Welcome!

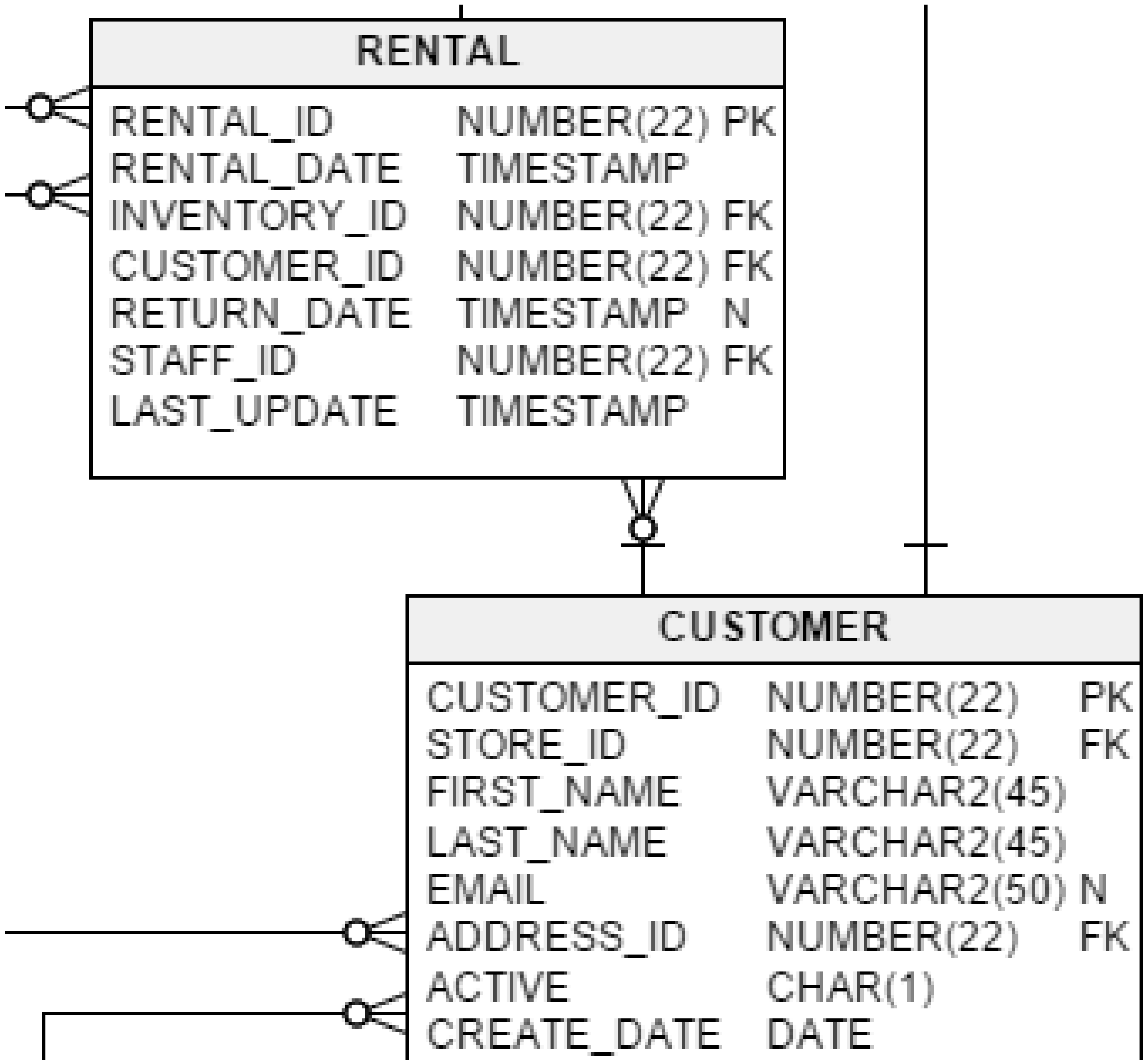
FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo

Sr. Director, Digital Strategy

The Sakila Database



RENTAL		
RENTAL_ID	NUMBER(22)	PK
RENTAL_DATE	TIMESTAMP	
INVENTORY_ID	NUMBER(22)	FK
CUSTOMER_ID	NUMBER(22)	FK
RETURN_DATE	TIMESTAMP	N
STAFF_ID	NUMBER(22)	FK
LAST_UPDATE	TIMESTAMP	

CUSTOMER		
CUSTOMER_ID	NUMBER(22)	PK
STORE_ID	NUMBER(22)	FK
FIRST_NAME	VARCHAR2(45)	
LAST_NAME	VARCHAR2(45)	
EMAIL	VARCHAR2(50)	N
ADDRESS_ID	NUMBER(22)	FK
ACTIVE	CHAR(1)	
CREATE_DATE	DATE	

- Highly normalized
- Representative data types
- Custom functions

Topics

- Common data types in PostgreSQL
- Date and time functions and operators
- Parsing and manipulating text
- Full-text search and PostgreSQL Extensions

Common data types

- Text data types
  - `CHAR` , `VARCHAR` and `TEXT`
- Numeric data types
  - `INT` and `DECIMAL`
- Date / time data types
  - `DATE` , `TIME` , `TIMESTAMP` , `INTERVAL`
- Arrays

Text data types

```sql
SELECT title
FROM film
LIMIT 5
```

```
+-------------------+
| title             |
|-------------------|
| ACADEMY DINOSAUR  |
| ACE GOLDFINGER    |
| ADAPTATION HOLES  |
| AFFAIR PREJUDICE  |
| AFRICAN EGG       |
+-------------------+
```

```sql
SELECT description
FROM film
LIMIT 2
```

```
+-----------------------------------------+
| description                             |
|-----------------------------------------|
| A Epic Drama of a Feminist And a Mad    |
|  Scientist who must Battle a Teacher in |
|  The Canadian Rockies.                  |
| A Astounding Epistle of a Database      |
|  Administrator And a Explorer who       |
|  must Find a Car in Ancient China       |
+-----------------------------------------+
```

Numeric data types

```sql
SELECT
    payment_id
FROM payment
LIMIT 5
```

```sql
SELECT
    amount
FROM payment
LIMIT 5
```

```
+-------------+
| payment_id  |
|-------------|
| 1           |
| 2           |
| 3           |
| 4           |
| 5           |
+-------------+
```

```
+--------+
| amount |
|--------|
| 2.99   |
| 0.99   |
| 5.99   |
| 0.99   |
| 9.99   |
+--------+
```

Determining data types from existing tables

```sql
SELECT
    title,
    description,
    special_features
FROM FILM
LIMIT 5
```

```
+---------------+------------------+------------------------------+
| title         |  description     | special_features             |
|---------------|------------------|------------------------------|
| ACADEMY D...  | A Epic...        | {Deleted Scenes,Behi...}     |
| ACE GOLD...   | A Astound..      | {Trailers,Deleted Scenes}    |
| AFFAIR PR...  | A Fanciful,..    | {Commentaries,Behind the...} |
+---------------+------------------+------------------------------+
```

Determining data types from existing tables

```sql
SELECT
    column_name,
    data_type
FROM INFORMATION_SCHEMA.COLUMNS
WHERE column_name in ('title','description','special_features')
  AND table_name ='film';
```

```
+------------------+-------------------+
| column_name      | data_type         |
|------------------|-------------------|
| title            | character varying |
| description      | text              |
| special_features | ARRAY             |
+------------------+-------------------+
```

Let's practice!

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

Date and time data types

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo

Sr. Director, Digital Strategy

TIMESTAMP data types

- ISO 8601 format: yyyy-mm-dd

```
+------------------------------+
| timestamp                    |
|------------------------------|
| 2019-03-26 01:05:17.93027+00 |
+------------------------------+
```

```sql
SELECT payment_date
FROM payment;
```

```
+---------------------+
| payment_date        |
|---------------------|
| 2005-05-25 11:30:37 |
+---------------------+
```

DATE and TIME data types

```
+------------+-------------------+
| date       | time              |
|------------|-------------------|
| 2005-05-28 | 01:05:17.93027+00 |
+------------+-------------------+
```

```sql
SELECT create_date
FROM customer
```

```
+-------------+
| create_date |
|-------------|
| 2006-02-14  |
+-------------+
```

INTERVAL data types

```
+-----------+
| interval  |
|-----------|
| 4 days    |
+-----------+
```

```sql
SELECT rental_date + INTERVAL '3 days' as expected_return
FROM rental;
```

```
+---------------------+
| expected_return     |
|---------------------|
| 2005-05-27 22:53:30 |
+---------------------+
```

Looking at date and time types

```
SELECT
    column_name,
    data_type
FROM INFORMATION_SCHEMA.COLUMNS
WHERE column_name in ('rental_date')
  AND table_name ='rental';
```

```
+-------------+-----------------------------+
| column_name | data_type                   |
|-------------|-----------------------------|
| rental_date | timestamp without time zone |
+-------------+-----------------------------+
```

Let's practice!

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

Working with ARRAYs

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo
Sr. Director, Digital Strategy

Before we get started

CREATE TABLE example

```
CREATE TABLE my_first_table (
    first_column text,
    second_column integer
);
```

INSERT example

```
INSERT INTO my_first_table
    (first_column, second_column) VALUES ('text value', 12);
```

ARRAY a special type

Let's create a simple table with two array columns.

```
CREATE TABLE grades (
    student_id int,
    email text[][],
    test_scores int[]
);
```

INSERT statements with ARRAYS

Example INSERT statement:

```
INSERT INTO grades
    VALUES (1,
    '{{"work","work1@datacamp.com"},{"other","other1@datacamp.com"}}',
    '{92,85,96,88}' );
```

Accessing ARRAYs

```
SELECT
   email[1][1] AS type,
   email[1][2] AS address,
   test_scores[1],
FROM grades;
```

```
+--------+--------------------+-------------+
| type   |  address           | test_scores |
|--------|--------------------|-------------|
| work   | work1@datacamp.com | 92          |
| work   | work2@datacamp.com | 76          |
+--------+--------------------+-------------+
```

Note that PostgreSQL array indexes start with one and not zero.

Searching ARRAYs

```
SELECT
   email[1][1] as type,
   email[1][2] as address,
   test_scores[1]
FROM grades
WHERE email[1][1] = 'work';
```

```
+--------+--------------------+-------------+
| type   |  address           | test_scores |
|--------|--------------------|-------------|
| work   | work1@datacamp.com | 92          |
| work   | work2@datacamp.com | 76          |
+--------+--------------------+-------------+
```

ARRAY functions and operators

```
SELECT
   email[2][1] as type,
   email[2][2] as address,
   test_scores[1]
FROM grades
WHERE 'other' = ANY (email);
```

```
+---------+---------------------+-------------+
| type    |  address            | test_scores |
|---------|-----------------------------------|
| other   | other1@datacamp.com | 92          |
| null    | null                | 76          |
+---------+---------------------+-------------+
```

ARRAY functions and operators

```sql
SELECT
   email[2][1] as type,
   email[2][2] as address,
   test_scores[1]
FROM grades
WHERE email @> ARRAY['other'];
```

```
+---------+---------------------+-------------+
| type    |  address            | test_scores |
|---------|-----------------------------------|
| other   | other1@datacamp.com | 92          |
| null    | null                | 76          |
+---------+---------------------+-------------+
```

Let's practice!

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

Overview of basic arithmetic operators

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo
Sr. Director, Digital Strategy

Topics

- Overview of basic arithmetic operators
- The `CURRENT_DATE` , `CURRENT_TIMESTAMP` , `NOW()` functions
- The `AGE()` function
- The `EXTRACT()` , `DATE_PART()` , and `DATE_TRUNC()` functions

Adding and subtracting date / time data

```
SELECT date '2005-09-11' - date '2005-09-10';
```

```
+----------+
| integer |
|---------|
| 1       |
+---------+
```

Adding and subtracting date / time data

```sql
SELECT date '2005-09-11' + integer '3';
```

```
+------------+
| date       |
|------------|
| 2005-09-14 |
+------------+
```

Adding and subtracting date / time data

```
SELECT date '2005-09-11 00:00:00' - date '2005-09-09 12:00:00';
```

```
+-----------------+
| interval        |
|-----------------|
| 1 day 12:00:00  |
+-----------------+
```

Calculating time periods with AGE

```sql
SELECT AGE(timestamp '2005-09-11 00:00:00', timestamp '2005-09-09 12:00:00');
```

```
+-----------------+
| interval        |
|-----------------|
| 1 day 12:00:00  |
+-----------------+
```

DVDs, really??

```
SELECT
    AGE(rental_date)
FROM rental;
```

```
+-----------------------------------+
| age                               |
|-----------------------------------|
| 13 years 11 mons 12 days 01:06:30 |
| 13 years 11 mons 12 days 01:05:27 |
| 13 years 11 mons 12 days 00:56:21 |
+-----------------------------------+
```

Date / time arithmetic using INTERVALs

```sql
SELECT rental_date + INTERVAL '3 days' as expected_return
FROM rental;
```

```
+---------------------+
| expected_return     |
|---------------------|
| 2005-05-27 22:53:30 |
+---------------------+
```

Date / time arithmetic using INTERVALs

```
SELECT timestamp '2019-05-01' + 21 * INTERVAL '1 day';
```

```
+-----------------------------+
| timestamp without timezone |
|-----------------------------|
| 2019-05-22 00:00:00        |
+-----------------------------+
```

Let's practice!

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

Functions for retrieving current date/time

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo
Sr. Director, Digital Strategy

Retrieving the current timestamp

```
SELECT NOW();
```

```
+-------------------------------+
| now()                         |
|-------------------------------|
| 2019-04-19 02:51:18.448641+00 |
+-------------------------------+
```

Retrieving the current timestamp

```
SELECT NOW()::timestamp;
```

```
+----------------------------+
| now()                      |
|----------------------------|
| 2019-04-19 02:51:18.448641 |
+----------------------------+
```

Retrieving the current timestamp

PostgreSQL specific casting

```
SELECT NOW()::timestamp;
```

CAST() function

```
SELECT CAST(NOW() as timestamp);
```

Retrieving the current timestamp

```sql
SELECT CURRENT_TIMESTAMP;
```

```
+-------------------------------+
| current_timestamp             |
|-------------------------------|
| 2019-04-19 02:51:18.448641+00 |
+-------------------------------+
```

Retrieving the current timestamp

```
SELECT CURRENT_TIMESTAMP(2);
```

```
+-------------------------------+
| current_timestamp             |
|-------------------------------|
| 2019-04-19 02:51:18.44+00     |
+-------------------------------+
```

Current date and time

```
SELECT CURRENT_DATE;
```

```
+--------------+
| current_date |
|--------------|
| 2019-04-19   |
+--------------+
```

Current date and time

```
SELECT CURRENT_TIME;
```

```
+------------------------+
| current_time           |
|------------------------|
| 04:06:30.929845+00:00  |
+------------------------+
```

Let's practice!

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

Extracting and transforming date / time data

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo
Sr. Director, Digital Strategy

Extracting and transforming date and time data

Exploring the `EXTRACT()` , `DATE_PART()` and `DATE_TRUNC()` functions

- Transactional timestamp precision not useful for analysis

```
2005-05-13 08:53:53
```

- Often need to extract parts of timestamps

```
2005 or 5 or 2 or Friday
```

- Or convert / truncate timestamp precision to standardize

```
2005-05-13 00:00:00
```

Extracting and transforming date / time data

- EXTRACT(**field** *FROM* **source**)

```
SELECT EXTRACT(quarter FROM timestamp '2005-01-24 05:12:00') AS quarter;
```

- DATE_PART(**'field'**, **source**)

```
SELECT DATE_PART('quarter', timestamp '2005-01-24 05:12:00') AS quarter;
```

```
+---------+
| quarter |
|---------|
| 1       |
+---------+
```

Extracting sub-fields from timestamp data

Transactional data from DVD Rentals *payment* table

```
SELECT * FROM payment;
```

```
+--------------------------------------------------------------------------------+
| payment_id | customer_id | staff_id | rental_id | amount | payment_date        |
|------------|-------------|----------|-----------|--------|---------------------|
| 1          | 1           | 1        | 76        | 2.99   | 2005-05-25 11:30:37 |
| 2          | 1           | 1        | 573       | 0.99   | 2005-05-28 10:35:23 |
| 3          | 1           | 1        | 1185      | 5.99   | 2005-06-15 0:54:12  |
+--------------------------------------------------------------------------------+
```

Extracting sub-fields from timestamp data

Data from *payment* table by year and quarter

```
SELECT
  EXTRACT(quarter FROM payment_date) AS quarter,
  EXTRACT(year FROM payment_date) AS year,
  SUM(amount) AS total_payments
FROM
  payment
GROUP BY 1, 2;
```

Results

```
+----------------------------------+
| quarter | year | total_payments |
|---------|------|----------------|
| 2       | 2005 | 14456.31       |
| 3       | 2005 | 52446.02       |
| 1       | 2006 | 514.18         |
+----------------------------------+
```

Truncating timestamps using DATE_TRUNC()

The `DATE_TRUNC()` function will truncate timestamp or interval data types.

- Truncate timestamp '2005-05-21 15:30:30' by year

```
SELECT DATE_TRUNC('year', TIMESTAMP '2005-05-21 15:30:30');
```

```
Result: 2005-01-01 00:00:00
```

- Truncate timestamp '2005-05-21 15:30:30' by month

```
SELECT DATE_TRUNC('month', TIMESTAMP '2005-05-21 15:30:30');
```

```
Result: 2005-05-01 00:00:00
```

Let's practice!

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

Reformatting string and character data

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo
Sr. Director, Digital Strategy

Topics

- Reformatting string and character data.
- Parsing string and character data.
- Determine string length and character position.
- Truncating and padding string data.

The string concatenation operator

```sql
SELECT
  first_name,
  last_name,
  first_name || ' ' || last_name AS full_name
FROM customer
```

```
+------------+-----------+-------------------+
| first_name | last_name | full_name         |
|------------|-----------|-------------------|
| MARY       | SMITH     | MARY SMITH        |
| LINDA      | WILLIAMS  | LINDA WILLIAMS    |
+------------+-----------+-------------------+
```

String concatenation with functions

```sql
SELECT
  CONCAT(first_name,' ', last_name) AS full_name
FROM customer;
```

```
+--------------------------------------------+
| first_name | last_name | full_name         |
|--------------------------------------------|
| MARY       | SMITH     | MARY SMITH        |
| LINDA      | WILLIAMS  | LINDA WILLIAMS    |
+--------------------------------------------+
```

String concatenation with a non-string input

```
SELECT
  customer_id || ': '
  || first_name || ' '
  || last_name AS full_name
FROM customer;
```

```
+-------------------+
| full_name         |
|-------------------|
| 1: MARY SMITH     |
| 2: LINDA WILLIAMS |
+-------------------+
```

Changing the case of string

```
SELECT
  UPPER(email)
FROM customer;
```

```
+-------------------------------------+
| UPPER(email)                        |
|-------------------------------------|
| MARY.SMITH@SAKILACUSTOMER.ORG       |
| PATRICIA.JOHNSON@SAKILACUSTOMER.ORG |
| LINDA.WILLIAMS@SAKILACUSTOMER.ORG   |
+-------------------------------------+
```

Changing the case of string

```sql
SELECT
  LOWER(title)
FROM film;
```

```
+-------------------+
| LOWER(title)      |
|-------------------|
| academy dinosaur  |
| ace goldfinger    |
| adaptation holes  |
+-------------------+
```

Changing the case of string

```
SELECT
  INITCAP(title)
FROM film;
```

```
+-------------------+
| INITCAP(title)    |
|-------------------|
| Academy Dinosaur  |
| Ace Goldfinger    |
| Adaptation Holes  |
+-------------------+
```

Replacing characters in a string

```sql
SELECT description FROM film;
```

```
+---------------------------------------------------------+
| description                                             |
|---------------------------------------------------------|
| A Epic Drama of a Feminist And a Mad Scientist...       |
| A Astounding Epistle of a Database Administrator...     |
| A Astounding Reflection of a Lumberjack And a Car...    |
| A Fanciful Documentary of a Frisbee And a Lumberjack... |
| A Fast-Paced Documentary of a Pastry Chef And a...      |
+---------------------------------------------------------+
```

Replacing characters in a string

```
SELECT
    REPLACE(description, 'A Astounding',
              'An Astounding') as description
FROM film;
```

```
+---------------------------------------------------------+
| description                                             |
|---------------------------------------------------------|
| A Epic Drama of a Feminist And a Mad Scientist...       |
| An Astounding Epistle of a Database Administrator...    |
| An Astounding Reflection of a Lumberjack And a Car...   |
+---------------------------------------------------------+
```

Manipulating string data with REVERSE

```
SELECT
  title,
  REVERSE(title)
FROM
  film AS f;
```

```
+-------------------------------------+
| title            | reverse(title)   |
|-------------------------------------|
| ACADEMY DINOSAUR | RUASONID YMEDACA |
| ACE GOLDFINGER   | REGNIFDLOG ECA   |
+-------------------------------------+
```

Let's practice!

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

Parsing string and character data

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo
Sr. Director, Digital Strategy

Determining the length of a string

```
SELECT
    title,
    CHAR_LENGTH(title)
FROM film;
```

```
+-------------------+---------------------+
| title             | CHAR_LENGTH(title)  |
|-------------------+---------------------|
| ACADEMY DINOSAUR  | 16                  |
| ACE GOLDFINGER    | 14                  |
| ADAPTATION HOLES  | 16                  |
+-------------------+---------------------+
```

Determining the length of a string

```
SELECT
    title,
    LENGTH(title)
FROM film;
```

```
+-------------------+----------------+
| title             | LENGTH(title)  |
|-------------------+----------------|
| ACADEMY DINOSAUR  | 16             |
| ACE GOLDFINGER    | 14             |
| ADAPTATION HOLES  | 16             |
+-------------------+----------------+
```

Finding the position of a character in a string

```
SELECT
    email,
    POSITION('@' IN email)
FROM customer;
```

```
+-------------------------------------+------------------------+
| email                               | POSITION('@' IN email) |
|-------------------------------------|------------------------|
| MARY.SMITH@sakilacustomer.org       | 11                     |
| PATRICIA.JOHNSON@sakilacustomer.org | 17                     |
| LINDA.WILLIAMS@sakilacustomer.org   | 15                     |
+-------------------------------------+------------------------+
```

Finding the position of a character in a string

```sql
SELECT
    email,
    STRPOS(email, '@')
FROM customer;
```

```
+--------------------------------------+---------------------+
| email                                | STRPOS(email, '@') |
|--------------------------------------|---------------------|
| MARY.SMITH@sakilacustomer.org        | 11                  |
| PATRICIA.JOHNSON@sakilacustomer.org | 17                  |
| LINDA.WILLIAMS@sakilacustomer.org   | 15                  |
+--------------------------------------+---------------------+
```

Parsing string data

```
SELECT
    LEFT(description, 50)
FROM film;
```

```
+----------------------------------------------------+
| description                                        |
|----------------------------------------------------|
| A Epic Drama of a Feminist And a Mad Scientist who |
| A Astounding Epistle of a Database Administrator A |
| A Astounding Reflection of a Lumberjack And a Car  |
+----------------------------------------------------+
```

Parsing string data

```
SELECT
    RIGHT(description, 50)
FROM film;
```

```
+----------------------------------------------------+
| description                                        |
|----------------------------------------------------|
|  who must Battle a Teacher in The Canadian Rockies |
| nd a Explorer who must Find a Car in Ancient China |
| Car who must Sink a Lumberjack in A Baloon Factory |
+----------------------------------------------------+
```

Extracting substrings of character data

```
SELECT
    SUBSTRING(description, 10, 50)
FROM
  film AS f;
```

```
+----------------------------------------------------+
| description                                        |
|----------------------------------------------------|
| ama of a Feminist And a Mad Scientist who must Bat |
| ing Epistle of a Database Administrator And a Expl |
| ing Reflection of a Lumberjack And a Car who must  |
+----------------------------------------------------+
```

Extracting substrings of character data

```
SELECT
    SUBSTRING(email FROM 0 FOR POSITION('@' IN email))
FROM
  customer;
```

```
+----------------------------------------------------+
| SUBSTRING(email FROM 0 FOR POSITION('@' IN email)) |
|----------------------------------------------------|
| MARY.SMITH                                         |
| PATRICIA.JOHNSON                                   |
| LINDA.WILLIAMS                                     |
+----------------------------------------------------+
```

Extracting substrings of character data

```
SELECT
    SUBSTRING(email FROM POSITION('@' IN email)+1 FOR CHAR_LENGTH(email))
FROM
  customer;
```

```
+-----------------------------------------------------------------------+
| SUBSTRING(email FROM POSITION('@' IN email)+1 FOR CHAR_LENGTH(email)) |
|-----------------------------------------------------------------------|
| sakilacustomer.org                                                    |
| sakilacustomer.org                                                    |
| sakilacustomer.org                                                    |
+-----------------------------------------------------------------------+
```

Extracting substrings of character data

```
SELECT
    SUBSTR(description, 10, 50)
FROM
  film AS f;
```

```
+----------------------------------------------------+
| description                                        |
|----------------------------------------------------|
| ama of a Feminist And a Mad Scientist who must Bat |
| ing Epistle of a Database Administrator And a Expl |
| ing Reflection of a Lumberjack And a Car who must  |
+----------------------------------------------------+
```

Let's practice!

Truncating and padding string data

FUNCTIONS FOR MANIPULATING DATA IN POSTGRESQL

SQL

Brian Piccolo

Sr. Director, Digital Strategy

Removing whitespace from strings

```
TRIM([leading | trailing | both] [characters] from string)
```

- **First parameter:** [leading | trailing | both]
- **Second parameter:** [characters]
- **Third parameter:** from string

Removing whitespace from strings

```sql
SELECT TRIM('  padded  ');
```

```
+--------+
| TRIM   |
|--------|
| padded |
+--------+
```

Removing whitespace from strings

```
SELECT LTRIM('  padded   ');
```

```
+------------+
| LTRIM      |
|------------|
| padded     |
+------------+
```

Removing whitespace from strings

```sql
SELECT RTRIM('  padded   ');
```

```
+----------+
| RTRIM    |
|----------|
|   padded |
+----------+
```

Padding strings with character data

```sql
SELECT LPAD('padded', 10, '#');
```

```
+-------------+
| LPAD        |
|-------------|
| ####padded  |
+-------------+
```

Padding strings with whitespace

```sql
SELECT LPAD('padded', 10);
```

```
+-------------+
| LPAD        |
|-------------|
|     padded  |
+-------------+
```

```sql
SELECT LPAD('padded', 5);
```

```
+-------------+
| LPAD        |
|-------------|
|     padde   |
+-------------+
```

Padding strings with whitespace

```sql
SELECT RPAD('padded', 10, '#');
```

```
+-------------+
| RPAD        |
|-------------|
| padded####  |
+-------------+
```

Let's practice!

Pivoting

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS

SQL

Michel Semaan
Data Scientist

Transforming tables

Before

```
| Country | Year | Awards |
|---------|------|--------|
| CHN     | 2008 | 74     |
| CHN     | 2012 | 56     |
| RUS     | 2008 | 43     |
| RUS     | 2012 | 47     |
| USA     | 2008 | 125    |
| USA     | 2012 | 147    |
```

- Gold medals awarded to China, Russia, and the USA

After

```
| Country | 2008 | 2012 |
|---------|------|------|
| CHN     | 74   | 56   |
| RUS     | 43   | 47   |
| USA     | 125  | 147  |
```

- Pivoted by `Year`
- Easier to scan, especially if pivoted by a chronologically ordered column

Enter CROSSTAB

```
CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT * FROM CROSSTAB($$
  source_sql TEXT
$$) AS ct (column_1 DATA_TYPE_1,
           column_2 DATA_TYPE_2,
           ...,
           column_n DATA_TYPE_N);
```

Queries

Before

```
SELECT
  Country, Year, COUNT(*) AS Awards
FROM Summer_Medals
WHERE
  Country IN ('CHN', 'RUS', 'USA')
  AND Year IN (2008, 2012)
  AND Medal = 'Gold'
GROUP BY Country, Year
ORDER BY Country ASC, Year ASC;
```

After

```
CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT * FROM CROSSTAB($$
  SELECT
    Country, Year, COUNT(*) :: INTEGER AS Awards
  FROM Summer_Medals
  WHERE
    Country IN ('CHN', 'RUS', 'USA')
    AND Year IN (2008, 2012)
    AND Medal = 'Gold'
  GROUP BY Country, Year
  ORDER BY Country ASC, Year ASC;
$$) AS ct (Country VARCHAR, "2008" INTEGER, "2012" INTEGER)

ORDER BY Country ASC;
```

Source query

```
WITH Country_Awards AS (
  SELECT
    Country, Year, COUNT(*) AS Awards
  FROM Summer_Medals
  WHERE
    Country IN ('CHN', 'RUS', 'USA')
    AND Year IN (2004, 2008, 2012)
    AND Medal = 'Gold' AND Sport = 'Gymnastics'
  GROUP BY Country, Year
  ORDER BY Country ASC, Year ASC)

SELECT
  Country, Year,
  RANK() OVER
    (PARTITION BY Year ORDER BY Awards DESC) :: INTEGER
    AS rank
FROM Country_Awards
ORDER BY Country ASC, Year ASC;
```

Source result

```
| Country | Year |Rank |
|---------|------|-----|
| CHN     | 2004 | 3   |
| CHN     | 2008 | 1   |
| CHN     | 2012 | 1   |
| RUS     | 2004 | 1   |
| RUS     | 2008 | 2   |
| RUS     | 2012 | 2   |
| USA     | 2004 | 2   |
| USA     | 2008 | 3   |
| USA     | 2012 | 3   |
```

Pivot query

```
CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT * FROM CROSSTAB($$
  ...
$$) AS ct (Country VARCHAR,
           "2004" INTEGER,
           "2008" INTEGER,
           "2012" INTEGER)

ORDER BY Country ASC;
```

Pivot result

```
| Country | 2004 | 2008 | 2012 |
|---------|------|------|------|
| CHN     | 3    | 1    | 1    |
| RUS     | 1    | 2    | 2    |
| USA     | 2    | 3    | 3    |
```

Let's practice!

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS

ROLLUP and CUBE

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS

SQL

Michel Semaan
Data Scientist

Group-level totals

Chinese and Russian medals in the 2008 Summer Olympics per medal class

```
| Country | Medal  | Awards |
|---------|--------|--------|
| CHN     | Bronze | 57     |
| CHN     | Gold   | 74     |
| CHN     | Silver | 53     |
| CHN     | Total  | 184    |
| RUS     | Bronze | 56     |
| RUS     | Gold   | 43     |
| RUS     | Silver | 44     |
| RUS     | Total  | 143    |
```

The old way

```sql
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM Summer_Medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY Country, Medal
ORDER BY Country ASC, Medal ASC
UNION ALL

SELECT
  Country, 'Total', COUNT(*) AS Awards
FROM Summer_Medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY Country, 2
ORDER BY Country ASC;
```

Enter ROLLUP

```
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM Summer_Medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY Country, ROLLUP(Medal)
ORDER BY Country ASC, Medal ASC;
```

- `ROLLUP` is a `GROUP BY` subclause that includes extra rows for group-level aggregations
- `GROUP BY Country, ROLLUP(Medal)` will count all `Country` - and `Medal` -level totals, then count only `Country` -level totals and fill in `Medal` with `null` s for these rows

ROLLUP - Query

```
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM summer_medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY ROLLUP(Country, Medal)
ORDER BY Country ASC, Medal ASC;
```

- `ROLLUP` is hierarchical, de-aggregating from the leftmost provided column to the right-most
  - `ROLLUP(Country, Medal)` includes `Country` -level totals
  - `ROLLUP(Medal, Country)` includes `Medal` -level totals
  - Both include grand totals

ROLLUP - Result

```
| Country | Medal  | Awards |
|---------|--------|--------|
| CHN     | Bronze | 57     |
| CHN     | Gold   | 74     |
| CHN     | Silver | 53     |
| CHN     | null   | 184    |
| RUS     | Bronze | 56     |
| RUS     | Gold   | 43     |
| RUS     | Silver | 44     |
| RUS     | null   | 143    |
| null    | null   | 327    |
```

- Group-level totals contain `nulls` ; the row with all `null` s is the grand total
- Notice that it didn't include `Medal` -level totals, since it's `ROLLUP(Country, Medal)` and not `ROLLUP(Medal, Country)`

Enter CUBE

```
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM summer_medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY CUBE(Country, Medal)
ORDER BY Country ASC, Medal ASC;
```

- `CUBE` is a non-hierarchical `ROLLUP`
- It generates all possible group-level aggregations
  - `CUBE(Country, Medal)` counts `Country` -level, `Medal` -level, and grand totals

CUBE - Result

```
| Country | Medal  | Awards |
|---------|--------|--------|
| CHN     | Bronze | 57     |
| CHN     | Gold   | 74     |
| CHN     | Silver | 53     |
| CHN     | null   | 184    |
| RUS     | Bronze | 56     |
| RUS     | Gold   | 43     |
| RUS     | Silver | 44     |
| RUS     | null   | 143    |
| null    | Bronze | 113    |
| null    | Gold   | 117    |
| null    | Silver | 97     |
| null    | null   | 327    |
```

- Notice that `Medal` -level totals are included

ROLLUP vs CUBE

Source

```
| Year | Quarter | Sales |
|------|---------|-------|
| 2008 | Q1      | 12    |
| 2008 | Q2      | 15    |
| 2009 | Q1      | 21    |
| 2009 | Q2      | 27    |
```

- Use `ROLLUP` when you have hierarchical data (e.g., date parts) and don't want all possible group-level aggregations
- Use `CUBE` when you want all possible group-level aggregations

`ROLLUP(Year, Quarter)`

```
| Year | Quarter | Sales |
|------|---------|-------|
| 2008 | null    | 27    |
| 2009 | null    | 48    |
| null | null    | 75    |
```

`CUBE(Year, Quarter)`

Above rows + the following

```
| Year | Quarter | Sales |
|------|---------|-------|
| null | Q1      | 33    |
| null | Q2      | 42    |
```

Let's practice!

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS

A survey of useful functions

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS

SQL

Michel Semaan
Data Scientist

Nulls ahoy

Query

```sql
SELECT
  Country, Medal, COUNT(*) AS Awards
FROM summer_medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY ROLLUP(Country, Medal)
ORDER BY Country ASC, Medal ASC;
```

- `null` s signify group totals

Result

```
| Country | Medal  | Awards |
|---------|--------|--------|
| CHN     | Bronze | 57     |
| CHN     | Gold   | 74     |
| CHN     | Silver | 53     |
| CHN     | null   | 184    |
| RUS     | Bronze | 56     |
| RUS     | Gold   | 43     |
| RUS     | Silver | 44     |
| RUS     | null   | 143    |
| null    | null   | 327    |
```

Enter COALESCE

- `COALESCE()` takes a list of values and returns the first non-`null` value, going from left to right
- `COALESCE(null, null, 1, null, 2) ? 1`
- Useful when using SQL operations that return `null`s
  - `ROLLUP` and `CUBE`
  - Pivoting
  - `LAG` and `LEAD`

Annihilating nulls

Query

```sql
SELECT
  COALESCE(Country, 'Both countries') AS Country,
  COALESCE(Medal, 'All medals') AS Medal,
  COUNT(*) AS Awards
FROM summer_medals
WHERE
  Year = 2008 AND Country IN ('CHN', 'RUS')
GROUP BY ROLLUP(Country, Medal)
ORDER BY Country ASC, Medal ASC;
```

Result

```
| Country        | Medal      | Awards |
|----------------|------------|--------|
| Both countries | All medals | 327    |
| CHN            | All medals | 184    |
| CHN            | Bronze     | 57     |
| CHN            | Gold       | 74     |
| CHN            | Silver     | 53     |
| RUS            | All medals | 143    |
| RUS            | Bronze     | 56     |
| RUS            | Gold       | 43     |
| RUS            | Silver     | 44     |
```

Compressing data

Before

```
| Country | Rank |
|---------|------|
| CHN     | 1    |
| RUS     | 2    |
| USA     | 3    |
```

- `Rank` is redundant because the ranking is implied

After

```
CHN, RUS, USA
```

- Succinct and provides all information needed because the ranking is implied

Enter STRING_AGG

- `STRING_AGG(column, separator)` takes all the values of a column and concatenates them, with `separator` in between each value

`STRING_AGG(Letter, ', ')` transforms this...

```
| Letter |
|--------|
| A      |
| B      |
| C      |
```

...into this

```
A, B, C
```

Query and result

Before

```
WITH Country_Medals AS (
  SELECT
    Country, COUNT(*) AS Medals
  FROM Summer_Medals
  WHERE Year = 2012
    AND Country IN ('CHN', 'RUS', 'USA')
    AND Medal = 'Gold'
    AND Sport = 'Gymnastics'
  GROUP BY Country),

  SELECT
    Country,
    RANK() OVER (ORDER BY Medals DESC) AS Rank
  FROM Country_Medals
  ORDER BY Rank ASC;
```

After

```
WITH Country_Medals AS (...),

  Country_Ranks AS (...)

  SELECT STRING_AGG(Country, ', ')
  FROM Country_Medals;
```

Result

```
CHN, RUS, USA
```

Let's practice!

POSTGRESQL SUMMARY STATS AND WINDOW FUNCTIONS