with * :-)

FROM

- **When selecting from a single table – FROM is pretty easy**
  - FROM <TABLENAME>
  - More on multiple tables later in this course
- **You *should* qualify all the columns in the select list with the Table's name**
  - <TABLENAME.COLUMNNAME>
  - This is to avoid a column name collision later if you subsequently add other tables to the query (which might have a column with the same name)
- **You can also alias your Table's name in the FROM clause**
  - Make the alias shorter than the table name and now qualifying the column names with the Table's name doesn't seem so bad

```
What are all the first names of the
    people in my contact list?
```

```
SELECT p.person_first_name
  FROM person p;
```

WHERE

- **The next clause you can add to a query is the "WHERE" clause**
- **The WHERE clause is like a search**
  - Much like searching on the web, but with more precision
- **The body of the WHERE clause will be a set of expressions that can evaluate to TRUE or FALSE**
  - We call such an expression a "boolean expression" – because a boolean value is either TRUE or FALSE
- **If the expression evaluates to TRUE for a particular row in the potential result set, it will be returned in the actual result set**

```
Who are all the people in my
contact list that have the first
name Jon?
```

```
SELECT p.person_first_name
    FROM person p
WHERE p.person_first_name = 'Jon';
```

Simple Operators

- **The key with WHERE clause expressions is to know how all the operators in SQL work**
- **Unlike in Math, these operators can work on non-numeric data types**
  - Strings, dates, etc.

Operator	Description
=	Equal to – returns true if the values on either side are equal to each other
<>	Not equal to – returns true if the values on either side are *not* equal
>	Greater than – returns true if the value on the left is larger than the value on the right
<	Less than – returns true if the value on the left is smaller than the value on the right
>=	Greater than or equal – returns true if the value on the left is greater than or equal to the value on the right
<=	Less than or equal – returns true if the value on the left is less than or equal to the value on the right

AND and OR

- **Each boolean expression can be combined with other boolean expressions**
  - This makes it possible to ask more complex questions of the data
- **AND means that the two expressions combined by the AND must *both* evaluate to TRUE for a row to make it into the result set**

Who are all the people in my contact list that have the first name Jon and have a birthday later than 1965?

```
SELECT p.person_first_name
    FROM person p
WHERE p.person_first_name = ‘Jon’ AND
p.birthday > ’12/31/1965’;
```

OR

- OR means that if either of the two combined expressions evaluates to TRUE, the row becomes part of the result set

Who are all the people in my contact list that have the first name Jon or a last name of Flanders?

```
SELECT p.person_first_name
    FROM person p
WHERE p.person_first_name = ‘Jon’ OR
p.last_name = ‘Flanders’;
```

BETWEEN

- **BETWEEN is an operator that acts on a column and two values**
- **If a row's column value is "between" these two values, the expression evaluates to TRUE**
  - BETWEEN is inclusive, it includes the two values in the calculation of what is "between"

Who are all the people in my contact list that I have contacted at least once but no more than 20 times?

```
SELECT p.person_first_name
       FROM person p
WHERE p.contacted_number BETWEEN 1 AND 20;
```

LIKE

- **LIKE is a special operator just for strings**
  - You give LIKE a pattern and it will match strings that match that pattern
  - % is the wild-card character
  - The % can go anywhere in the string

Who are all the people in my contact list that have a first name that begins with the letter J?

```
SELECT p.person_first_name
    FROM person p
WHERE p.person_first_name LIKE ‘J%’;
```

IN

- **An expression with IN requires a column and a list of potential values**
  - Values can be numeric or string or date
  - It might seem to overlap with BETWEEN, but it will come in handy later
- **If a row's column value matches *any* of the potential values in the list, then the row is added to the result set**

Who are all the people in my contact list that are named Jon, Shannon, or Fritz?

```
SELECT p.person_first_name
    FROM person p
WHERE p.person_first_name IN
('Jon','Shannon','Fritz');
```

IS

- **IS is a special operator that helps address when a value in a column might be NULL**
  - NULL is special in SQL and it doesn't work with the equality (=) operator

Who are all the people in my contact list that don't have a last name?

```
SELECT p.person_first_name
       FROM person p
WHERE p.last_name IS NULL;
```

IS NOT

- **IS NOT is the complement to IS**
  - It is true if the column value is *not* NULL

Who are all the people in my contact list that have a last name?

```
SELECT p.person_first_name
    FROM person p
WHERE p.last_name IS NOT NULL;
```

Summary

- The SELECT command in SQL has more power than it seems
- Using FROM expressions combined with AND or OR enables you to write very powerful queries

Introduction to SQL

Shaping result sets, and using joins

Jon Flanders
@jonflanders

pluralsight
hardcore developer training

What you will learn...

- How to shape a single table result set, and then how to join two tables together for a combined result set

Shaping a Result

- **In the last module, I showed you how to get a result from a single Table using SELECT**
- **Using the WHERE clause, I showed you how to restrict the results from that Table by using expressions**
  - Expressions that evaluate to true on a particular row cause that row to be part of the result set
- **In addition to restricting the result, another common SQL task is to sort the result**
- **You also might want to transform the data so it can be more easily consumed**
  - Consumed by the end-user or an application
- **SQL provides native capabilities that should be used before resorting to other means**

ORDER BY

- **The ORDER BY clause allows you to sort the result set**
  - ORDER BY comes after the WHERE clause, but the WHERE clause isn't required
- **You specify one or more of the columns from the result set**
  - Multiple columns are separated by commas
- **Ascending (ASC) order is the default**
  - You can add the keyword DESC to cause the ordering to be descending

Who are all the people in my contact list, ordered by last name?

```
SELECT p.person_first_name,p.persons_last_name
        FROM person p
        ORDER BY p.person_last_name
```

Set Functions

- **SQL includes a number of built-in functions to provide additional functionality**
- **Some of those functions help to turn column data from Tables into computed values**
  - You pass the column name as the parameter to the function
- **You use these functions instead of columns in the select list**
  - This rule has an exception – stay tuned
- **These functions help add additional interesting questions we can ask of the data**

```
How many contacts do I have?
```

```
Who is the contact that I've
interacted with the least?
```

```
What is the average number of times
I've contacted people in my contact
list?
```

Computational Functions

Operator	Description
COUNT	Returns the count of the column specified – note that COUNT **includes** NULL values when used with *
MAX	Returns the maximum value of the column specified – does not include NULL values
MIN	Returns the minimum value of the column specified – does not include NULL values
AVG	Returns the average value for the column specified – does not include NULL values and only works on numeric columns
SUM	Returns the sum of the values for the column specified – does not include NULL values and only works on numeric columns

Result Set Qualifiers - Review

- **By default, a query will return ALL rows from the Table listed after the FROM clause**
- **There are two ways to limit the number of rows in a simple SELECT Statement**
- **One is to add additional clauses after the FROM clause**
- **Another is to use the DISTINCT qualifier before the select list**
  - This limits the result set to all the distinct values rather than all the values
  - ALL is also a result set qualifier – but it is always implied

What are all the unique first names among my contacts?

```
SELECT DISTINCT p.person_first_name
  FROM person p;
```

Qualifiers and Set Functions

- **You can add the DISTINCT or ALL qualifier to a column passed to a Set Function**
  - ALL is the implicit default here, as it is when it is at the beginning of the select list
- **The Set Function is run against the DISTINCT result set of that column rather than all values**

What is the count of unique first names among my contacts?

```
SELECT Count(DISTINCT p.person_first_name)
  FROM person p;
```

GROUP BY

- **Normally, a SET Function has to be the only item in the select list**
  - Including other columns wouldn't make sense anyway
- **If you could run the function against a subset of the set, you could get a "sub" result**

- **GROUP BY lets you put all the subsets together, and link those subsets up to the column(s) specified in the GROUP BY clause**
  - So to appear in the GROUP BY clause, the column must be in the select list

```
What is the count of every
unique first names among my
contacts?
```

```
SELECT Count(DISTINCT p.person_first_name),
       p.person_first_name
  FROM person p
GROUP BY p.person_first_name;
```

HAVING

- The HAVING clause works against the result set returned with a GROUP BY clause in the same way that the WHERE clause works against the result set returned from a simple SELECT

What is the count of unique first name among my contacts that appears at least 5 times?

```
SELECT
Count(DISTINCT p.person_first_name) as
NameCount,
p.person_first_name
FROM person p
GROUP BY p.person_first_name
HAVING NameCount >= 5;
```

JOINS

- **A join in SQL is creating a new result set from two or more tables**
- **Joins are made by extending the FROM clause to include all of the tables you wish to query**
- **The vast majority of the time, you will also extend the WHERE clause to include an equality expression that includes columns from each of the joined tables**
  - Most often these columns will be primary or foreign keys
  - Using the relations between the Tables to answer more complex questions

CROSS

- **The simplest type of JOIN is perhaps the least useful**
  - Also incredibly inefficient
- **The CROSS JOIN is what Maths calls a Cartesian Product**
  - All the rows and columns from both tables
- **It isn't very useful because there isn't any connection between the rows**
- **CROSS keyword is optional**

What are all the first names
and email addresses I have?

```
SELECT
p.person_first_name, e.email_address
  FROM person p, email_address e;
```

INNER JOIN

- **INNER JOIN is the expected "typical" join in a relational system**
- **Take all the rows from Table A**
  - Find all the rows in Table B where a key in Table A is equal to a key in Table B
  - Make a new result set with all those rows
- **INNER JOIN goes between table names in the FROM clause**
- **The ON clause follows the INNER JOIN clause**
  - Boolean expression to match the key columns

What are my contacts' email addresses?

```
SELECT
p.person_first_name, p.person_last_name,
e.email_address
FROM person p INNER JOIN email_address e
ON p.person_id = e.email_address_person_id;
```

OUTER JOIN

- **The difference between the INNER JOIN and the OUTER JOIN relates to NULL**
- **The INNER JOIN only joins against rows where there is a match in the joined table**
- **OUTER JOIN works even when a row in one of the tables doesn't match with a row in the joined table**
  - i.e., even when there is no row that matches the WHERE clause expression
- **A FULL OUTER JOIN will return a result set with all the joined rows**
  - The rows from the first table will also be returned, matched with NULL values for the second table's columns
  - If there are any rows in the second table that match the expression, they are also returned – and the cells from the first table will be NULL

OUTER JOIN

What are all my contacts and their email addresses, including the ones missing an email address and the ones with an email address but missing a contact name.?

```
SELECT
p.person_first_name, p.person_last_name,
e.email_address
FROM person p FULL OUTER JOIN email_address e
ON p.person_id = e.email_address_person_id;
```

person_first_name	person_last_name	email_address
Jon	Flanders	jon@...
Fritz	Onion	fritz@...
Shannon	Ahern	NULL
NULL	NULL	aaron@...

LEFT OUTER JOIN

- **Using LEFT OUTER JOIN means that only the rows from the table on the left of the LEFT OUTER JOIN clause will be returned**
  - Rows that are not matched will have NULL for the columns from the right-hand side table

LEFT OUTER JOIN

What are my contacts and their email addresses, including those I don't have an email for?

```
SELECT
  p.person_first_name, p.person_last_name,
  e.email_address
FROM person p LEFT OUTER JOIN email_address e
ON p.person_id = e.email_address_person_id;
```

person_first_name	person_last_name	email_address
Jon	Flanders	jon@...
Fritz	Onion	fritz@...
Shannon	Ahern	NULL

RIGHT OUTER JOIN

- RIGHT OUTER JOIN returns all the rows from the table on the right-hand side of the JOIN clause
- NULL values for rows that don't have a match in the left-hand side table

RIGHT OUTER JOIN

What are the email addresses I have, including those emails I don't have a person for?

```
SELECT
  p.person_first_name, p.person_last_name,
    e.email_address
FROM person p RIGHT OUTER JOIN email_address e
  ON p.person_id = e.email_address_person_id;
```

person_first_name	person_last_name	email_address
Jon	Flanders	jon@...
Fritz	Onion	fritz@...
NULL	NULL	aaron@...

SELF JOIN

- It is odd but sometimes it is useful to join a table against itself
- There isn't any specific syntax for this JOIN, but it is worth mentioning that the same table can be on both the left-hand side and the right-hand side of a JOIN clause

SUMMARY

- ORDER BY and GROUP BY can help you to shape your results to more easily answer complex questions from your data.
- JOINS make the relational model come to life by associating tables together

Introduction to SQL

Insert, Update, and Delete

Jon Flanders
@jonflanders

pluralsight
hardcore developer training

What you will learn...

- How to add, change, and delete data inside of Tables

CRUD

- **Create**
  - Add new rows to a Table
- **Read**
  - We've already done this!
- **Update**
  - Change a value in one or more of the columns in one or more rows in a Table
- **Delete**
  - Remove one or more rows from a Table
- **Should really be ISUD**
  - INSERT, SELECT, UPDATE, and DELETE are the actual commands
  - I guess CRUD sounds better

INSERT

- **The INSERT command enables you to put new data into a Table**
- **Really should be known as "INSERT INTO"**
  - INTO is a required part of the command
- **Table name follows INSERT INTO**
  - Only one Table at a time is allowed
- **The list of columns follows in a set of parens**
  - Separated by commas
  - All required columns must be present
- **After the columns comes the VALUES keyword**
  - Followed by a parens that hold onto the list of values to be put into each column
  - Each column in the list must have a matching value in the value list

INSERT INTO

```
INSERT INTO person
  (person_id, person_first_name, person_last_name,
person_contacted_number,person_date_last_contacted,
                person_date_added )
  VALUES (1, 'Jon', 'Flanders', 5,'2013-09-14
      11:43:31','2013-01-14 11:43:31');
```

Inserting Multiple Rows

- **You can insert multiple rows using multiple value lists**
  - Separate each value list with a comma
- **You can also use a query to select multiple rows "into" another Table**

```
INSERT INTO person
(person_id, person_first_name, person_last_name,
person_contacted_number,person_date_last_contact
ed,person_date_added )
VALUES (1, 'Jon', 'Flanders', 5,'2013-09-14
11:43:31','2013-01-14 11:43:31'),
(2, 'Shannon', 'Ahern', 0,'2013-08-14
11:43:31','2013-02-14 11:43:31');
```

```
INSERT INTO person p
SELECT * from old_person op
WHERE op.person_id > 300;
```

UPDATE

- UPDATE enables you to change one or more columns in a Table
- Without a WHERE clause, UPDATE (like DELETE) will affect all the rows in the Table
- SET keyword follows Table name (and alias)

```
UPDATE email_address e
   SET e.email_address_person_id = 5
 WHERE e.email_address = 'aaron@mail.com';
```

DELETE

- **DELETE deletes one or more rows**
  - Permanently!
  - Be careful!
- **Command is actually "DELETE FROM"**
- **Table name comes after the FROM**
- **Only one Table at a time allowed**
- **With no conditions – DELETE FROM deletes all rows!**

```
DELETE FROM person;
```

Transactions

- **A Transaction is a specific computer science concept**
  - Not the same as a ATM or bank transaction
- **A Transaction is a construct that creates a "context" around one or more SQL statements**

ACID

- **Another common SQL-related acronym**
- **It describes the qualities of a Transaction**
- **Atomic**
  - The Transaction either happens or it doesn't
- **Consistent**
  - The Transaction leaves the database in a consistent state
- **Isolated**
  - The Transaction happens in a isolated (serial) way
- **Durable**
  - The Transaction will be stored permanently, even in the face of a disaster (e.g. power outage)

Transaction Syntax

```
START TRANSACTION;
DELETE FROM
person;
COMMIT;
-- or ROLLBACK;
```

DELETE + WHERE

- **You can limit the number of rows that are deleted**
  - Use the same technique as limiting the number of rows returned from a SELECT statement
- **The WHERE clause comes after the Table name**

```
DELETE FROM person p
WHERE p.person_first_name LIKE ‘J%’;
```

DELETE + WHERE + IN

- **The IN clause can also be used after the WHERE clause to list a set of rows to be deleted**
  - Often useful for deleting a large number of rows based on the primary key

```
DELETE FROM person p
WHERE p.person_id IN (1,2,3,4,5,6);
```

Summary

- INSERT, UPDATE, and DELETE are the three SQL commands you want to learn if you are going to be modifying data in your database tables
- Remember to learn transactions in full to be able to protect your SQL statements against incorrect behavior

Introduction to SQL

Creating Tables

Jon Flanders
@jonflanders

pluralsight
hardcore developer training

What you will learn...

- How to create and modify Tables

Creating Things with SQL

- **There is a whole set of SQL commands that relate to creating and modifying constructs in a database**
- **These are classified as DDL**
  - Data Definition Language
  - Formally, it is a subset of SQL
- **Many RDMS products have visual tools to help you create Tables and relations**
  - So you may never need to do DDL by hand, understanding it is a good foundation for using those tools

Creating the Database

```
--this is not ANSI SQL
--but is supported by most vendors
CREATE DATABASE Contacts;
USE DATABASE Contact;
```

CREATE TABLE

- **CREATE TABLE is part of the ANSI standard**
- **CREATE TABLE takes a Table name as input**
- **A list of column definitions follows the Table name**
  - Inside of parens ( )
  - Each column definition is separated by a comma
- **Each column definition contains the following**
  - A column name
  - A column type definition

Column Definition

- **Starts with a name**
- **Followed by a data type**

Data Type	Value Space
CHARACTER	Can hold N character values – set to N statically
CHARACTER VARYING	Can hold N character values – set to N dynamically – storage can be less than N
BINARY	Hexadecimal data
SMALLINT	-2^15 (-32,768) to 2^15-1 (32,767)
INTEGER	-2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)
BIGINT	-2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)
BOOLEAN	TRUE or FALSE
DATE	YEAR, MONTH, and DAY in the format YYYY-MM-DD
TIME	HOUR, MINUTE, and SECOND in the format HH:MM:SS[.sF] where F is the fractional part of the SECOND value
TIMESTAMP	Both DATE and TIME

CREATE TABLE Example

```
CREATE TABLE email_address
( email_address_id INTEGER,
email_address_person_id INTEGER,
   email_address VARCHAR(55));
```

NULL or NOT NULL

- **After the column type, you can specify NULL or NOT NULL**
- **NULL is the default**
  - In fact, ANSI SQL disallows the default to be specified (most DBs allow it however)
  - Means that NULL values are valid values for that column in a particular row
- **NOT NULL means that a valid value is required for that column**
  - An attempt to insert a NULL value will fail
  - An attempt to use INSERT without specifying a value for that column will fail

```
CREATE TABLE email_address
( email_address_id INTEGER NOT NULL,
  email_address_person_id INTEGER,
email_address VARCHAR(55) NOT NULL);
```

PRIMARY KEY

- **After the data type you can add the PRIMARY KEY constraint instead of NULL or NOT NULL**
  - PRIMARY KEY columns are implicitly NOT NULL
- **It is possible for more than one column to form together as a compound PRIMARY KEY**

```
CREATE TABLE email_address
( email_address_id INTEGER PRIMARY KEY,
  email_address_person_id INTEGER,
  email_address VARCHAR(55) NOT NULL);
```

CONSTRAINT

- **You can add a CONSTRAINT clause at the end of your column definitions**
  - You can add the PRIMARY KEY constraint this way if you want to
  - You can add FOREIGN KEY constraints

```
CREATE TABLE phone_number
( phone_number_id INTEGER NOT NULL,
phone_number_person_id INTEGER NOT  NULL,
    phone_number VARCHAR(55) NOT NULL,
CONSTRAINT PK_phone_number PRIMARY KEY
(phone_number_id));
```

ALTER TABLE

- **ALTER TABLE enables you to add or change a column definition or CONSTRAINT on an existing Table**
  - Often used in database creation scripts to enable correct order of operations
- **Allows you to do anything that you could do in CREATE TABLE definition**
- **Warning – doesn't work if Table already has data in it that conflicts with the change**
  - For example, you can't change a column to NOT NULL if it already has NULL data in it
  - Need to use some sort of temporary Table to hold the original data, change it, ALTER the Table, and then put the data back

```
ALTER TABLE email_address
ADD CONSTRAINT FK_email_address_person FOREIGN
KEY(email_address_person_id)
REFERENCES person (person_id);
```

DROP TABLE

- **DROP TABLE command removes the Table from the database**
  - Also removes all the data
- **Use it wisely**
- **You often have to deal with relations**
  - Can't delete Table that has a column referenced by another Table as a foreign key

```
DROP TABLE person;
```

Summary

- Understanding DDL is a good foundation for working with SQL even if you use it rarely
- CREATE TABLE is the command to configure your columns and relations
- ALTER TABLE lets you change existing definitions
- DROP TABLE removes the table and all its rows from the database