SQL/MySQL in Customer Relationship Management

Chu Junhong

NUS Business School

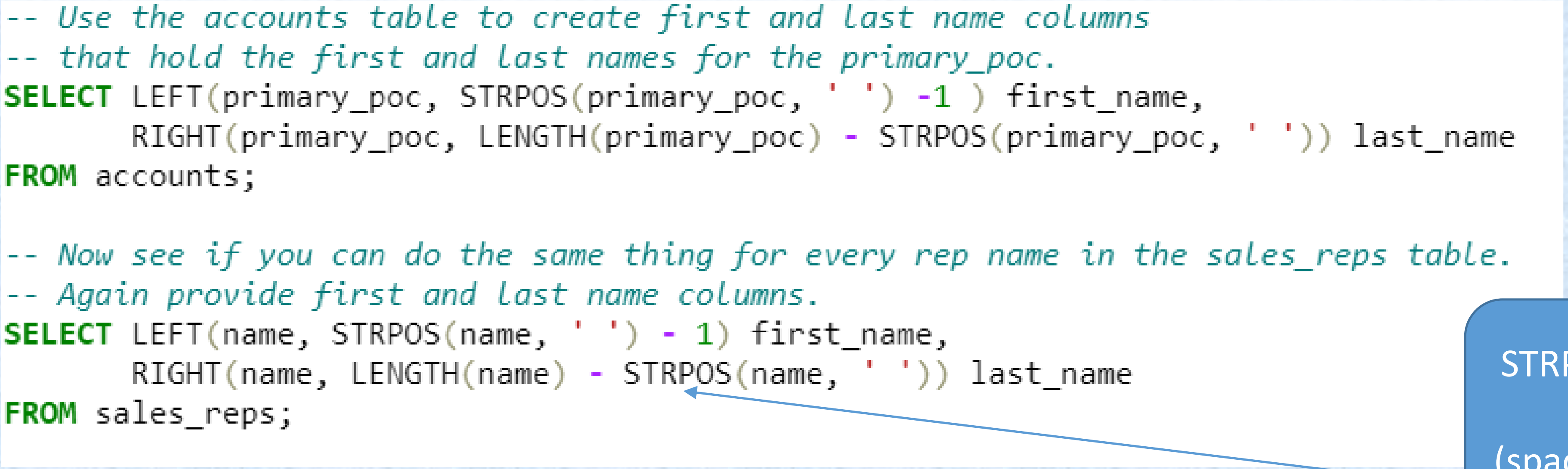
Junhong.chu@nus.edu.sg

Table of Content

1. SQL basics
2. SQL JOINs
3. SQL AGGREGATIONs
4. SQL subqueries and temporary tables
5. SQL data cleaning
6. SQL WINDOW functions

Part V Data Cleaning (1)

- **LEFT:** pulls a specified number of characters in a specified column from the left **LEFT(phone_number, 3)** (Same in Excel).
- **RIGHT:** pulls a specified number of characters in a specified column from the right **RIGHT(phone_number, 8)** (same in Excel).
- **LENGTH:** provides the number of characters for each row of a specified column. **LENGTH(phone_number)** (LEN in Excel)



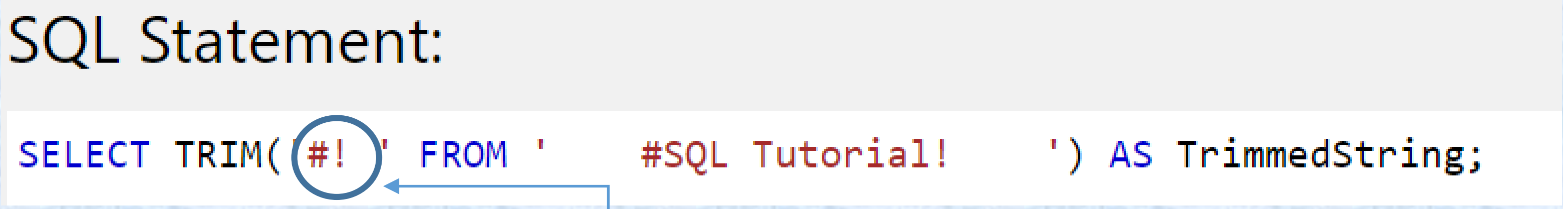
```
-- Use the accounts table to create first and last name columns
-- that hold the first and last names for the primary_poc.
SELECT LEFT(primary_poc, STRPOS(primary_poc, ' ') -1 ) first_name,
       RIGHT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, ' ')) last_name
FROM accounts;

-- Now see if you can do the same thing for every rep name in the sales_reps table.
-- Again provide first and last name columns.
SELECT LEFT(name, STRPOS(name, ' ') - 1) first_name,
       RIGHT(name, LENGTH(name) - STRPOS(name, ' ')) last_name
FROM sales_reps;
```

STRPOS(var, “ ”) returns the index of the “ “ (space, some character, etc.)

Part V Data Cleaning (2)

- **~~TRIM~~**: remove characters from the beginning and end of a string like unwanted spaces at the beginning or end of a row
- Syntax: TRIM([*characters* FROM]*string*)
- Example
  - **TRIM**('   SQL Tutorial!   ') AS TrimmedString;

    => SQL Tutorial!



SQL Statement:

```
SELECT TRIM('#! ' FROM '    #SQL Tutorial!    ') AS TrimmedString;
```

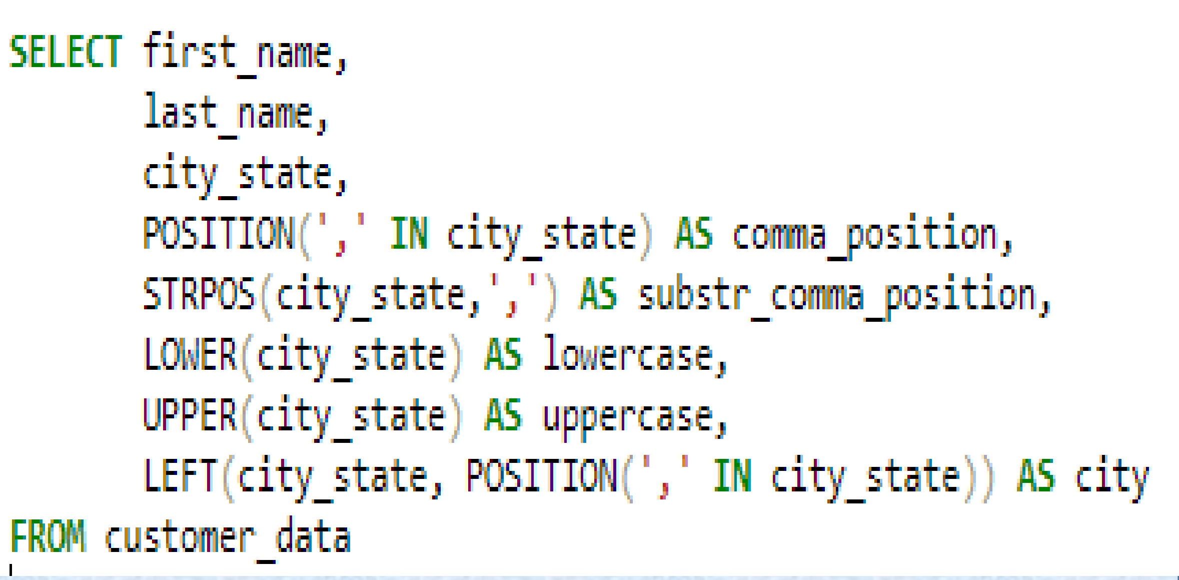
=> SQL Tutorial

Removes '#! ' from the string

More Functions for Data Cleaning

- **POSITION**: provides the index of a character in a variable, **POSITION(',' IN** city_state**)**
- **STRPOS**: provides the same result as **POSITION**, **STRPOS(**city_state**, ',')**
- Both **POSITION** and **STRPOS** are case sensitive, **A** is NOT **a**
- **LOWER** or **UPPER** to make all of the characters lower or uppercase.
  - UPPER
  - LOWER
- **SUBSTR**(text, 7, 4) (text, start, length)

SUBSTR('Junhong Chu', 7, 4) => 'g Ch'



```
SELECT first_name,
       last_name,
       city_state,
       POSITION(',' IN city_state) AS comma_position,
       STRPOS(city_state,',') AS substr_comma_position,
       LOWER(city_state) AS lowercase,
       UPPER(city_state) AS uppercase,
       LEFT(city_state, POSITION(',' IN city_state)) AS city
FROM customer_data
```

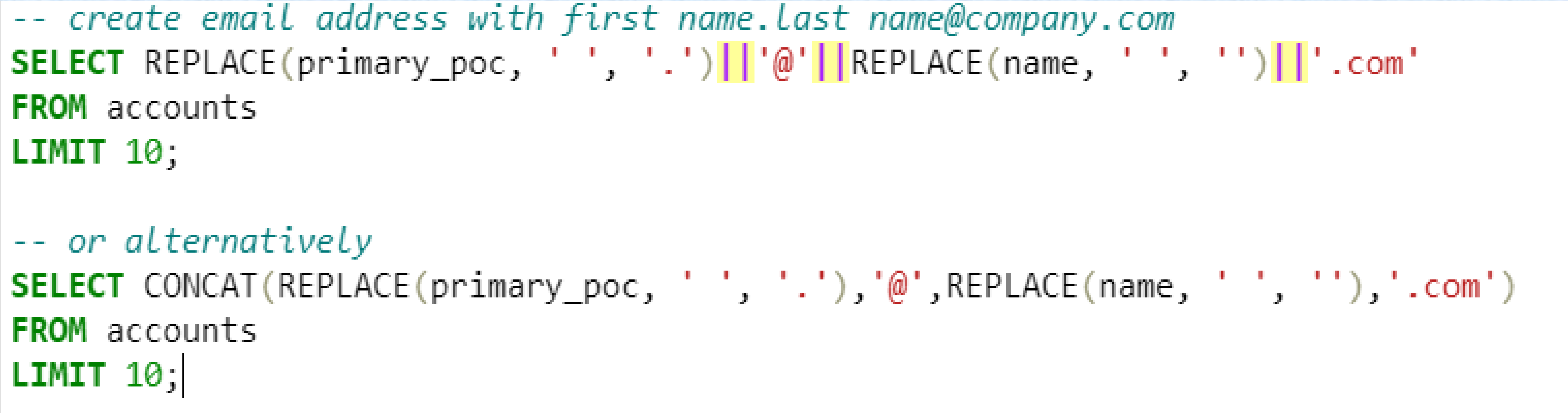
More Functions on Data Cleaning

- **REPLACE(‘Junhong Chu’, ‘ ‘, ‘.’): replace the space in “Junhong Chu” with a “.” => Junhong.Chu**

Replace the empty space ‘ ‘ with ‘.’

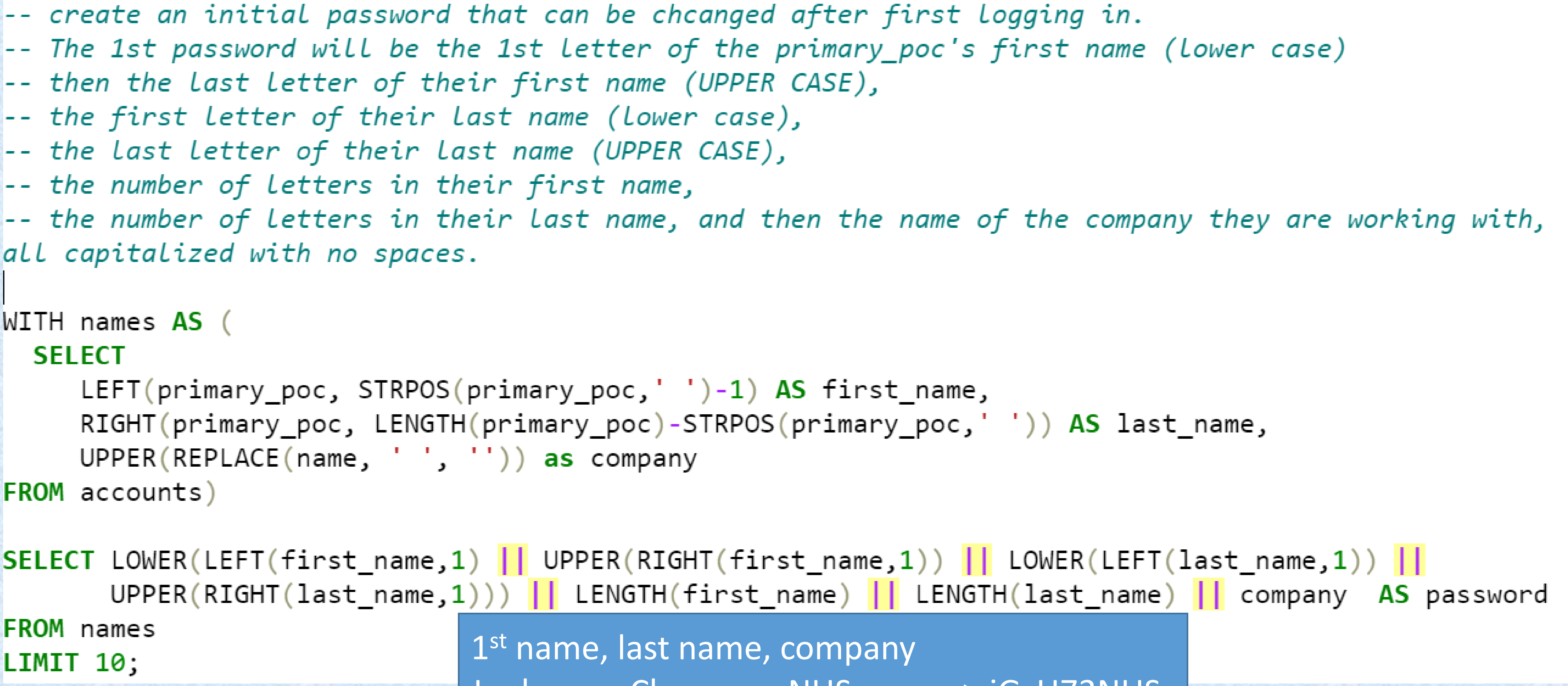
- **CONCAT or** Piping sign “**||**”: connect text together:
  - **CONCAT**(‘Junhong’, ‘.’, ‘Chu’) => Junhong.Chu
  - ‘Junhong’ **||** ‘.’ **||** ‘Chu’ => Junhong.Chu

More Functions on Data Cleaning: Example 1



```
-- create email address with first name.last name@company.com
SELECT REPLACE(primary_poc, ' ', '.')||'@'||REPLACE(name, ' ', '')||'.com'
FROM accounts
LIMIT 10;

-- or alternatively
SELECT CONCAT(REPLACE(primary_poc, ' ', '.'),'@',REPLACE(name, ' ', ''),'.com')
FROM accounts
LIMIT 10;
```

More Functions on Data Cleaning: Example 2

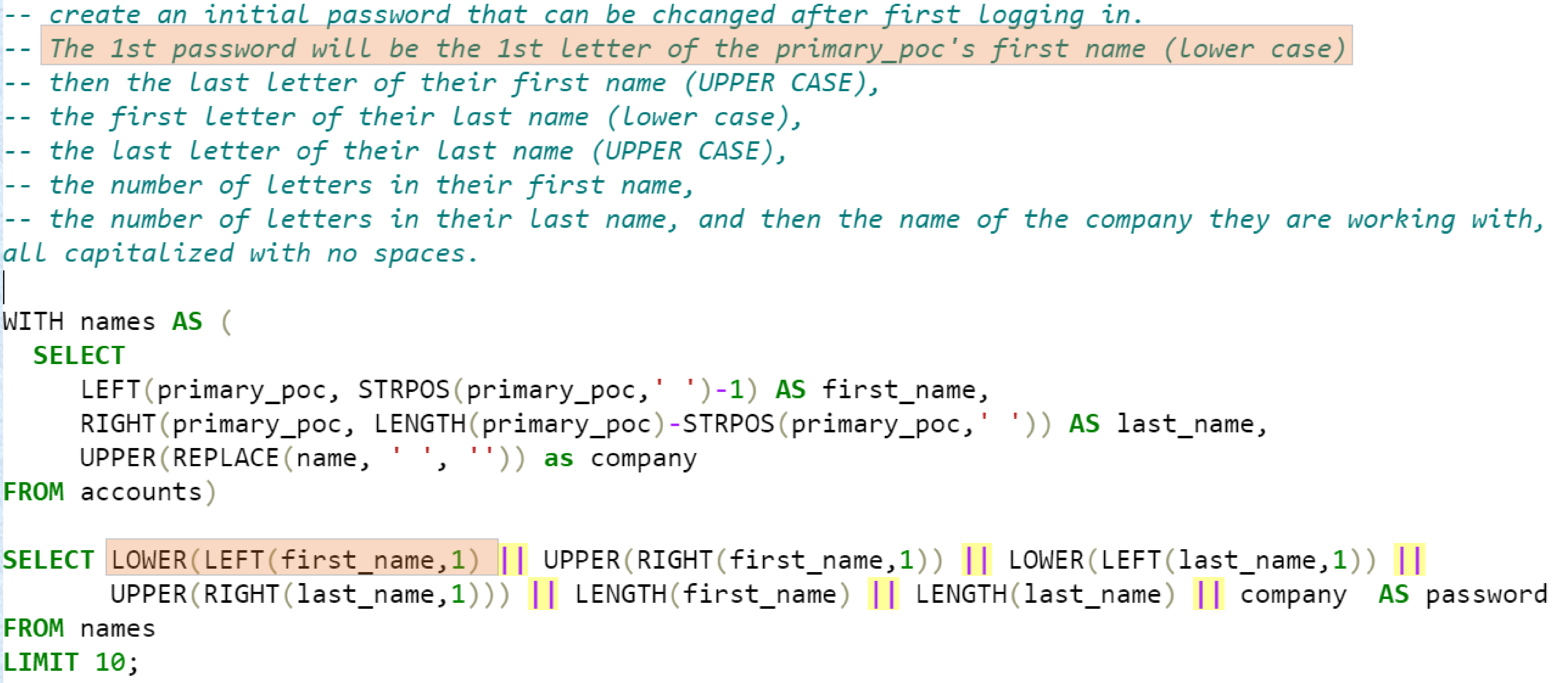


```
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
       UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company  AS password
FROM names
LIMIT 10;
```

1st name, last name, company

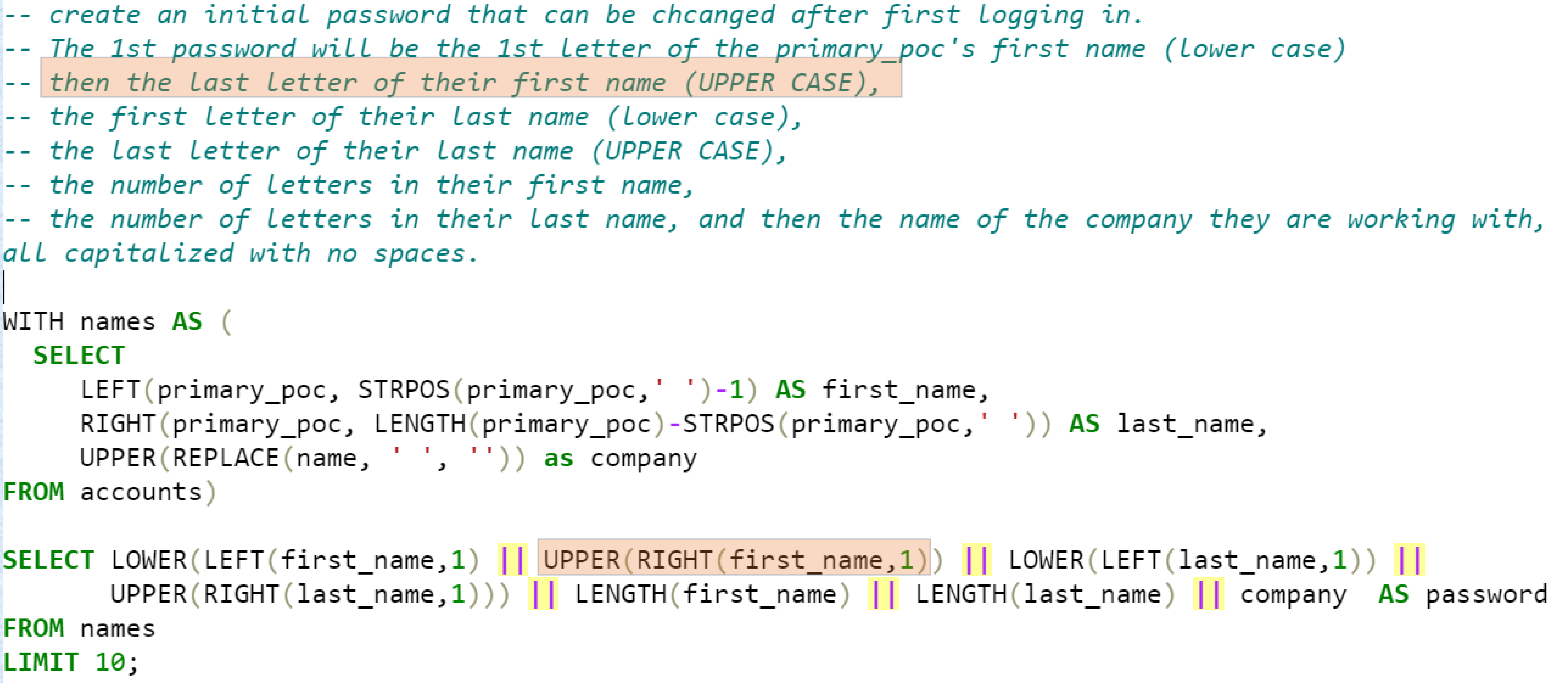
Junhong   Chu   NUS   => jGcU73NUS

More Functions on Data Cleaning: Example 2



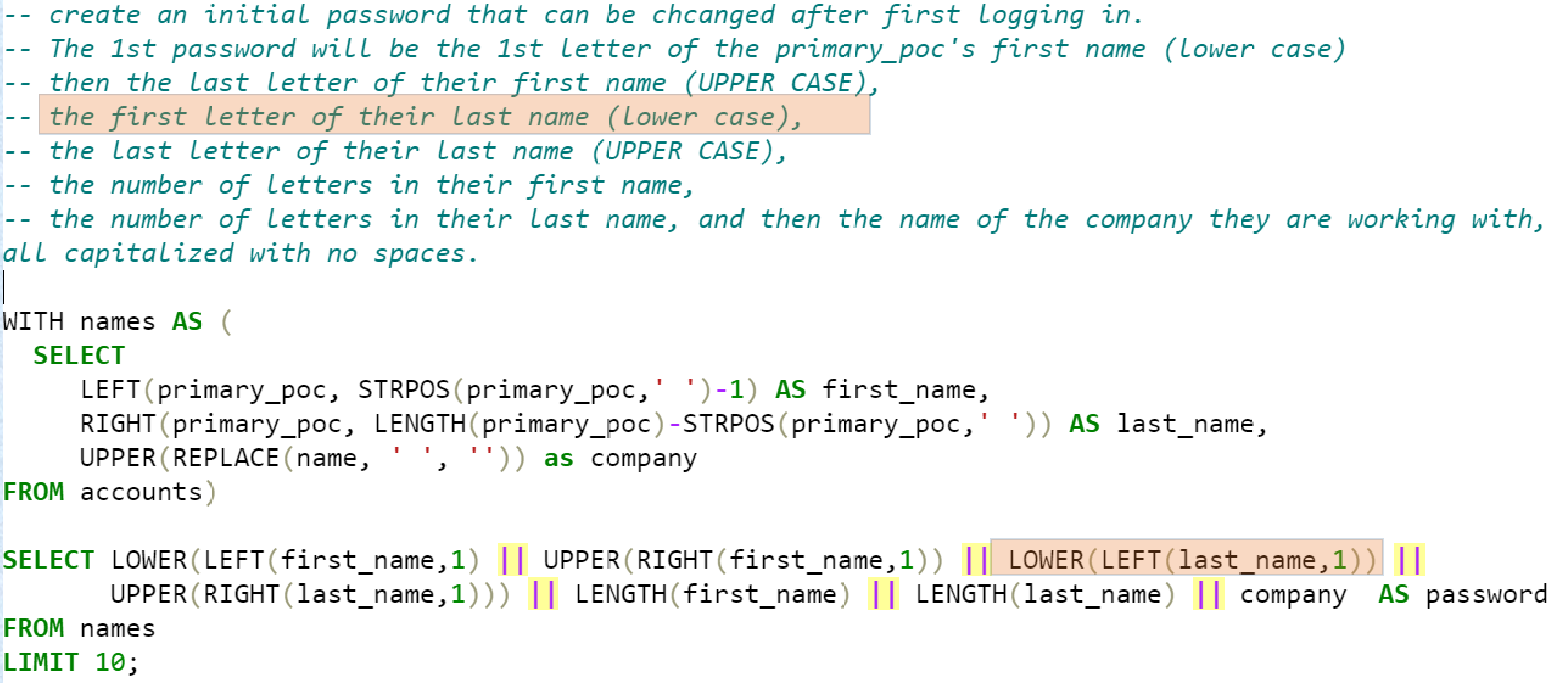
```
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
       UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company  AS password
FROM names
LIMIT 10;
```

More Functions on Data Cleaning: Example 2



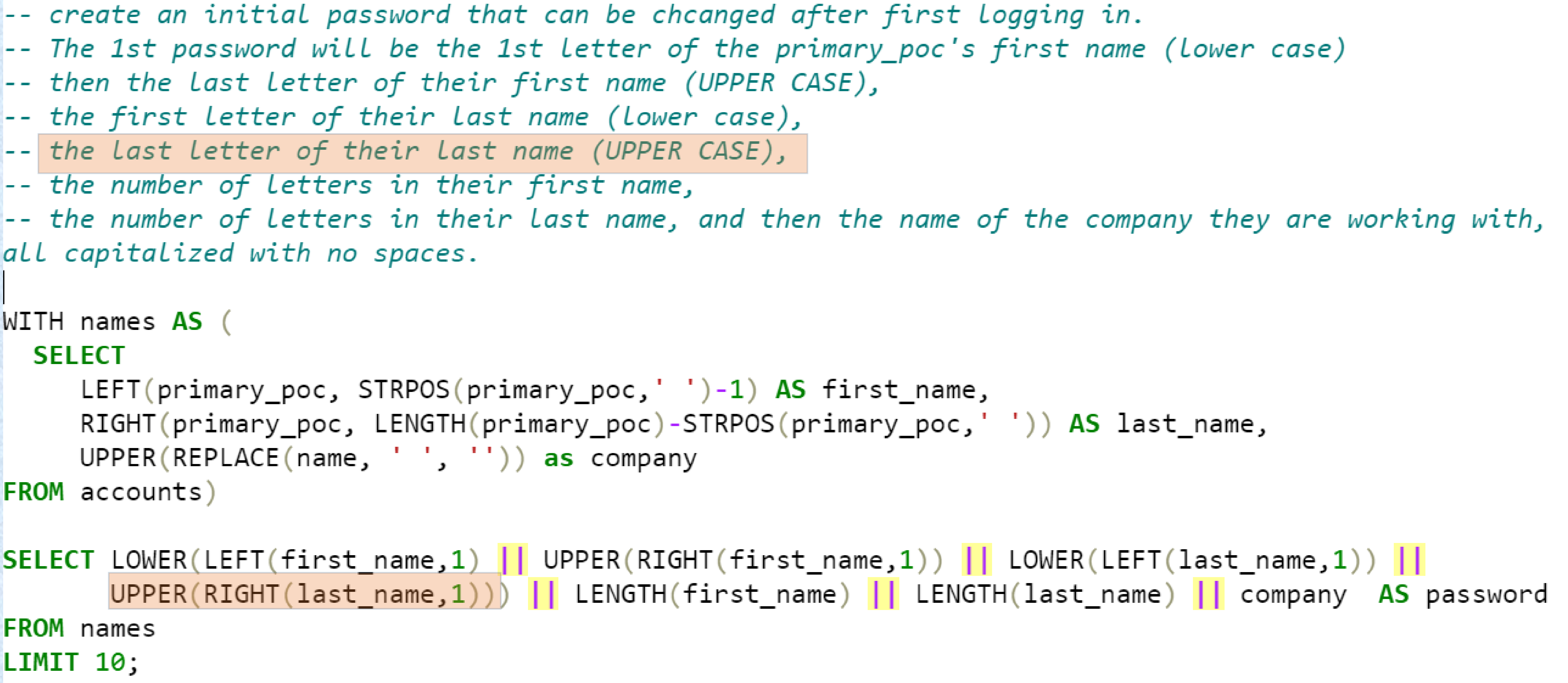
```
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
       UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company  AS password
FROM names
LIMIT 10;
```

More Functions on Data Cleaning: Example 2



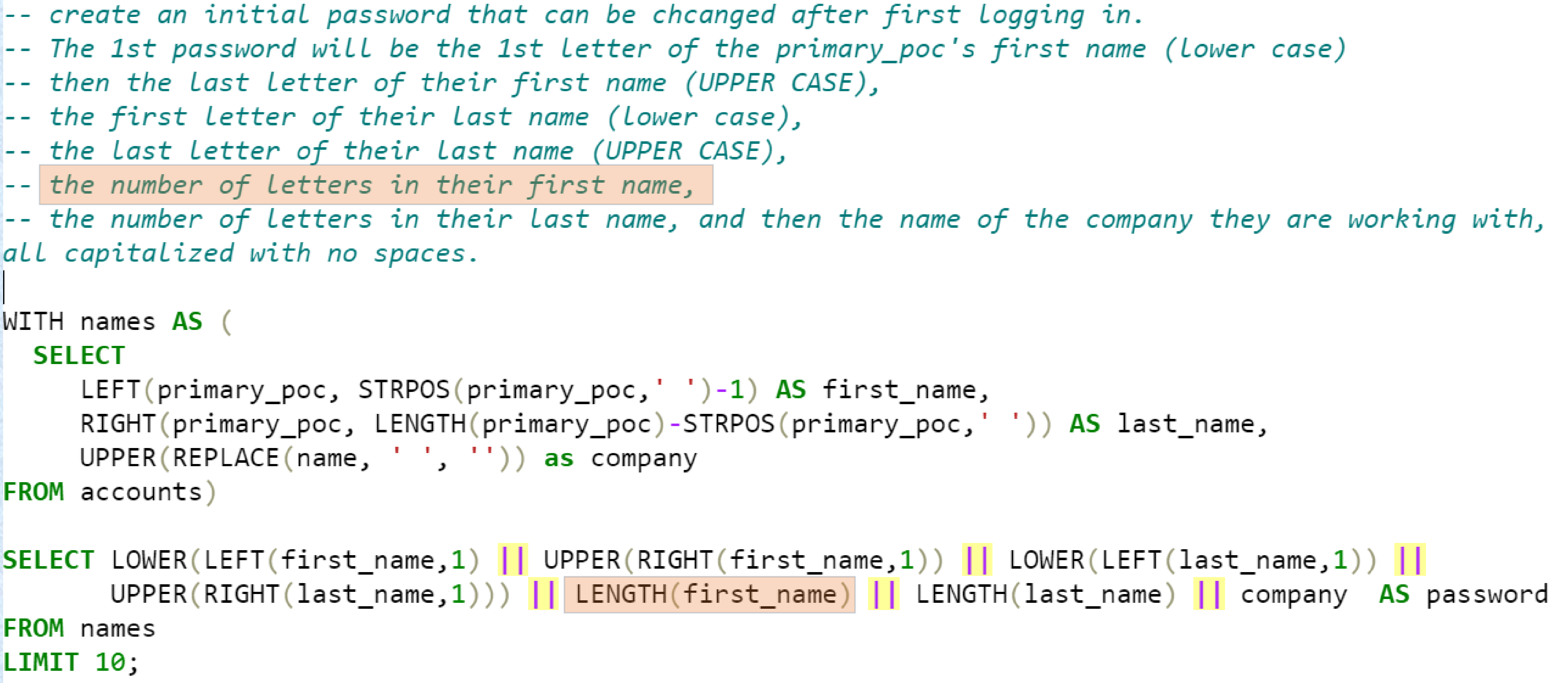
```
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
       UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company  AS password
FROM names
LIMIT 10;
```

More Functions on Data Cleaning: Example 2



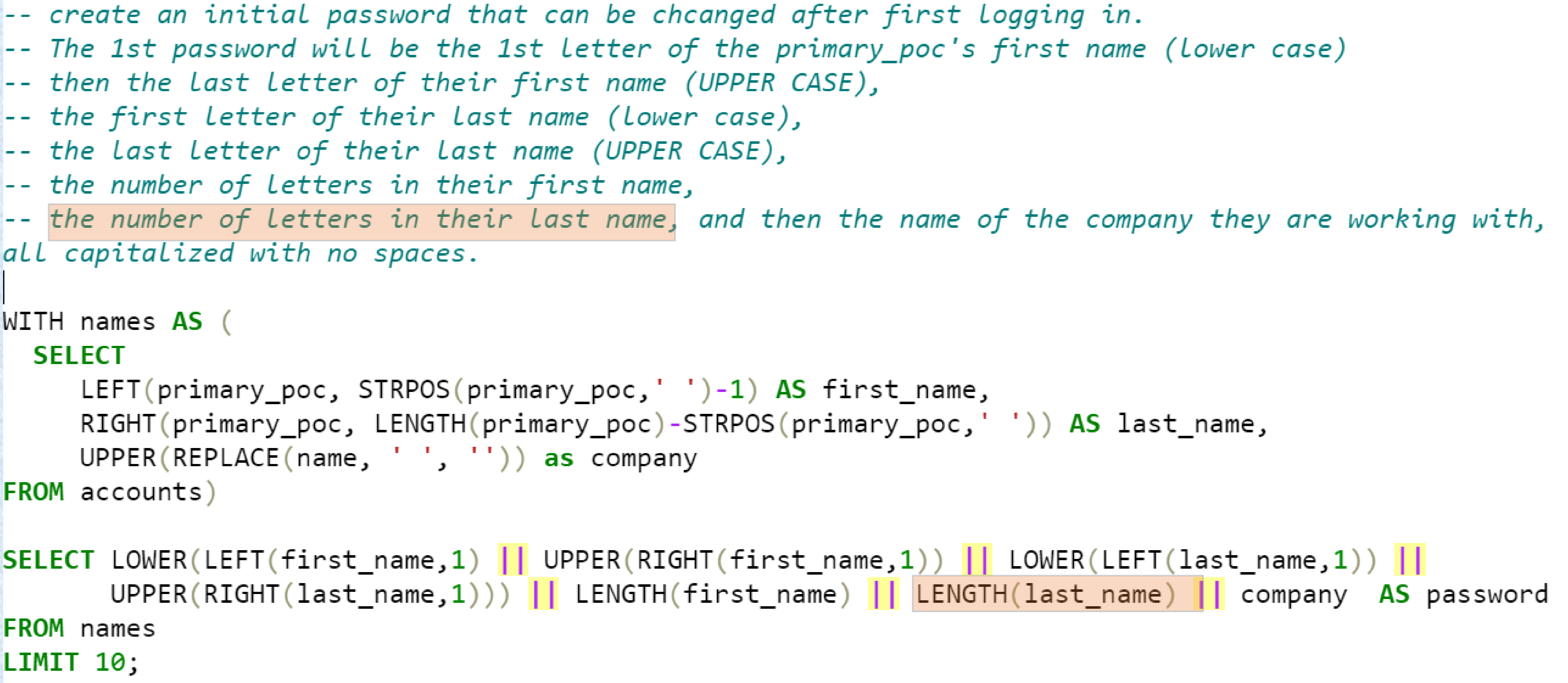
```
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1)) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
        UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company  AS password
FROM names
LIMIT 10;
```

More Functions on Data Cleaning: Example 2



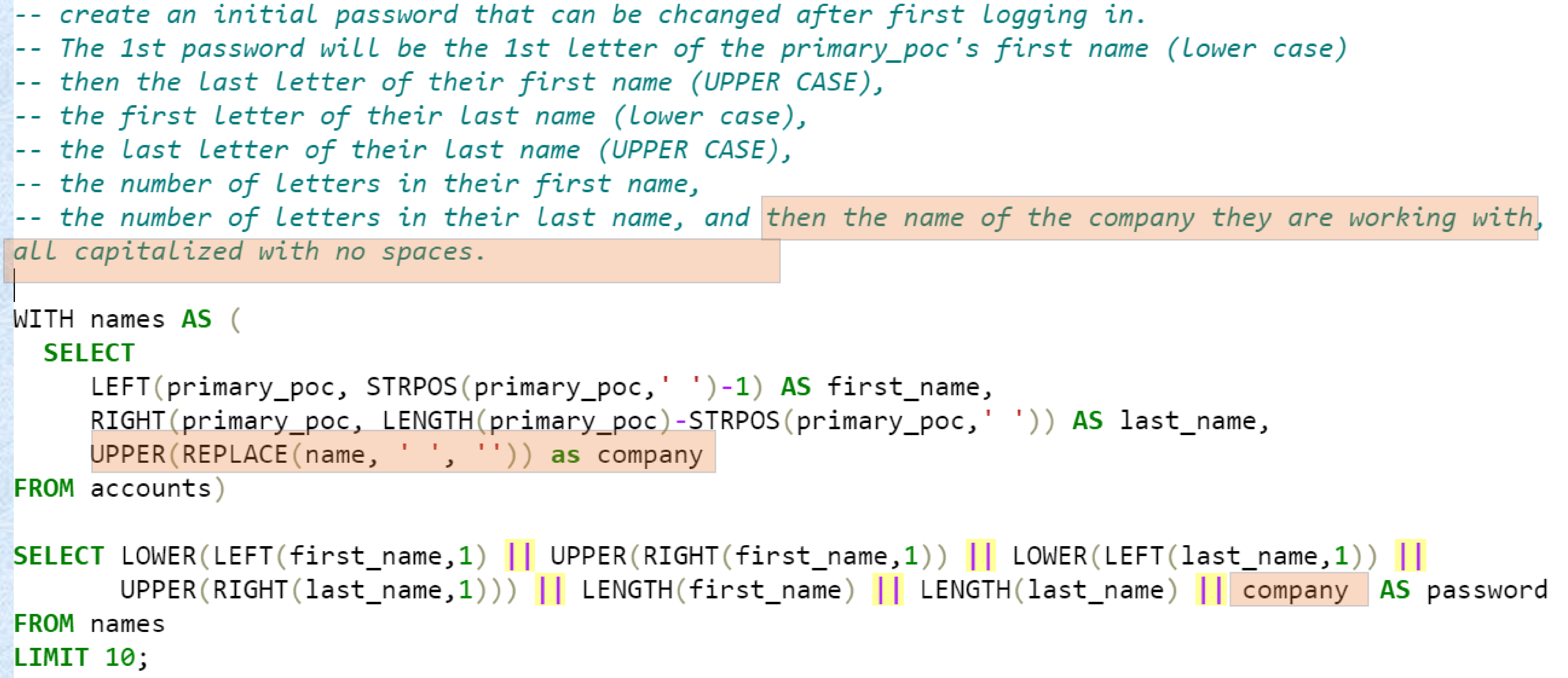
```
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
       UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company  AS password
FROM names
LIMIT 10;
```

More Functions on Data Cleaning: Example 2



```sql
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
       UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company  AS password
FROM names
LIMIT 10;
```

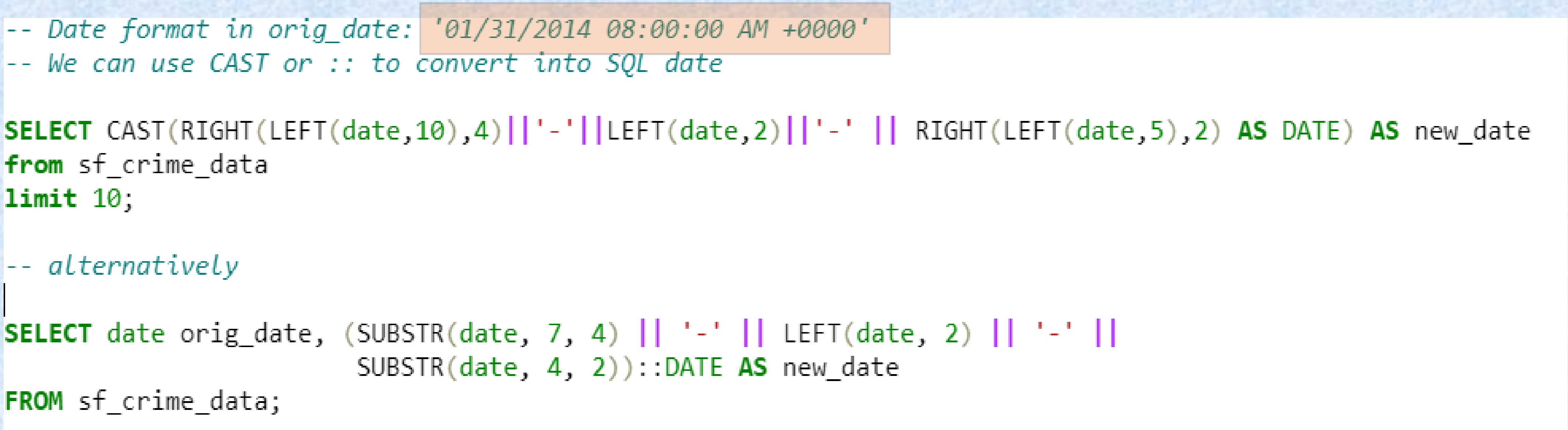
More Functions on Data Cleaning: Example 2



```
-- create an initial password that can be chcanged after first logging in.
-- The 1st password will be the 1st letter of the primary_poc's first name (lower case)
-- then the last letter of their first name (UPPER CASE),
-- the first letter of their last name (lower case),
-- the last letter of their last name (UPPER CASE),
-- the number of letters in their first name,
-- the number of letters in their last name, and then the name of the company they are working with,
all capitalized with no spaces.

WITH names AS (
  SELECT
     LEFT(primary_poc, STRPOS(primary_poc,' ')-1) AS first_name,
     RIGHT(primary_poc, LENGTH(primary_poc)-STRPOS(primary_poc,' ')) AS last_name,
     UPPER(REPLACE(name, ' ', '')) as company
FROM accounts)

SELECT LOWER(LEFT(first_name,1) || UPPER(RIGHT(first_name,1)) || LOWER(LEFT(last_name,1)) ||
       UPPER(RIGHT(last_name,1))) || LENGTH(first_name) || LENGTH(last_name) || company AS password
FROM names
LIMIT 10;
```

Convert string into SQL date stamp

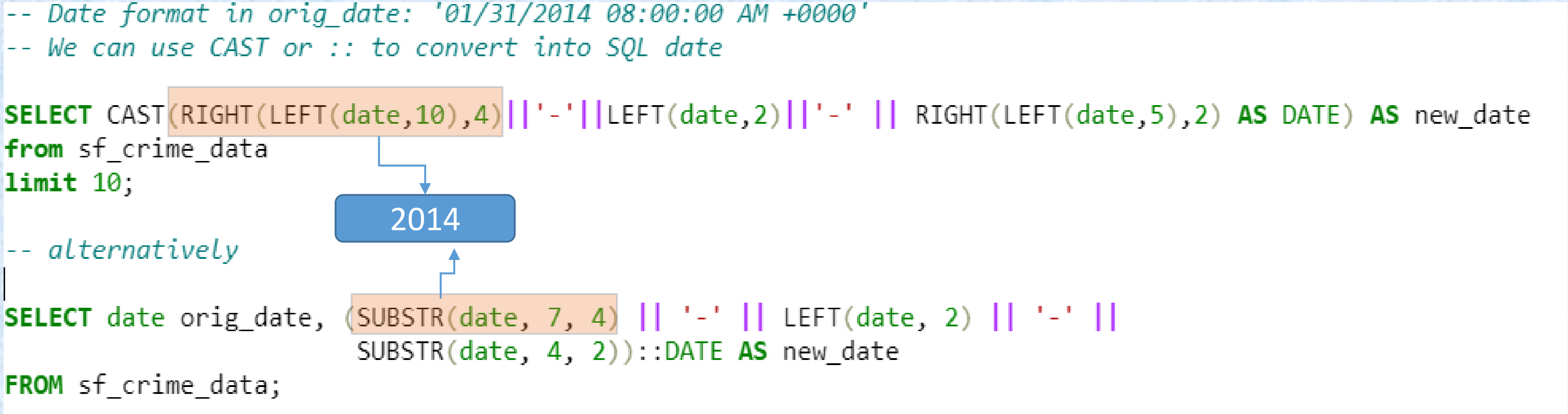
- **CAST:** **CAST**(‘2020-11-14’ **AS DATE) AS** sql_date
- Casting with **::** **date_column::DATE**.
- **STR_TO_DATE**("August 10 2017", "%M %d %Y") => 2017-08-10
- **CAST** is actually useful to change lots of column types. You can make other changes to your columns in terms of their data types. You can see other examples **here**.



```
-- Date format in orig_date: '01/31/2014 08:00:00 AM +0000'
-- We can use CAST or :: to convert into SQL date

SELECT CAST(RIGHT(LEFT(date,10),4)||'-'||LEFT(date,2)||'-' || RIGHT(LEFT(date,5),2) AS DATE) AS new_date
from sf_crime_data
limit 10;

-- alternatively

SELECT date orig_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' ||
                        SUBSTR(date, 4, 2))::DATE AS new_date
FROM sf_crime_data;
```

Convert string into SQL date stamp

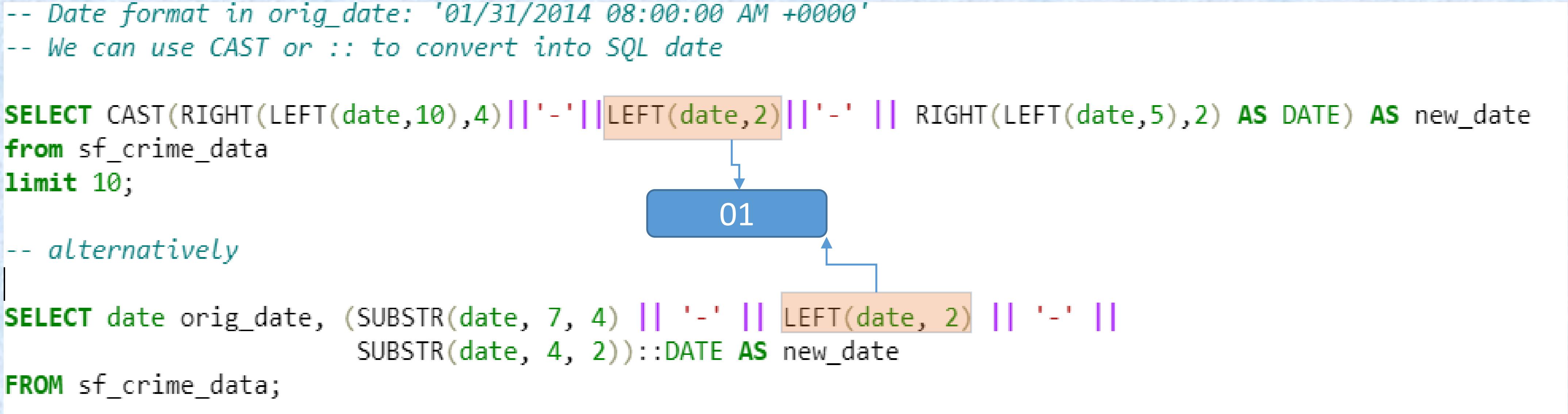
- **CAST: CAST**(‘2020-11-14’ **AS DATE) AS** sql_date
- Casting with **::** **date_column::DATE**.
- **STR_TO_DATE**("August 10 2017", "%M %d %Y") => 2017-08-10
- **CAST** is actually useful to change lots of column types. You can make other changes to your columns in terms of their data types. You can see other examples **here**.



```
-- Date format in orig_date: '01/31/2014 08:00:00 AM +0000'
-- We can use CAST or :: to convert into SQL date

SELECT CAST(RIGHT(LEFT(date,10),4)||'-'||LEFT(date,2)||'-' || RIGHT(LEFT(date,5),2) AS DATE) AS new_date
from sf_crime_data
limit 10;

-- alternatively

SELECT date orig_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' ||
                        SUBSTR(date, 4, 2))::DATE AS new_date
FROM sf_crime_data;
```

Convert string into SQL date stamp

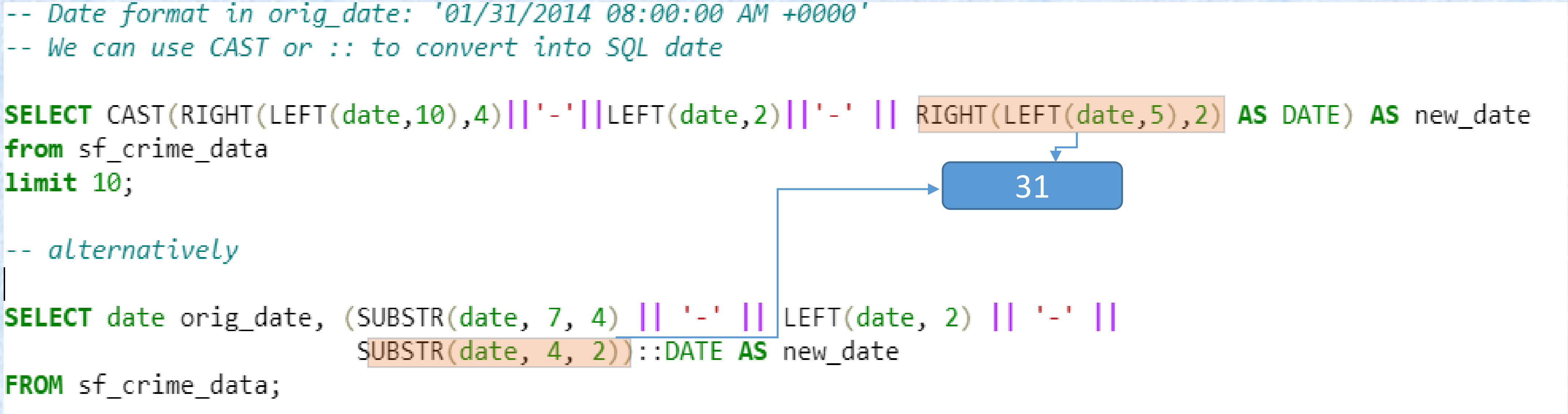
- **CAST: CAST**(‘2020-11-14’ **AS DATE) AS** sql_date
- Casting with **::** **date_column::DATE**.
- **STR_TO_DATE**("August 10 2017", "%M %d %Y") => 2017-08-10
- **CAST** is actually useful to change lots of column types. You can make other changes to your columns in terms of their data types. You can see other examples **here**.



```
-- Date format in orig_date: '01/31/2014 08:00:00 AM +0000'
-- We can use CAST or :: to convert into SQL date

SELECT CAST(RIGHT(LEFT(date,10),4)||'-'||LEFT(date,2)||'-' || RIGHT(LEFT(date,5),2) AS DATE) AS new_date
from sf_crime_data
limit 10;

-- alternatively

SELECT date orig_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' ||
                        SUBSTR(date, 4, 2))::DATE AS new_date
FROM sf_crime_data;
```

Convert string into SQL date stamp

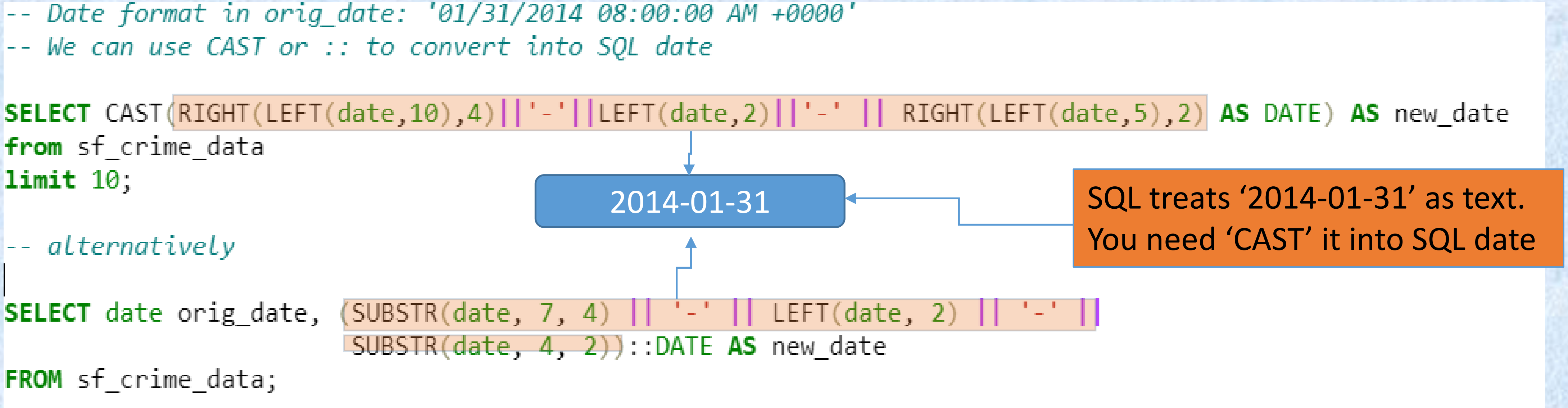
- **CAST: CAST**(‘2020-11-14’ **AS DATE) AS** sql_date
- Casting with **::** **date_column::DATE**.
- **STR_TO_DATE**("August 10 2017", "%M %d %Y") => 2017-08-10
- **CAST** is actually useful to change lots of column types. You can make other changes to your columns in terms of their data types. You can see other examples **here**.



```
-- Date format in orig_date: '01/31/2014 08:00:00 AM +0000'
-- We can use CAST or :: to convert into SQL date

SELECT CAST(RIGHT(LEFT(date,10),4)||'-'||LEFT(date,2)||'-' || RIGHT(LEFT(date,5),2) AS DATE) AS new_date
from sf_crime_data
limit 10;

-- alternatively

SELECT date orig_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' ||
                        SUBSTR(date, 4, 2))::DATE AS new_date
FROM sf_crime_data;
```

Convert string into SQL date stamp

- **CAST: CAST**(‘2020-11-14’ **AS DATE) AS** sql_date
- Casting with **::** **date_column::DATE**.
- **STR_TO_DATE**("August 10 2017", "%M %d %Y") => 2017-08-10
- **CAST** is actually useful to change lots of column types. You can make other changes to your columns in terms of their data types. You can see other examples **here**.



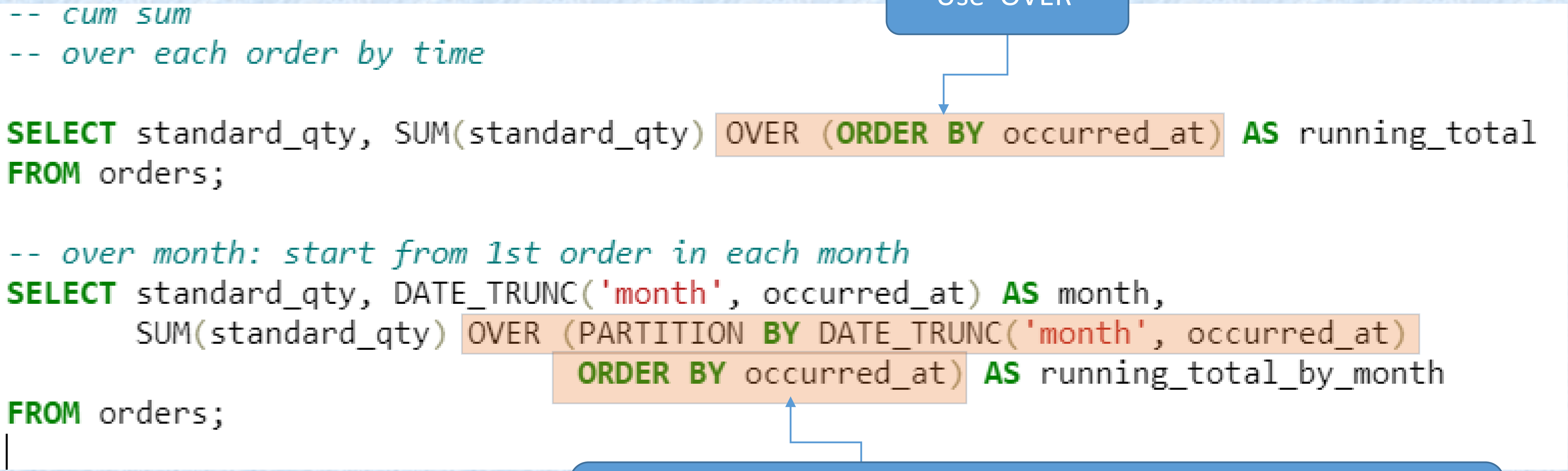
```
-- Date format in orig_date: '01/31/2014 08:00:00 AM +0000'
-- We can use CAST or :: to convert into SQL date

SELECT CAST(RIGHT(LEFT(date,10),4)||'-'||LEFT(date,2)||'-' || RIGHT(LEFT(date,5),2) AS DATE) AS new_date
from sf_crime_data
limit 10;

-- alternatively

SELECT date orig_date, (SUBSTR(date, 7, 4) || '-' || LEFT(date, 2) || '-' ||
                        SUBSTR(date, 4, 2))::DATE AS new_date
FROM sf_crime_data;
```

2014-01-31

SQL treats ‘2014-01-31’ as text. You need ‘CAST’ it into SQL date

PART VI The WINDOW functions

- Compute cumulative sum for each month



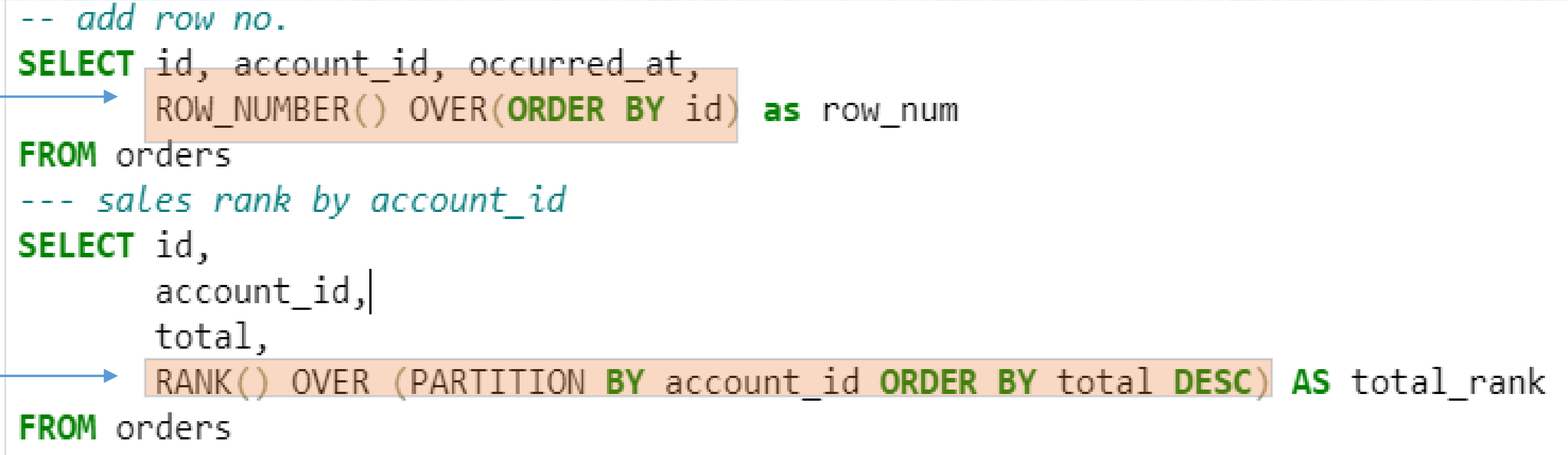
```
-- cum sum
-- over each order by time

SELECT standard_qty, SUM(standard_qty) OVER (ORDER BY occurred_at) AS running_total
FROM orders;

-- over month: start from 1st order in each month
SELECT standard_qty, DATE_TRUNC('month', occurred_at) AS month,
       SUM(standard_qty) OVER (PARTITION BY DATE_TRUNC('month', occurred_at)
                               ORDER BY occurred_at) AS running_total_by_month
FROM orders;
```

Use ‘OVER’

Step 1: Partition the data by ‘month’
Step 2: SUM over orders within each month (restart for each month)

ROW_NUMBER, RANK, DENSE_RANK functions

- Row_number()

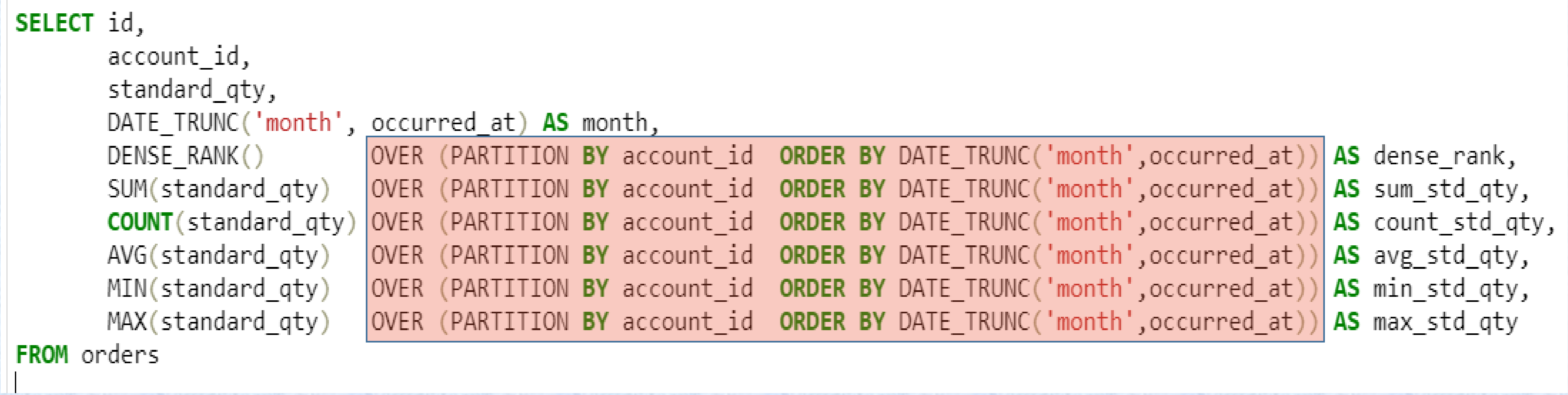


```
-- add row no.
SELECT id, account_id, occurred_at,
       ROW_NUMBER() OVER(ORDER BY id) as row_num
FROM orders
--- sales rank by account_id
SELECT id,
       account_id,
       total,
       RANK() OVER (PARTITION BY account_id ORDER BY total DESC) AS total_rank
FROM orders
```

- Rank(): assign same ranks to same value, but skip if there are ties: 1, 2, 2, 4, 5, 5, 7

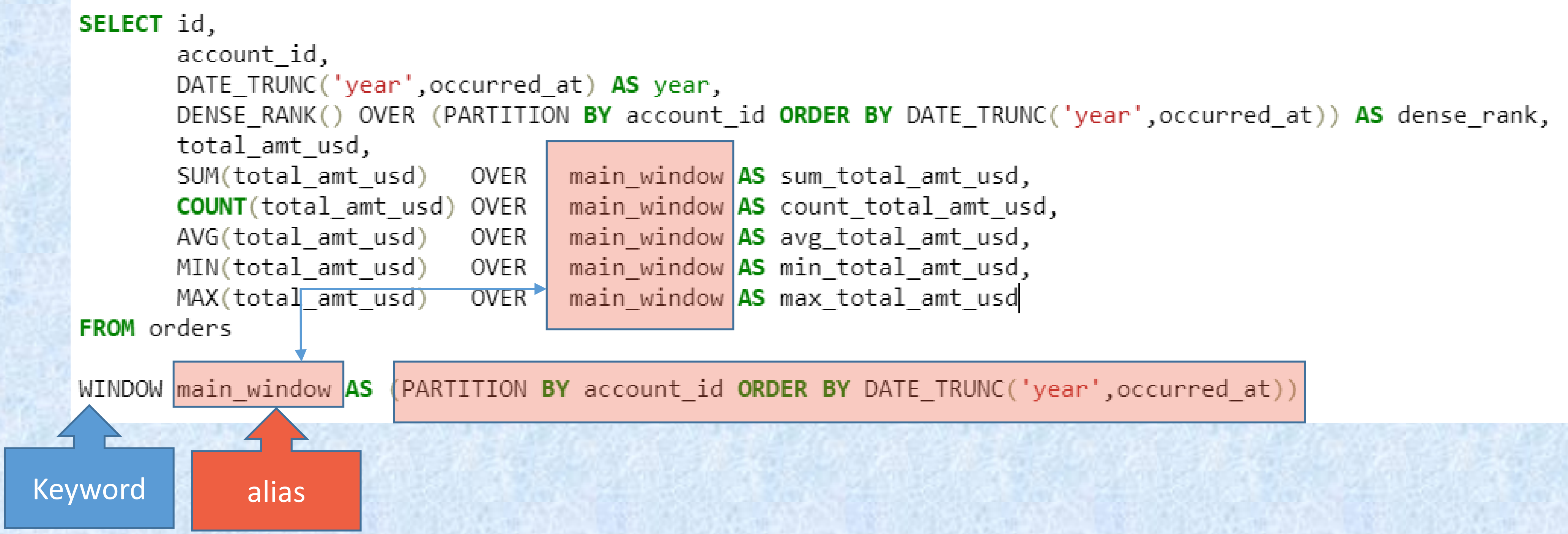
- Dense_rank(): no skipping if there are ties: 1, 2, 2, 3, 4,4, 5

Monthly sales rank



```
SELECT id,
       account_id,
       standard_qty,
       DATE_TRUNC('month', occurred_at) AS month,
       DENSE_RANK()        OVER (PARTITION BY account_id  ORDER BY DATE_TRUNC('month',occurred_at)) AS dense_rank,
       SUM(standard_qty)   OVER (PARTITION BY account_id  ORDER BY DATE_TRUNC('month',occurred_at)) AS sum_std_qty,
       COUNT(standard_qty) OVER (PARTITION BY account_id  ORDER BY DATE_TRUNC('month',occurred_at)) AS count_std_qty,
       AVG(standard_qty)   OVER (PARTITION BY account_id  ORDER BY DATE_TRUNC('month',occurred_at)) AS avg_std_qty,
       MIN(standard_qty)   OVER (PARTITION BY account_id  ORDER BY DATE_TRUNC('month',occurred_at)) AS min_std_qty,
       MAX(standard_qty)   OVER (PARTITION BY account_id  ORDER BY DATE_TRUNC('month',occurred_at)) AS max_std_qty
FROM orders
```

Use alias to replace repetitive part



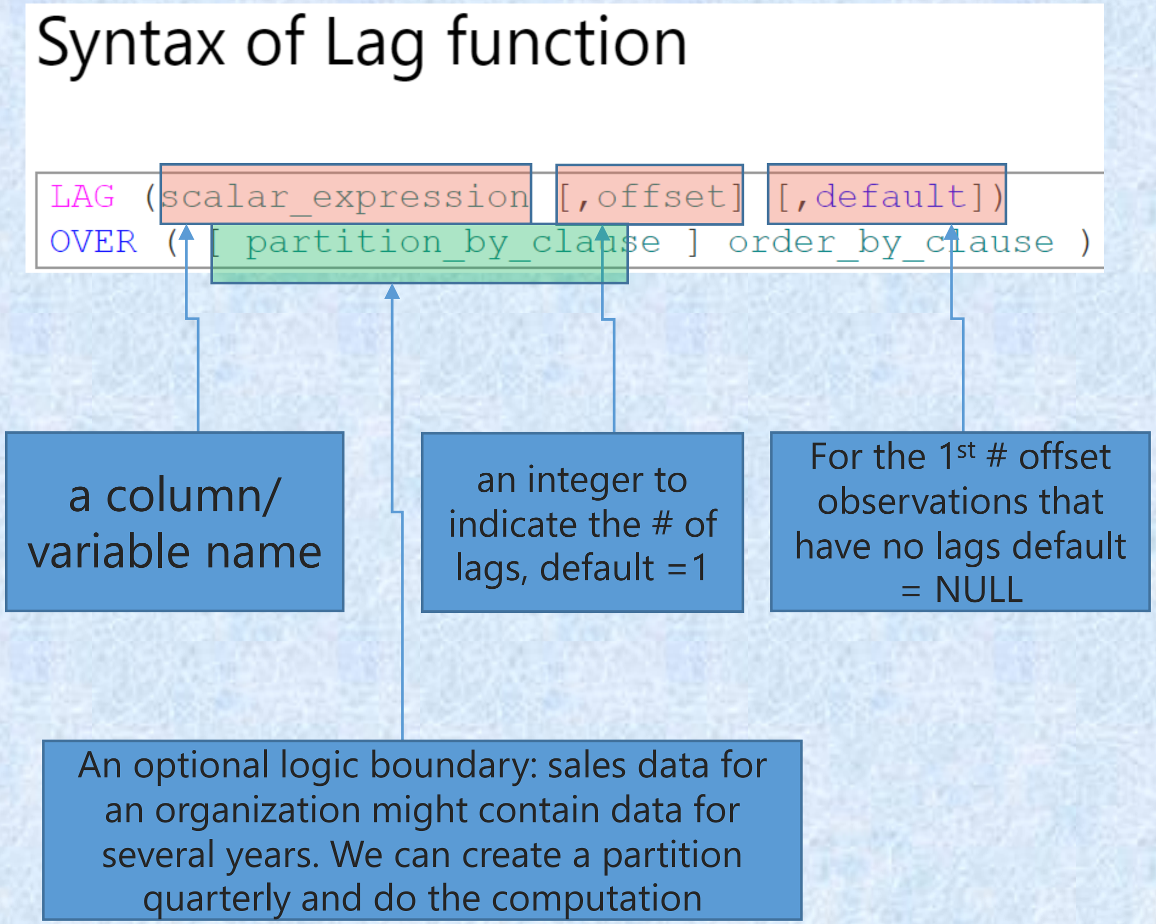
```
SELECT id,
       account_id,
       DATE_TRUNC('year',occurred_at) AS year,
       DENSE_RANK() OVER (PARTITION BY account_id ORDER BY DATE_TRUNC('year',occurred_at)) AS dense_rank,
       total_amt_usd,
       SUM(total_amt_usd)   OVER   main_window AS sum_total_amt_usd,
       COUNT(total_amt_usd) OVER   main_window AS count_total_amt_usd,
       AVG(total_amt_usd)   OVER   main_window AS avg_total_amt_usd,
       MIN(total_amt_usd)   OVER   main_window AS min_total_amt_usd,
       MAX(total_amt_usd)   OVER   main_window AS max_total_amt_usd
FROM orders

WINDOW main_window AS (PARTITION BY account_id ORDER BY DATE_TRUNC('year',occurred_at))
```

Keyword

alias

must put it below the main syntax

LEAD and LAG functions



Syntax of Lag function

```
LAG (scalar_expression [,offset] [,default])
OVER ( [ partition_by_clause ] order_by_clause )
```

a column/ variable name

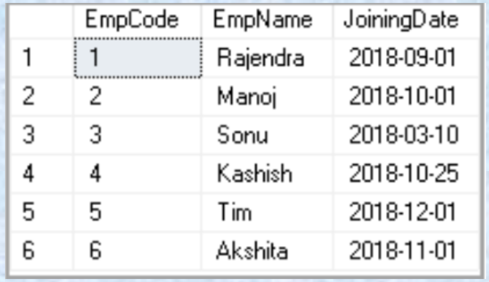
an integer to indicate the # of lags, default =1

For the 1st # offset observations that have no lags default = NULL

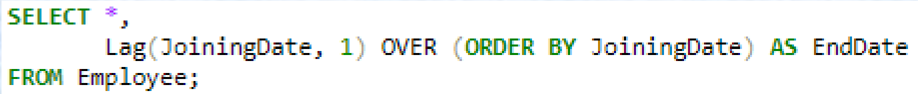
An optional logic boundary: sales data for an organization might contain data for several years. We can create a partition quarterly and do the computation

LEAD and LAG functions: Examples

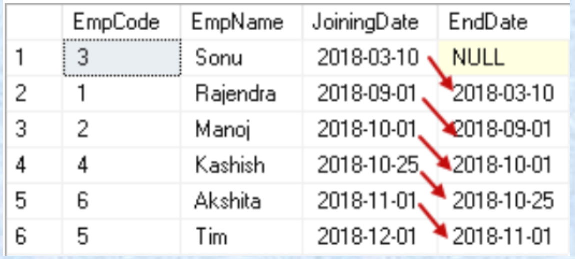
Data:



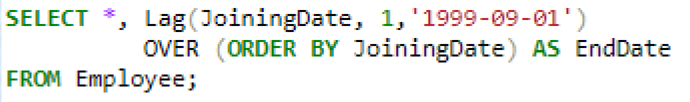
	EmpCode	EmpName	JoiningDate
1	1	Rajendra	2018-09-01
2	2	Manoj	2018-10-01
3	3	Sonu	2018-03-10
4	4	Kashish	2018-10-25
5	5	Tim	2018-12-01
6	6	Akshita	2018-11-01



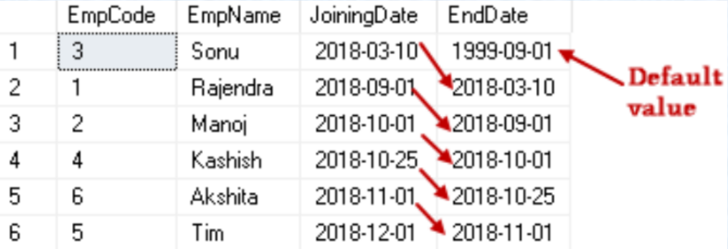
```
SELECT *,
       Lag(JoiningDate, 1) OVER (ORDER BY JoiningDate) AS EndDate
FROM Employee;
```



	EmpCode	EmpName	JoiningDate	EndDate
1	3	Sonu	2018-03-10	NULL
2	1	Rajendra	2018-09-01	2018-03-10
3	2	Manoj	2018-10-01	2018-09-01
4	4	Kashish	2018-10-25	2018-10-01
5	6	Akshita	2018-11-01	2018-10-25
6	5	Tim	2018-12-01	2018-11-01

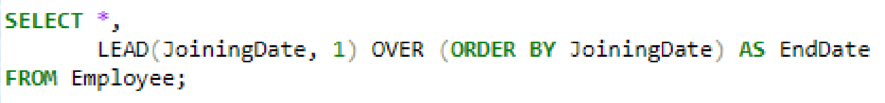


```
SELECT *, Lag(JoiningDate, 1,'1999-09-01')
        OVER (ORDER BY JoiningDate) AS EndDate
FROM Employee;
```

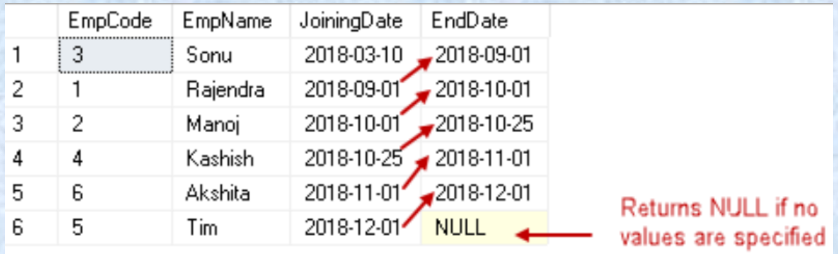


	EmpCode	EmpName	JoiningDate	EndDate
1	3	Sonu	2018-03-10	1999-09-01
2	1	Rajendra	2018-09-01	2018-03-10
3	2	Manoj	2018-10-01	2018-09-01
4	4	Kashish	2018-10-25	2018-10-01
5	6	Akshita	2018-11-01	2018-10-25
6	5	Tim	2018-12-01	2018-11-01

Default value

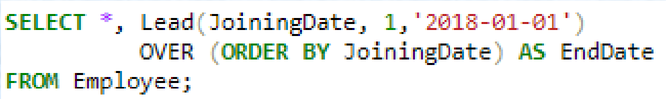


```
SELECT *,
       LEAD(JoiningDate, 1) OVER (ORDER BY JoiningDate) AS EndDate
FROM Employee;
```

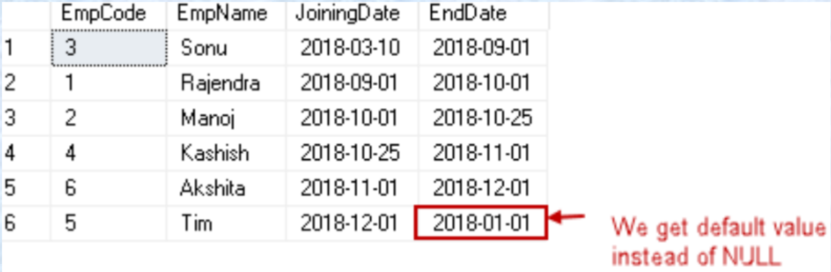


	EmpCode	EmpName	JoiningDate	EndDate
1	3	Sonu	2018-03-10	2018-09-01
2	1	Rajendra	2018-09-01	2018-10-01
3	2	Manoj	2018-10-01	2018-10-25
4	4	Kashish	2018-10-25	2018-11-01
5	6	Akshita	2018-11-01	2018-12-01
6	5	Tim	2018-12-01	NULL

Returns NULL if no values are specified

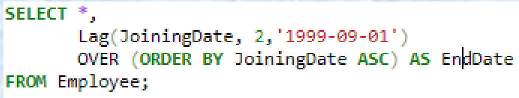


```
SELECT *, Lead(JoiningDate, 1,'2018-01-01')
        OVER (ORDER BY JoiningDate) AS EndDate
FROM Employee;
```

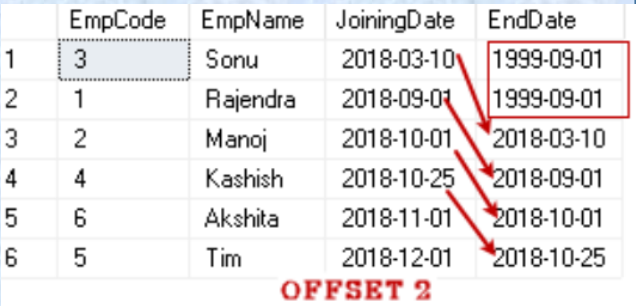


	EmpCode	EmpName	JoiningDate	EndDate
1	3	Sonu	2018-03-10	2018-09-01
2	1	Rajendra	2018-09-01	2018-10-01
3	2	Manoj	2018-10-01	2018-10-25
4	4	Kashish	2018-10-25	2018-11-01
5	6	Akshita	2018-11-01	2018-12-01
6	5	Tim	2018-12-01	2018-01-01

We get default value instead of NULL



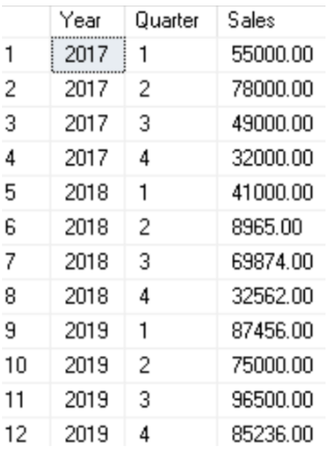
```
SELECT *,
       Lag(JoiningDate, 2,'1999-09-01')
       OVER (ORDER BY JoiningDate ASC) AS EndDate
FROM Employee;
```



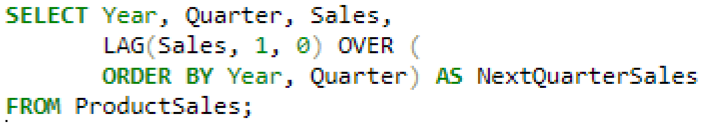
	EmpCode	EmpName	JoiningDate	EndDate
1	3	Sonu	2018-03-10	1999-09-01
2	1	Rajendra	2018-09-01	1999-09-01
3	2	Manoj	2018-10-01	2018-03-10
4	4	Kashish	2018-10-25	2018-09-01
5	6	Akshita	2018-11-01	2018-10-01
6	5	Tim	2018-12-01	2018-10-25

OFFSET 2

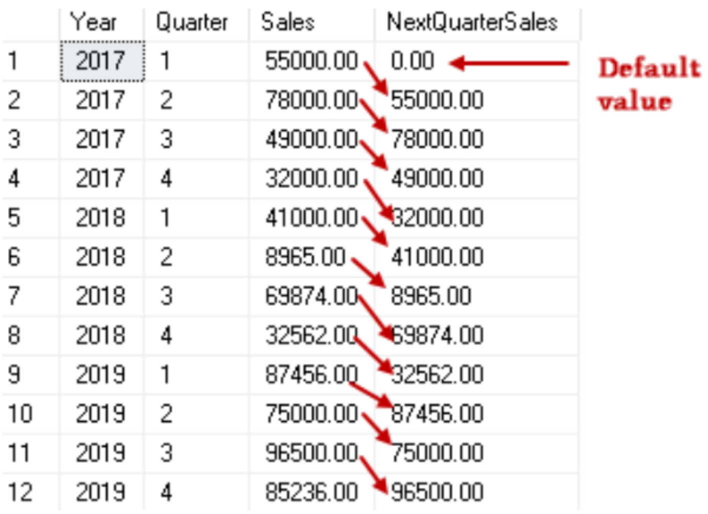
LEAD and LAG functions: Examples



	Year	Quarter	Sales
1	2017	1	55000.00
2	2017	2	78000.00
3	2017	3	49000.00
4	2017	4	32000.00
5	2018	1	41000.00
6	2018	2	8965.00
7	2018	3	69874.00
8	2018	4	32562.00
9	2019	1	87456.00
10	2019	2	75000.00
11	2019	3	96500.00
12	2019	4	85236.00

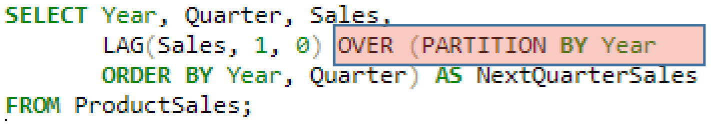


```
SELECT Year, Quarter, Sales,
       LAG(Sales, 1, 0) OVER (
       ORDER BY Year, Quarter) AS NextQuarterSales
FROM ProductSales;
```

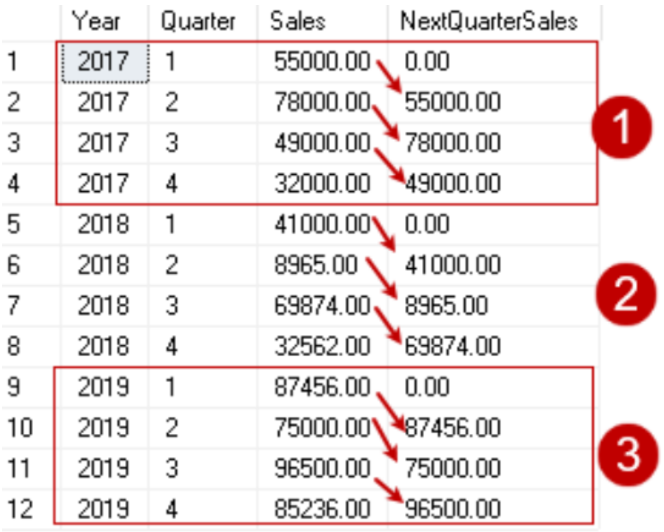


	Year	Quarter	Sales	NextQuarterSales
1	2017	1	55000.00	0.00
2	2017	2	78000.00	55000.00
3	2017	3	49000.00	78000.00
4	2017	4	32000.00	49000.00
5	2018	1	41000.00	32000.00
6	2018	2	8965.00	41000.00
7	2018	3	69874.00	8965.00
8	2018	4	32562.00	69874.00
9	2019	1	87456.00	32562.00
10	2019	2	75000.00	87456.00
11	2019	3	96500.00	75000.00
12	2019	4	85236.00	96500.00

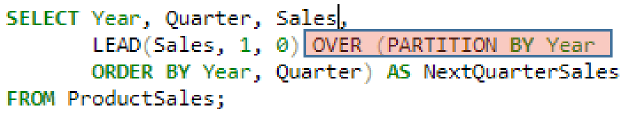
Default value



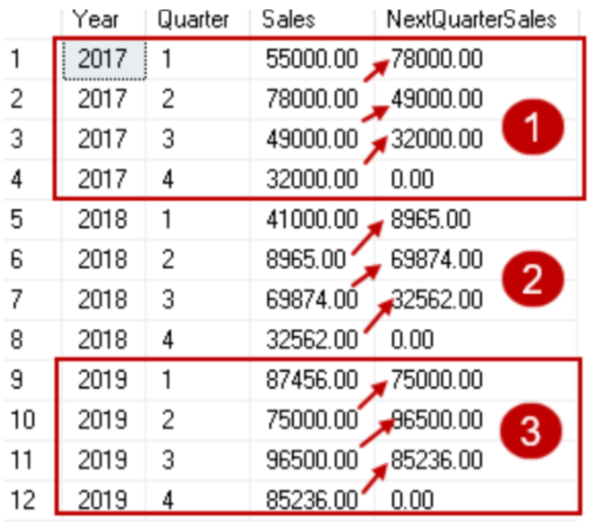
```
SELECT Year, Quarter, Sales,
       LAG(Sales, 1, 0) OVER (PARTITION BY Year
       ORDER BY Year, Quarter) AS NextQuarterSales
FROM ProductSales;
```



	Year	Quarter	Sales	NextQuarterSales
1	2017	1	55000.00	0.00
2	2017	2	78000.00	55000.00
3	2017	3	49000.00	78000.00
4	2017	4	32000.00	49000.00
5	2018	1	41000.00	0.00
6	2018	2	8965.00	41000.00
7	2018	3	69874.00	8965.00
8	2018	4	32562.00	69874.00
9	2019	1	87456.00	0.00
10	2019	2	75000.00	87456.00
11	2019	3	96500.00	75000.00
12	2019	4	85236.00	96500.00



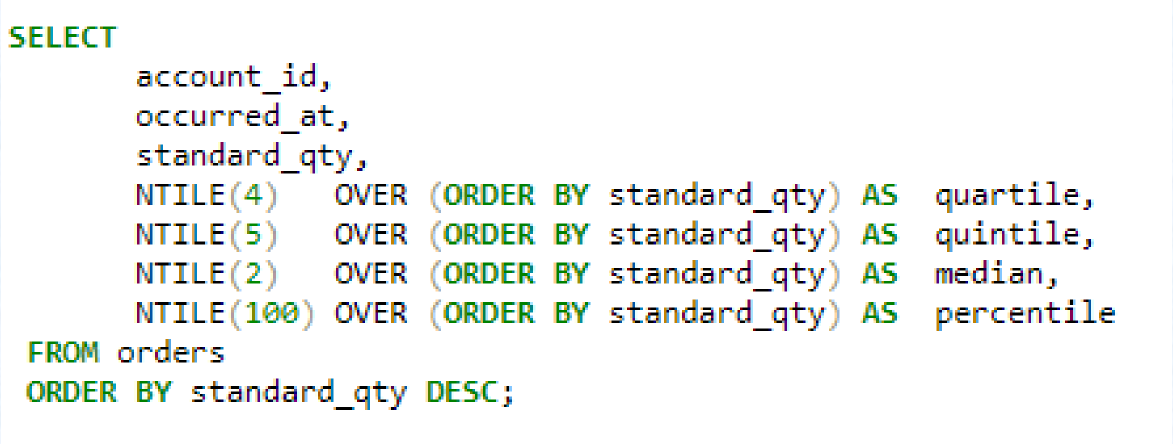
```
SELECT Year, Quarter, Sales,
       LEAD(Sales, 1, 0) OVER (PARTITION BY Year
       ORDER BY Year, Quarter) AS NextQuarterSales
FROM ProductSales;
```



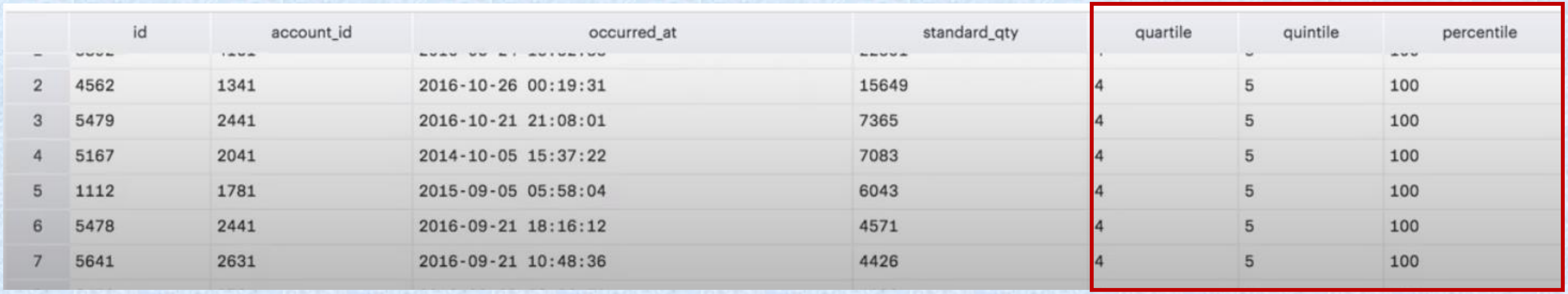
	Year	Quarter	Sales	NextQuarterSales
1	2017	1	55000.00	78000.00
2	2017	2	78000.00	49000.00
3	2017	3	49000.00	32000.00
4	2017	4	32000.00	0.00
5	2018	1	41000.00	8965.00
6	2018	2	8965.00	69874.00
7	2018	3	69874.00	32562.00
8	2018	4	32562.00	0.00
9	2019	1	87456.00	75000.00
10	2019	2	75000.00	96500.00
11	2019	3	96500.00	85236.00
12	2019	4	85236.00	0.00

Lead function on PARTITION for Year column

NTILE function (quartile, median, percentile)

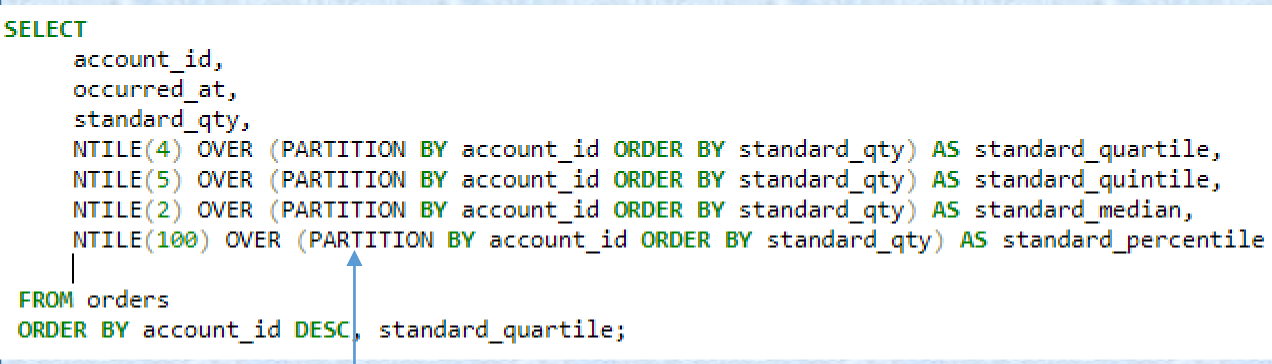


```
SELECT
       account_id,
       occurred_at,
       standard_qty,
       NTILE(4)   OVER (ORDER BY standard_qty) AS  quartile,
       NTILE(5)   OVER (ORDER BY standard_qty) AS  quintile,
       NTILE(2)   OVER (ORDER BY standard_qty) AS  median,
       NTILE(100) OVER (ORDER BY standard_qty) AS  percentile
 FROM orders
 ORDER BY standard_qty DESC;
```



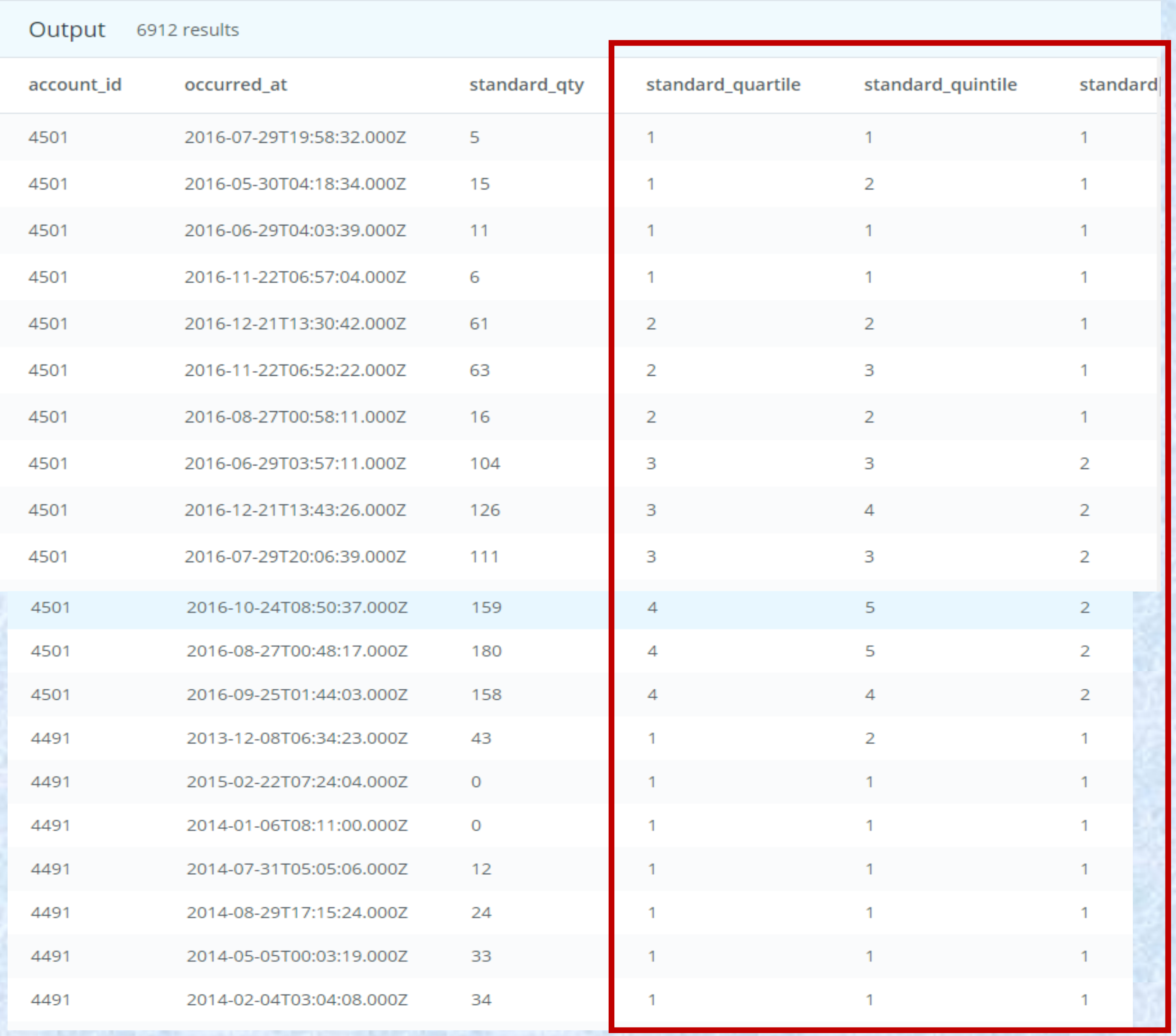
	id	account_id	occurred_at	standard_qty	quartile	quintile	percentile
2	4562	1341	2016-10-26 00:19:31	15649	4	5	100
3	5479	2441	2016-10-21 21:08:01	7365	4	5	100
4	5167	2041	2014-10-05 15:37:22	7083	4	5	100
5	1112	1781	2015-09-05 05:58:04	6043	4	5	100
6	5478	2441	2016-09-21 18:16:12	4571	4	5	100
7	5641	2631	2016-09-21 10:48:36	4426	4	5	100

NTILE function (quartile, median, percentile)



```
SELECT
     account_id,
     occurred_at,
     standard_qty,
     NTILE(4) OVER (PARTITION BY account_id ORDER BY standard_qty) AS standard_quartile,
     NTILE(5) OVER (PARTITION BY account_id ORDER BY standard_qty) AS standard_quintile,
     NTILE(2) OVER (PARTITION BY account_id ORDER BY standard_qty) AS standard_median,
     NTILE(100) OVER (PARTITION BY account_id ORDER BY standard_qty) AS standard_percentile

 FROM orders
 ORDER BY account_id DESC, standard_quartile;
```

1. PARTITION the data by account_id: You get NTILE for each account_id, which has the effect as "GROUP BY"
2. Sort the data in ascending order by "ORDER BY"
3. Use "OVER" to find out the NTILE

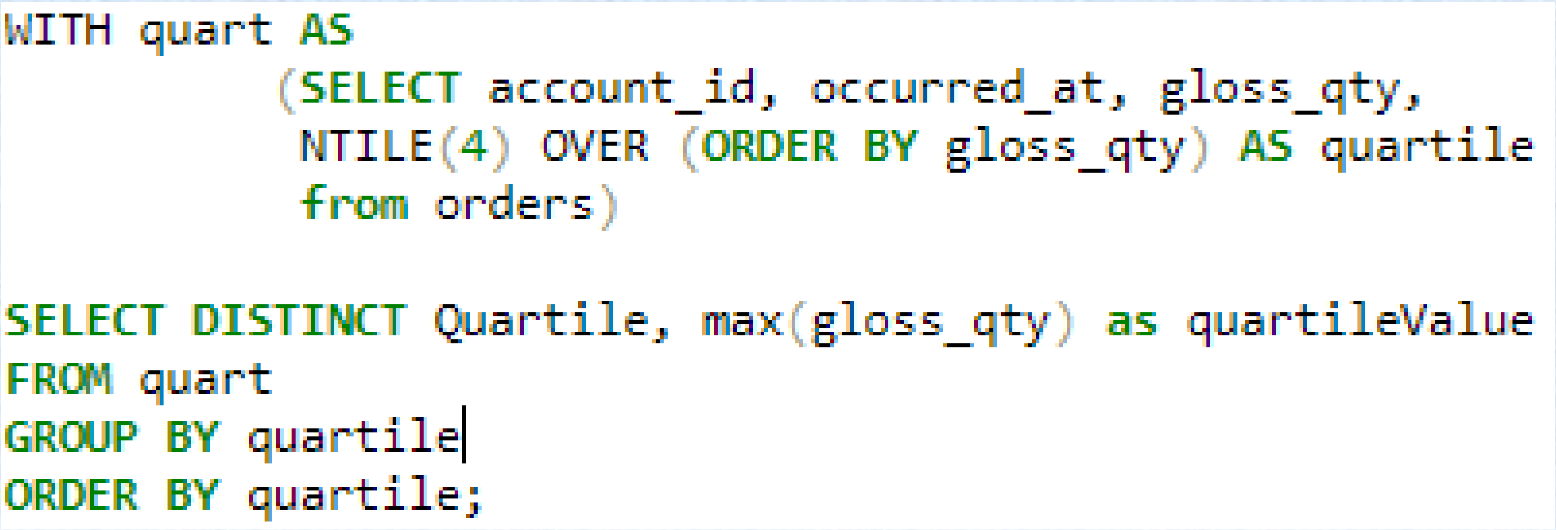


Output 6912 results

account_id	occurred_at	standard_qty	standard_quartile	standard_quintile	standard
4501	2016-07-29T19:58:32.000Z	5	1	1	1
4501	2016-05-30T04:18:34.000Z	15	1	2	1
4501	2016-06-29T04:03:39.000Z	11	1	1	1
4501	2016-11-22T06:57:04.000Z	6	1	1	1
4501	2016-12-21T13:30:42.000Z	61	2	2	1
4501	2016-11-22T06:52:22.000Z	63	2	3	1
4501	2016-08-27T00:58:11.000Z	16	2	2	1
4501	2016-06-29T03:57:11.000Z	104	3	3	2
4501	2016-12-21T13:43:26.000Z	126	3	4	2
4501	2016-07-29T20:06:39.000Z	111	3	3	2
4501	2016-10-24T08:50:37.000Z	159	4	5	2
4501	2016-08-27T00:48:17.000Z	180	4	5	2
4501	2016-09-25T01:44:03.000Z	158	4	4	2
4491	2013-12-08T06:34:23.000Z	43	1	2	1
4491	2015-02-22T07:24:04.000Z	0	1	1	1
4491	2014-01-06T08:11:00.000Z	0	1	1	1
4491	2014-07-31T05:05:06.000Z	12	1	1	1
4491	2014-08-29T17:15:24.000Z	24	1	1	1
4491	2014-05-05T00:03:19.000Z	33	1	1	1
4491	2014-02-04T03:04:08.000Z	34	1	1	1

More on the NTILE function

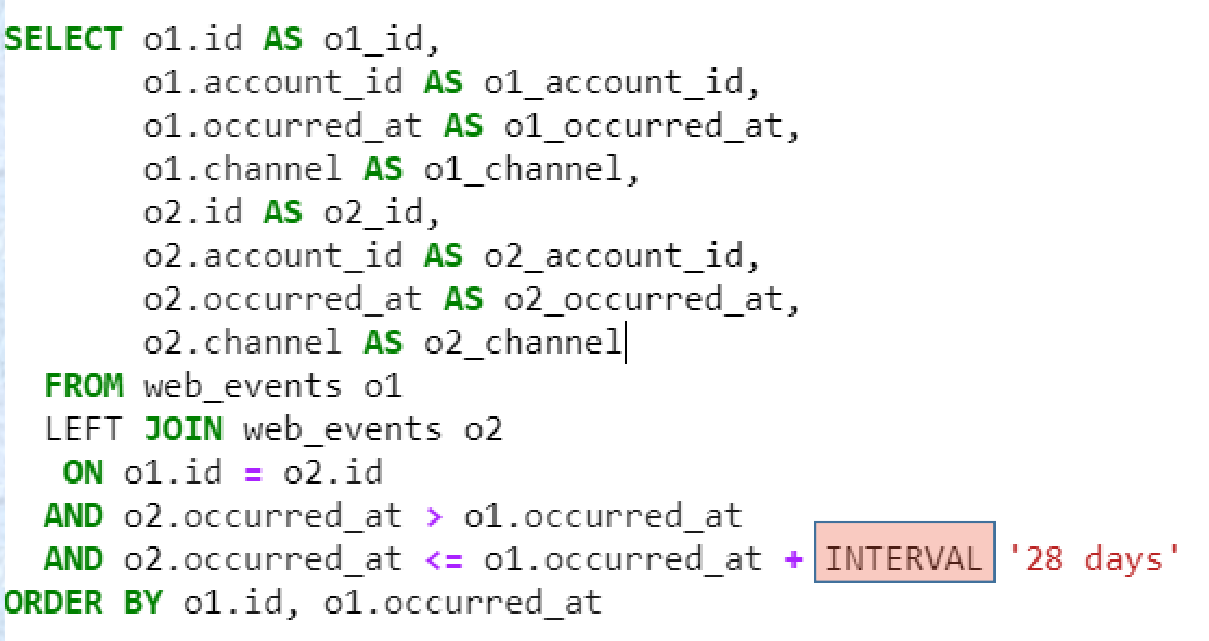
- What NTILE produces is a new variable that indicates which NTILE any observation belongs to.
- **Question: How to get the exact NTILE values?**



```
WITH quart AS
          (SELECT account_id, occurred_at, gloss_qty,
           NTILE(4) OVER (ORDER BY gloss_qty) AS quartile
           from orders)

SELECT DISTINCT Quartile, max(gloss_qty) as quartileValue
FROM quart
GROUP BY quartile
ORDER BY quartile;
```

Self JOINs

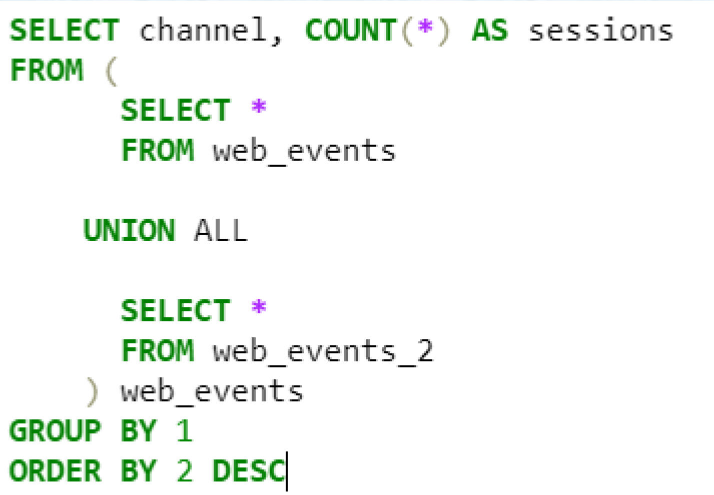
- One of the most common use cases for self JOINs is in cases where two events occurred, one after another.
    - Find out orders that come **within 28 days**
    - Use **alias** to distinguish



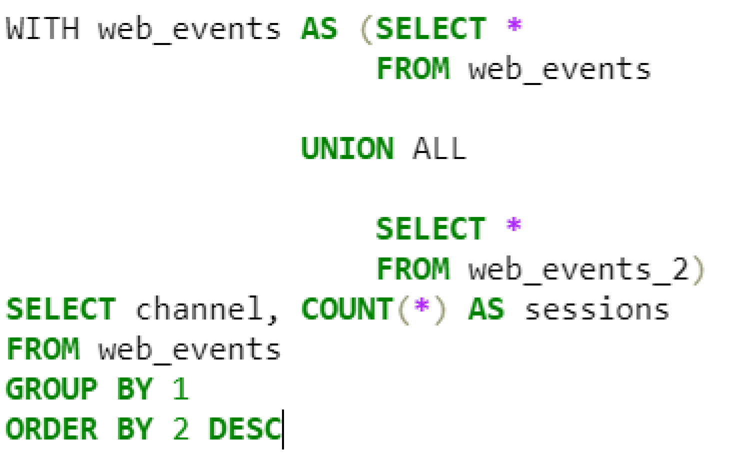
```
SELECT o1.id AS o1_id,
       o1.account_id AS o1_account_id,
       o1.occurred_at AS o1_occurred_at,
       o1.channel AS o1_channel,
       o2.id AS o2_id,
       o2.account_id AS o2_account_id,
       o2.occurred_at AS o2_occurred_at,
       o2.channel AS o2_channel
  FROM web_events o1
  LEFT JOIN web_events o2
   ON o1.id = o2.id
  AND o2.occurred_at > o1.occurred_at
  AND o2.occurred_at <= o1.occurred_at + INTERVAL '28 days'
ORDER BY o1.id, o1.occurred_at
```

Appending Data via UNION

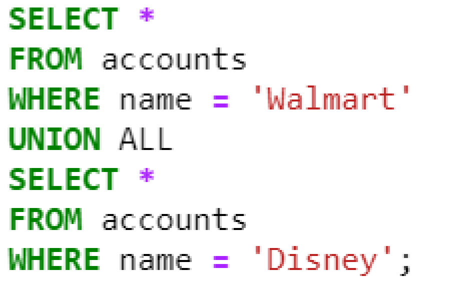
- **UNION Use Case**
  - To combine the result sets of 2 or more SELECT statements.
  - **“UNION”** removes duplicate rows between the various SELECT statements.
  - **“UNION ALL”** keep all
- **SQL's two strict rules for appending data**
  - There must be the **same number** of columns in both SELECT statements.
  - Those columns must have the **same data types** in the same order as the first table
- **Expert Tip**
  - UNION removes duplicate rows.
  - UNION ALL does not remove duplicate rows.



```
SELECT channel, COUNT(*) AS sessions
FROM (
      SELECT *
      FROM web_events

    UNION ALL

      SELECT *
      FROM web_events_2
    ) web_events
GROUP BY 1
ORDER BY 2 DESC
```

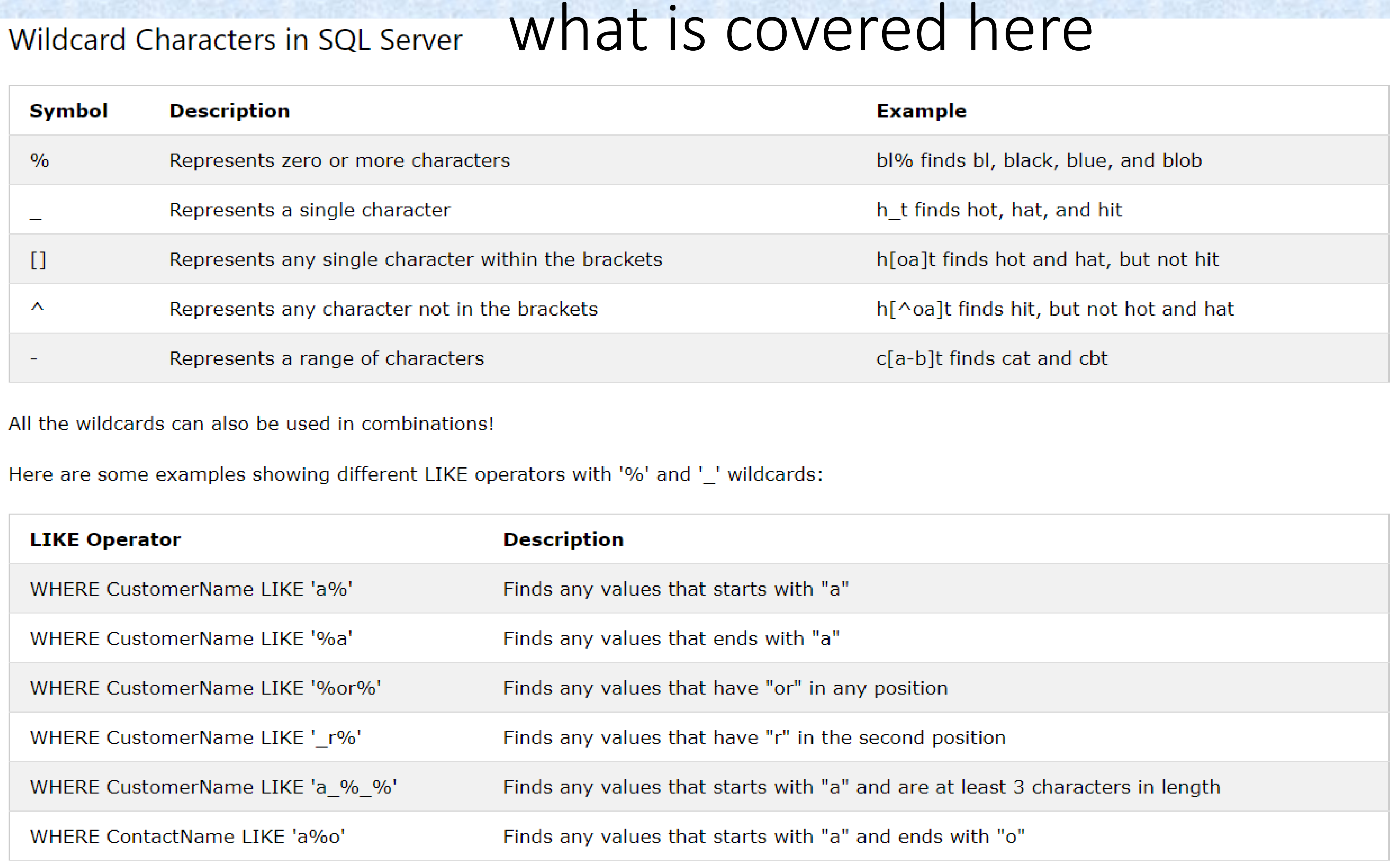


```
WITH web_events AS (SELECT *
                    FROM web_events

                UNION ALL

                    SELECT *
                    FROM web_events_2)
SELECT channel, COUNT(*) AS sessions
FROM web_events
GROUP BY 1
ORDER BY 2 DESC
```



```
SELECT *
FROM accounts
WHERE name = 'Walmart'
UNION ALL
SELECT *
FROM accounts
WHERE name = 'Disney';
```

SQL is much more powerful than what is covered here



Wildcard Characters in SQL Server

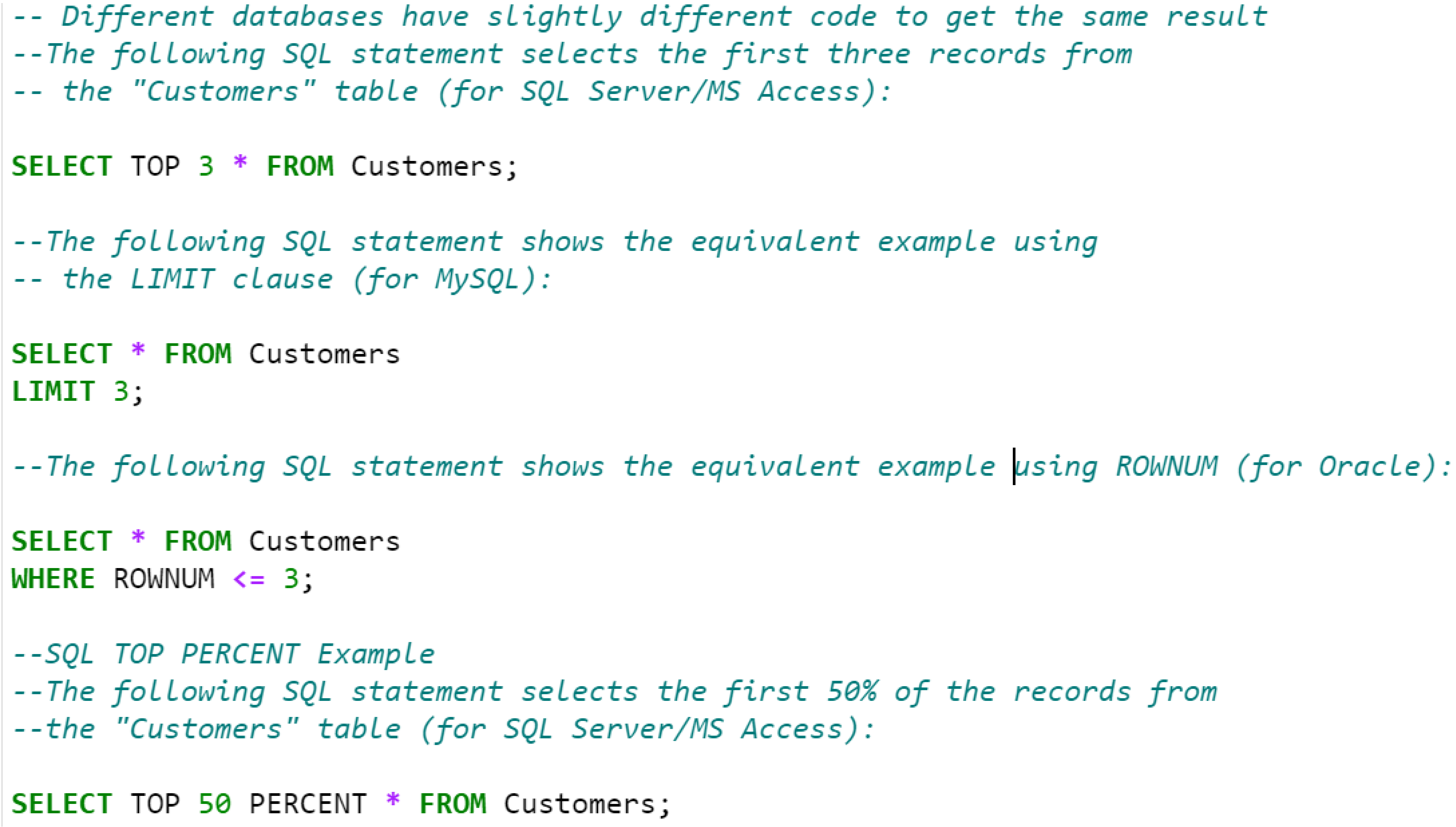
Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

All the wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

Some SQL syntax varies slightly across databases



```
-- Different databases have slightly different code to get the same result
--The following SQL statement selects the first three records from
-- the "Customers" table (for SQL Server/MS Access):

SELECT TOP 3 * FROM Customers;

--The following SQL statement shows the equivalent example using
-- the LIMIT clause (for MySQL):

SELECT * FROM Customers
LIMIT 3;

--The following SQL statement shows the equivalent example using ROWNUM (for Oracle):

SELECT * FROM Customers
WHERE ROWNUM <= 3;

--SQL TOP PERCENT Example
--The following SQL statement selects the first 50% of the records from
--the "Customers" table (for SQL Server/MS Access):

SELECT TOP 50 PERCENT * FROM Customers;
```

SQL and MySQL keywords and functions

Functions	Website
SQL Keywords Reference	https://www.w3schools.com/sql/sql_ref_keywords.asp
MySQL Functions	https://www.w3schools.com/sql/sql_ref_mysql.asp
SQL Server Functions	https://www.w3schools.com/sql/sql_ref_sqlserver.asp
MS Access Functions	https://www.w3schools.com/sql/sql_ref_msaccess.asp

If you know some better web sources for learning/teaching SQL, please share with me so I can use it for future teaching.
Much appreciate it ☺, and thank you!