Introduction to SQL

Using SELECT

Jon Flanders
@jonflanders

pluralsight
hardcore developer training

What you will learn...

What you will learn...

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

Questions

Questions

- We store data in databases for a reason – because that data might be important to us later

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

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**

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

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

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

Example Questions

Example Questions

```
Who are all my contacts?
```

Example Questions

Who are all my contacts?

How many contacts do I have?

Example Questions

Who are all my contacts?

How many contacts do I have?

Which of my contacts is the oldest?

Anatomy of a Query

Anatomy of a Query

- From the first module, you should already know that all queries start with the SELECT command

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"

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

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

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

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

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

The Select List

- More often, the select list contains a list of columns from a table you want to query

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

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

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>;
```

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!

Just Give Me all the Columns!

- If you want all the columns from a table, you can use the wildcard character (*)

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

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

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

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

FROM

- When selecting from a single table – FROM is pretty easy

FROM

- **When selecting from a single table – FROM is pretty easy**
  - FROM <TABLENAME>

FROM

- **When selecting from a single table – FROM is pretty easy**
  - FROM <TABLENAME>
  - More on multiple tables later in this course

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**

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>

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)

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**

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

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?
```

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;
```

Limiting the Result Set

Limiting the Result Set

- By default, a query will return ALL rows from the Table listed after the FROM clause

Limiting the Result Set

- 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

Limiting the Result Set

- 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

Limiting the Result Set

- 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

Limiting the Result Set

- 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
  - DISTINCT is called a result set qualifier

Limiting the Result Set

- **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**
  - DISTINCT is called a result set qualifier
  - This limits the result set to all the distinct values rather than all the values

Limiting the Result Set

- **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**
  - DISTINCT is called a result set qualifier
  - 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

Limiting the Result Set

- 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
  - DISTINCT is called a result set qualifier
  - 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?
```

Limiting the Result Set

- 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
  - DISTINCT is called a result set qualifier
  - 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;
```

WHERE

WHERE

- The next clause you can add to a query is the “WHERE” clause

WHERE

- The next clause you can add to a query is the “WHERE” clause
- The WHERE clause is like a search

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

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**

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

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**

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?
```

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

Simple Operators

- The key with WHERE clause expressions is to know how all the operators in SQL work

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

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.

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

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

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

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

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

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

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

AND and OR

- Each boolean expression can be combined with other boolean expressions

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 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**

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?
```

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

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

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?

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’;
```

More Operators

More Operators

- These operators may not be as familiar to you

More Operators

- **These operators may not be as familiar to you**
  - The first set of operators we looked at are very similar to operators from arithmetic

More Operators

- **These operators may not be as familiar to you**
  - The first set of operators we looked at are very similar to operators from arithmetic
- **These operators start to get a bit deeper into dealing with sets in a sophisticated way**

BETWEEN

BETWEEN

- BETWEEN is an operator that acts on a column and two values

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

- **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"

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?

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

- LIKE is a special operator just for strings

LIKE

- **LIKE is a special operator just for strings**
  - You give LIKE a pattern and it will match strings that match that pattern

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

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

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?
```

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

IN

- An expression with IN requires a column and a list of potential values

IN

- **An expression with IN requires a column and a list of potential values**
  - Values can be numeric or string or date

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

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**

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?
```

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 is a special operator that helps address when a value in a column might be NULL

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

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?

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 NOT is the complement to IS

IS NOT

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

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

Summary

- The SELECT command in SQL has more power than it seems

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