DB Connect: Deep Dive

Beyond the basics

Tyler Muth, Denis Vergnes

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

splunk> .conf18

DB Connect: Deep Dive Collateral

bit.ly/conf-dbx18

Our Speakers

DENIS VERGNES

Principal Software Engineer, Splunk

TYLER MUTH

Analytics Architect, Splunk

splunk> .conf18

Q&A

- End of session in room
- After session outside of room
- DBX Office Hours at the “Foundations & Platform Booth” #12 (dates and times TBA)

DB Connect

What is DB Connect?

RDBMS and Splunk Bridge

Highlights:

- Over 5800+ installations
- Flexible
- Runs real-time
- Java based

SQL and SPL

Together for a better world

splunk> .conf18

Where do I start?

...at the beginning

- SQL > SPL
- SPL > SQL
  - Generate SQL query e.g. In-list: last_name in (‘foo’,’bar’,’baz’)
- Join: small is beautiful
- Buttercup game:
  - Games data in MySQL
  - Site web access logs in Splunk
- Better solutions may exist

SQL > SPL

SQL > SPL

Winners VS losers: who buys most?

- Context:
  - find whether a correlation exists between victories and customers' behavior
- Steps:
  - finding the number of victories per session
  - join logs on session ID
  - analyze

SQL > SPL

```
| dbxquery connection=mysql query="
        SELECT sum(gu.victory) AS victories, session_id
        FROM conf2018.user u
        JOIN conf2018.game_user gu ON gu.user_id = u.id
        JOIN conf2018.game g ON g.id = gu.game_id
        JOIN conf2018.user_session us ON us.user_id = u.id
        GROUP BY session_id"
| join session_id
        [
        search source="tutorialdata.zip:*" sourcetype=access_combined_wcookie JSESSIONID action=addtocart productId
        | rename JSESSIONID as session_id
        ]
| stats count by victories
| sort + victories
```

splunk> .conf18

SQL > SPL

Getting number of victories from DB

```
| dbxquery connection=mysql query="
        SELECT sum(gu.victory) AS victories, session_id
        FROM conf2018.user u
        JOIN conf2018.game_user gu ON gu.user_id = u.id
        JOIN conf2018.game g ON g.id = gu.game_id
        JOIN conf2018.user_session us ON us.user_id = u.id
        GROUP BY session_id"
| join session_id
        [
        search source="tutorialdata.zip:*" sourcetype=access_combined_wcookie JSESSIONID action=addtocart productId
        | rename JSESSIONID as session_id
        ]
| stats count by victories
| sort + victories
```

SQL > SPL

Join on session ID with logs

```
| dbxquery connection=mysql query="
        SELECT sum(gu.victory) AS victories, session_id
        FROM conf2018.user u
        JOIN conf2018.game_user gu ON gu.user_id = u.id
        JOIN conf2018.game g ON g.id = gu.game_id
        JOIN conf2018.user_session us ON us.user_id = u.id
        GROUP BY session_id"
| join session_id
        [
        search source="tutorialdata.zip:*" sourcetype=access_combined_wcookie JSESSIONID action=addtocart productId
        | rename JSESSIONID as session_id
        ]
| stats count by victories
| sort + victories
```

splunk> .conf18

SQL > SPL

Analyze

```
| dbxquery connection=mysql query="
        SELECT sum(gu.victory) AS victories, session_id
        FROM conf2018.user u
        JOIN conf2018.game_user gu ON gu.user_id = u.id
        JOIN conf2018.game g ON g.id = gu.game_id
        JOIN conf2018.user_session us ON us.user_id = u.id
        GROUP BY session_id"
| join session_id
        [
        search source="tutorialdata.zip:*" sourcetype=access_combined_wcookie JSESSIONID action=addtocart productId
        | rename JSESSIONID as session_id
        ]
| stats count by victories
| sort + victories
```

SPL > SQL

Oopsie !

I forgot to write the sessions

- Context:
  - A bug prevents the sessions tracking in DB (Around 22th and 23rd of August)
  - Find missing data

Steps:

- finding all sessions from logs
- join to the user_session table
- get all missing records

SPL > SQL

```
source="tutorialdata.zip:*" sourcetype=access_combined_wcookie action=purchase productId
| dedup JSESSIONID
| fields JSESSIONID
| rename JSESSIONID as temp_table
| eval temp_table="'".temp_table."'"
| mvcombine delim="AS ID UNION SELECT " temp_table
| nomv temp_table
| eval temp_table="SELECT ".temp_table
| eval search_query="
                SELECT * FROM (".temp_table.") ids
                WHERE NOT EXISTS (
                    SELECT session_id FROM conf2018.bad_user_session us
                    WHERE us.session_id = ids.id)"
| fields search_query
| map search="dbxquery connection=mysql query=$search_query$"
```

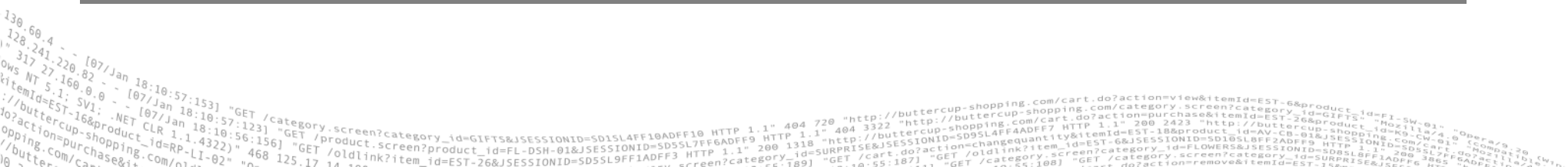
splunk> .conf18

SPL > SQL

Finding sessions

source="tutorialdata.zip:*" sourcetype=access_combined_wcookie action=purchase productId

```
| dedup JSESSIONID
| fields JSESSIONID
| rename JSESSIONID as temp_table
| eval temp_table="'".temp_table."'"
| mvcombine delim="AS ID UNION SELECT " temp_table
| nomv temp_table
| eval temp_table="SELECT ".temp_table
| eval search_query="
                    SELECT * FROM (".temp_table.") ids
                   WHERE NOT EXISTS (
                       SELECT session_id FROM conf2018.bad_user_session us
                       WHERE us.session_id = ids.id)"
| fields search_query
| map search="dbxquery connection=mysql query=$search_query$"
```

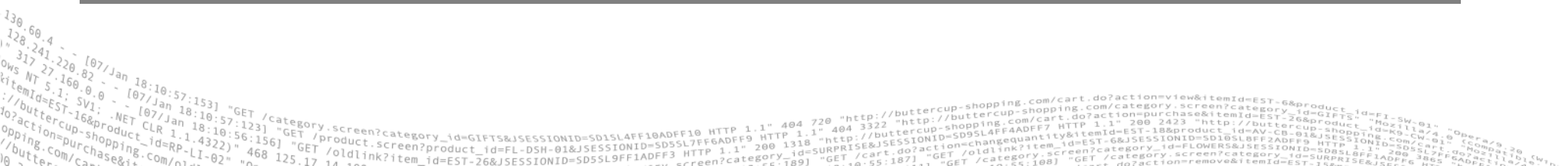


splunk> .conf18

SPL > SQL

Create a temporary table with all sessions

```
source="tutorialdata.zip:*" sourcetype=access_combined_wcookie action=purchase productId
| dedup JSESSIONID
| fields JSESSIONID
| rename JSESSIONID as temp_table
| eval temp_table="'".temp_table."'"
| mvcombine delim="AS ID UNION SELECT " temp_table
| nomv temp_table
| eval temp_table="SELECT ".temp_table
| eval search_query="
                SELECT * FROM (".temp_table.") ids
                WHERE NOT EXISTS (
                    SELECT session_id FROM conf2018.bad_user_session us
                    WHERE us.session_id = ids.id)"
| fields search_query
| map search="dbxquery connection=mysql query=$search_query$"
```



splunk> .conf18

SPL > SQL

Build anti-join query

```
source="tutorialdata.zip:*" sourcetype=access_combined_wcookie action=purchase productId
| dedup JSESSIONID
| fields JSESSIONID
| rename JSESSIONID as temp_table
| eval temp_table="'".temp_table."'"
| mvcombine delim="AS ID UNION SELECT " temp_table
| nomv temp_table
| eval temp_table="SELECT ".temp_table
| eval search_query="
        SELECT * FROM (".temp_table.") ids
        WHERE NOT EXISTS (
            SELECT session_id FROM conf2018.bad_user_session us
            WHERE us.session_id = ids.id)"
| fields search_query
| map search="dbxquery connection=mysql query=$search_query$"
```

splunk> .conf18

SPL > SQL

Inject the query into SPL and run it

```
source="tutorialdata.zip:*" sourcetype=access_combined_wcookie action=purchase productId
| dedup JSESSIONID
| fields JSESSIONID
| rename JSESSIONID as temp_table
| eval temp_table="'".temp_table."'"
| mvcombine delim="AS ID UNION SELECT " temp_table
| nomv temp_table
| eval temp_table="SELECT ".temp_table
| eval search_query="
                    SELECT * FROM (".temp_table.") ids
                   WHERE NOT EXISTS (
                       SELECT session_id FROM conf2018.bad_user_session us
                       WHERE us.session_id = ids.id)"
| fields search_query
| map search="dbxquery connection=mysql query=$search_query$"
```

splunk> .conf18

Stored procedures

When a simple query is not enough

DBX and Queries

This is where the subtitle goes

DBX Input

- Single Query ONLY
- Returns results
- Can’t call a stored procedure directly. (or maybe you can?)

dbxquery

- Single statement: SELECT, INSERT, UPDATE or DELETE
- Can also call a stored procedure with parameters

What is a SQL Query?

A Single Select Statement

Single

```
SELECT column_name_1, column_name_2
  FROM table_name
 WHERE column_name_3 = 'ABC'
```

Single

```
SELECT column_name_1, column_name_2
  FROM table_name
 WHERE column_name_3 > ?
 ORDER BY column_name_3 ASC
```

Compound With DDL

```
CREATE TABLE my_temp_table;

INSERT INTO my_temp_table
SELECT *
  FROM some_other_table;

-- manipulate my_temp_table

SELECT *
  FROM my_temp_table;

DROP TABLE my_temp_table;
```

splunk> .conf18

Why and How

Why

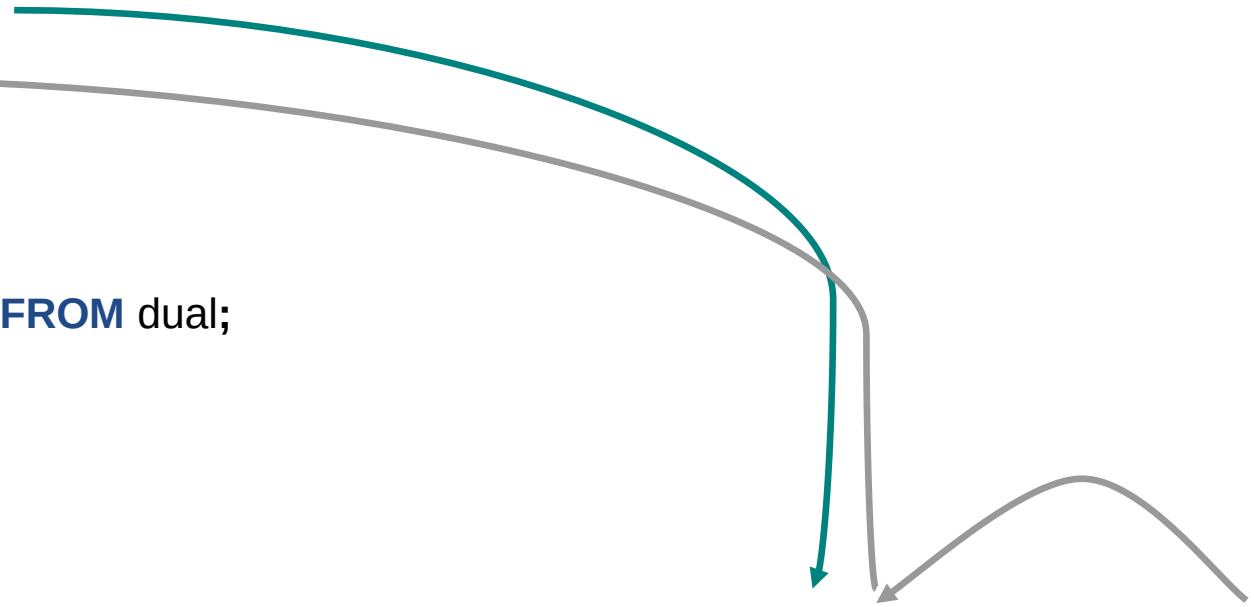
- Compound statements
- Deletes
- Temp tables
- DDL

Oracle Example 1

DB Connect Doc > dbxquery command

Oracle PL/SQL

```
CREATE OR REPLACE PROCEDURE test_orasp_1(
  p_ref_cursor  OUT SYS_REFCURSOR,
  p_var_in          IN  VARCHAR)
AS
BEGIN
 OPEN p_ref_cursor FOR
 SELECT 'you passed-in: '|| p_var_in out_var FROM dual;
END test_orasp_1;
```



Splunk SPL

```
| dbxquery connection=splunk_test procedure="{call test_orasp_1(?,?) }" params="foo"
```

Demo Stored Procedures

- Return Input
- Filter a query
- Compound Statement

splunk> .conf18

Demo: Stored Procedures

Return Input
Filter a query
Compound Statement

Oracle Example 2

Can be used as an input

Implementation Details

Oracle

- **PL/SQL Package Contains**
  - **A “type” that represents an array of rows (basically a table)**
  - **Pipelined Table Function**
    - **Returns rows as the “type” defined above**
    - **Can contain DML and DDL as long as it has “pragma autonomous_transaction”**

Splunk

- **Query the table function instead of a table**
- **Can be from an input or dbxquery**

Oracle Example 2

Oracle PL/SQL

```
-- package specification
  TYPE sample_data_tbl is table of sample_data%ROWTYPE;
-- package body
-- ...
 function get_sample_data(p_min_id IN NUMBER) return sample_data_tbl
 PIPELINED as
          cursor sample_data_cur is
            SELECT * FROM sample_data where id > p_min_id order by id asc;
 begin
          for current_row in sample_data_cur loop
            pipe row(current_row);
          end loop;
 end get_sample_data;
```

Oracle Example 2

SQL for Splunk Input

Batch

```
SELECT *
 FROM TABLE(sample_pkg.get_sample_data(0))
ORDER by id asc;
```

Rising Column

```
SELECT *
 FROM TABLE(sample_pkg.get_sample_data(?))
ORDER by id asc;
```

Demo: Pipelined Table Function

Input “Queries” a function that can perform other procedural operations then returns rows

Thank You

Don't forget to rate this session in the .conf18 mobile app

Q&A

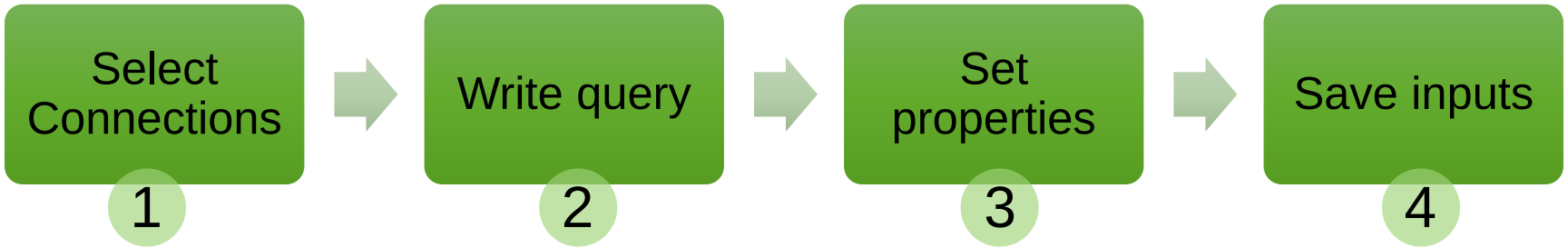
- End of session in room
- After session outside of room
- DBX Office Hours at the “Foundations & Platform Booth” #12 (dates and times TBA)

Additional Content

Input Bulk Operations

One query to Many Inputs

- Time saver
  - Create or edit many similar inputs at the same time
- Flexible
  - Input can still be edited individually



Input templates

A better version of cookies mix

- Pre-made inputs
- 3 fields unique to each input
  - Connection
  - Input name
  - Index