The Eye

A data ingestion pipeline for casino gaming analytics

W205 Final Presentation

Mohammad J. Habib

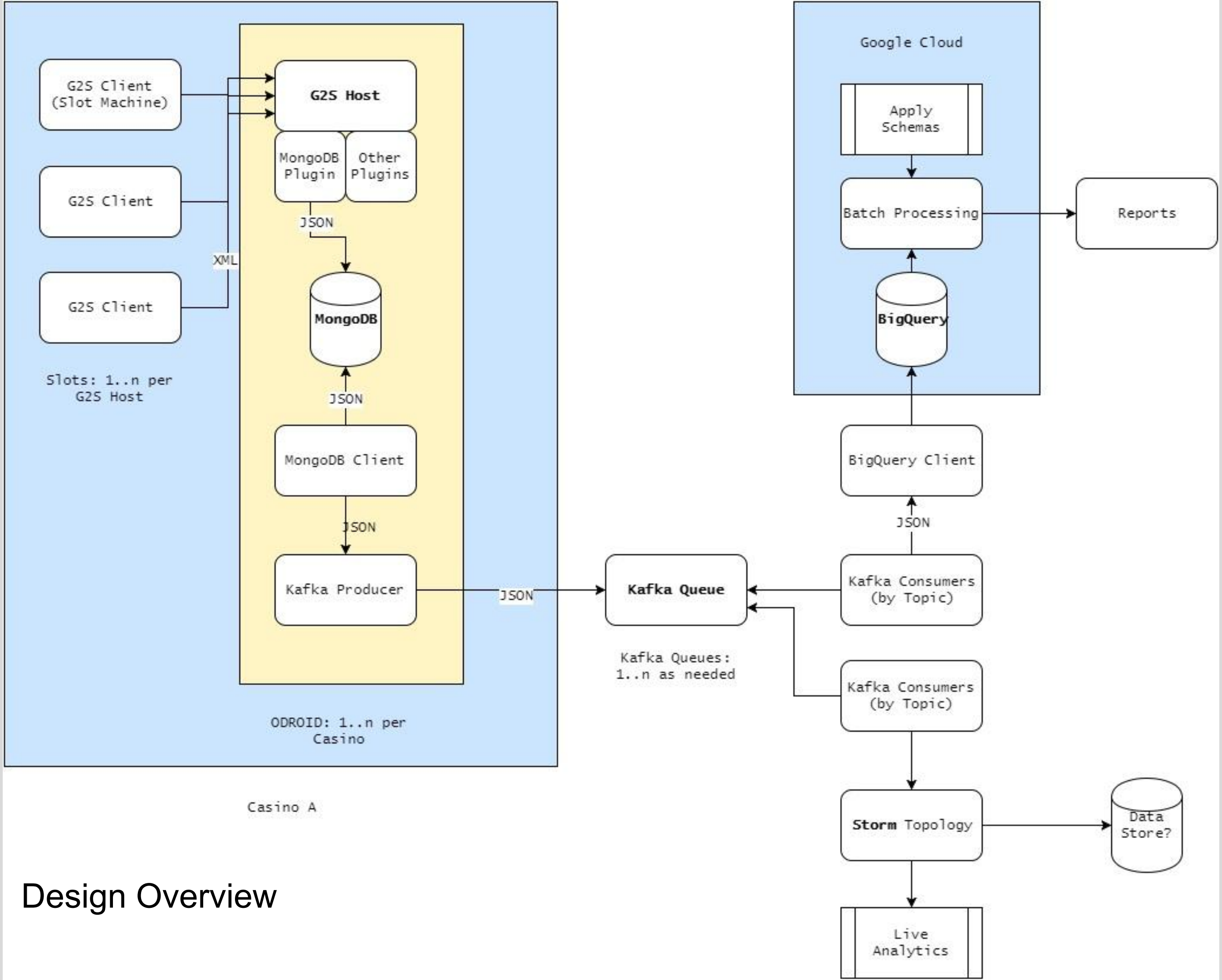
Problem Overview

Getting data from slot machines is very difficult for vendors:

- Regulators strictly control access to gaming machines
- Casino operators guard accounting data gathered from slot machines
- Casinos traditionally have Serial networks (not Ethernet); Web connectivity is spotty at best
- Using data for billing and game performance analytics is not easily possible

Plan for getting data

- Introduce a cheap hardware device that can “attach” to EGMs and listen for data (ODROID C1+)
- Use G2S; a common protocol used by many slot machine vendors
- Use VPN over cellular network to get data out of Casinos
- Batch data locally and send when connection becomes available



Design Overview

Scalability

Architecture scales horizontally:

- Add ODROIDs to support more Slot machines
- Add more Kafka queues to ingest higher volume of data
- Add more consumers as range of topics and volume grows
- BigQuery datasets can be created per Casino or Region
- BigQuery tables can be partitioned by date
- Add storage or remove old data as needed

Extensibility

- Stream processing can be added for live analytics (just add a new consumer)
- Other data processing systems can be bolted on e.g. Spark
- Bigquery data can be exposed to Tableau, Google Sheets and other business apps
- Bigquery can be replaced with another backend e.g. Hadoop

Known Limitations

The “proof of concept” developed for this project has following limitations. I’ve also listed how these limitations can be addressed for production.

Limitation in Proof of Concept	**How to address in Production?**
Data are not assigned a unique ODROID identifier and therefore we cannot tell apart data from one ODROID and another	• Add an ODROID identifier to each row of data e.g. ODROID public IP or MAC address, Or • Store data from each different ODROID in a different table
Data are currently not removed from the MongoDB on ODROID	• Purge data after a certain number of days from MongoDB to free up disk space on ODROID
Data is batched up in an in-memory Python queue before sending to BigQuery	• Replace in-memory queue with something like Redis
BigQuery tables are not partitioned (one table is forever growing)	• Partition tables based on date, or by ODROID + Date
Storm topology is not yet implemented (lack of time)	• This can be implemented in the future for production

Complexity and Storage

Architecture if fairly straightforward and decoupled:

- Data comes in as XML, gets transformed to JSON and makes it's way through the pipeline. It is stored as JSON in BigQuery
- Schemas are applied in BigQuery and the data can be combined with additional data (e.g. marketing identifiers for games)
- Complexity can be further reduced by RESTifying the data exchange between various components

Storage and throughput requirements are fairly demanding:

- Load test showed that 10 Slots generate ~300MB of data in 12 hours when set to auto-play (one wager per 3 seconds per slot).
- Guessing that we will need to move ~150MB per Casino per day. Since Slots are not constantly in auto-play.
- There are 1000 or so Casinos who are Aristocrat's customers. Overall ~150GB of data moved and stored per day.

Google BigQuery | RegExr: Learn, Build, & Tes | Query Reference | BigQu

https://bigquery.cloud.google.com/results/sauronbigquery:bquijob_4290e9e3_1566c83cce2

Apps | Log in to TeamCity -- | AgileCraft | CodeSkulptor | Grid R | SOA - Society of Actu | A Designers Guide to | Solving Polynomial In | VGT VersionOne | GMAT | JIRA | Other bookmarks

Google BigQuery

New Query ?

Query Editor | UDF Editor

```
SELECT
  egmId, wagerCategory,
  COUNT(wagerCategory) AS wagerCategoryCounts
FROM (
  SELECT
    _id,
    REGEXP_EXTRACT(Key, r'([^-]*)') AS egmId,
    JSON_EXTRACT(Value, '$.meterList.meterInfo[1].wagerMeters[0].wagerCategory') AS wagerCategory
  FROM (
    SELECT
      *
```

RUN QUERY | Save Query | Save View | Format Query | Show Options | Query complete (0.8s elapsed, cached)

Results | Explanation | Download as CSV | Download as JSON | Save as Table | Save to Google Sheets

Row	egmId	dateTime	wager	win
1	ATI_GEN7_3064128B17	2016-07-21 15:53:06 UTC	1.25	5.0
2	ATI_GEN7_3064128B17	2016-07-21 15:53:20 UTC	1.25	1.75
3	ATI_GEN7_3064128B17	2016-07-21 15:53:24 UTC	1.25	0.0
4	ATI_GEN7_3064128B17	2016-07-21 15:53:28 UTC	1.25	0.0
5	ATI_GEN7_3064128B17	2016-07-21 15:53:31 UTC	1.25	0.0
6	ATI_GEN7_3064128B17	2016-07-21 15:53:33 UTC	1.25	0.0
7	ATI_GEN7_3064128B17	2016-07-21 15:55:01 UTC	1.25	32.75
8	ATI_GEN7_3064128B17	2016-07-21 15:59:20 UTC	1.25	0.0
9	ATI_GEN7_3064128B17	2016-07-21 16:04:59 UTC	1.25	0.0
10	ATI_GEN7_3064128B17	2016-07-21 16:05:06 UTC	1.25	4.0
11	ATI_GEN7_3064128B17	2016-07-21 16:05:07 UTC	1.25	0.0

Table | JSON

First < Prev Rows 1 - 11 of 17351 Next > Last

Google BigQuery | RegExr: Learn, Build, & Tes | Query Reference | BigQu

https://bigquery.cloud.google.com/results/sauronbigquery:bquijob_361987ca_1566def029a

Apps | Log in to TeamCity -- | AgileCraft | CodeSkulptor | Grid R | SOA - Society of Actu | A Designers Guide to | Solving Polynomial In | VGT VersionOne | GMAT | JIRA | Other bookmarks

Google BigQuery

New Query

Query Editor | UDF Editor

```
SELECT * FROM [sauronbigquery:sauron.view_winLossByEgm] LIMIT 1000
```

Valid: This query will process 422 MB when run.

RUN QUERY | Save Query | Save View | Format Query | Show Options

Query complete (3.3s elapsed, 422 MB processed)

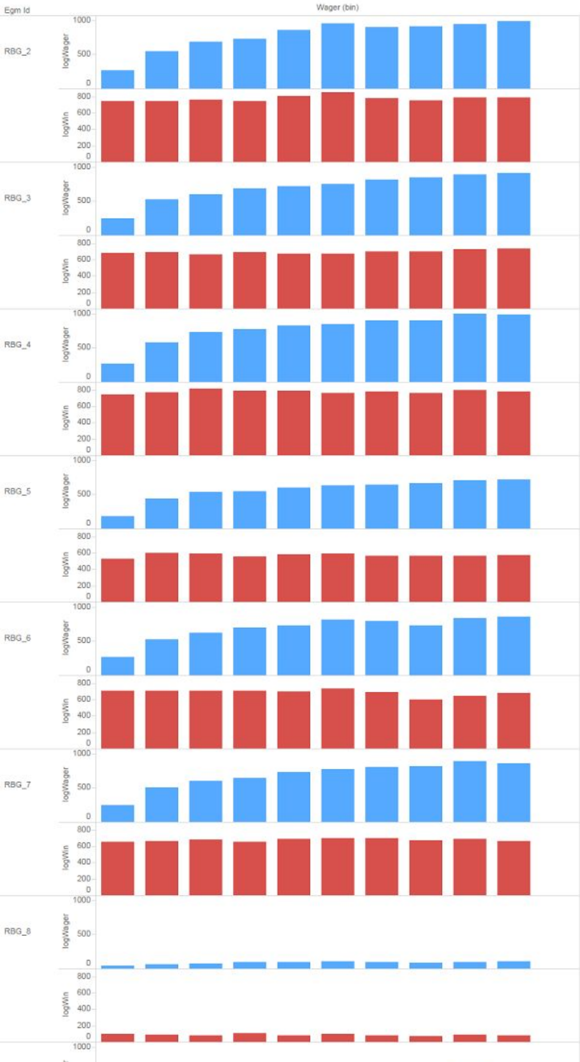
Results | Explanation

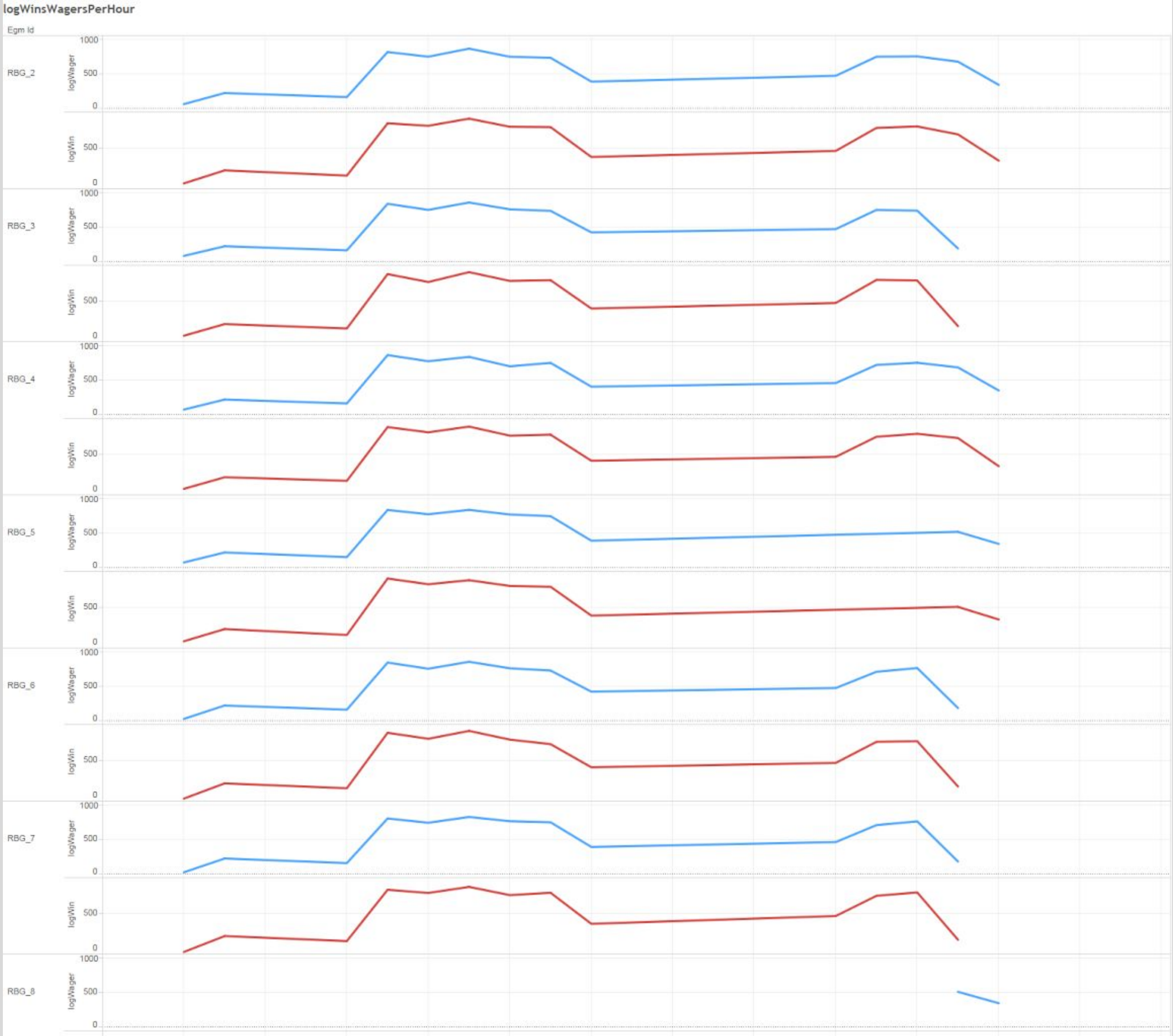
Download as CSV | Download as JSON | Save as Table | Save to Google Sheets

Row	egmId	totalBet	totalWin	netWin	winRatio
1	ATI_GEN7_3064128B17	885625000	878725000	-6900000	0.9922088920253944
2	RBG_1234	1026600000	53870000	-972730000	0.05247418663549903

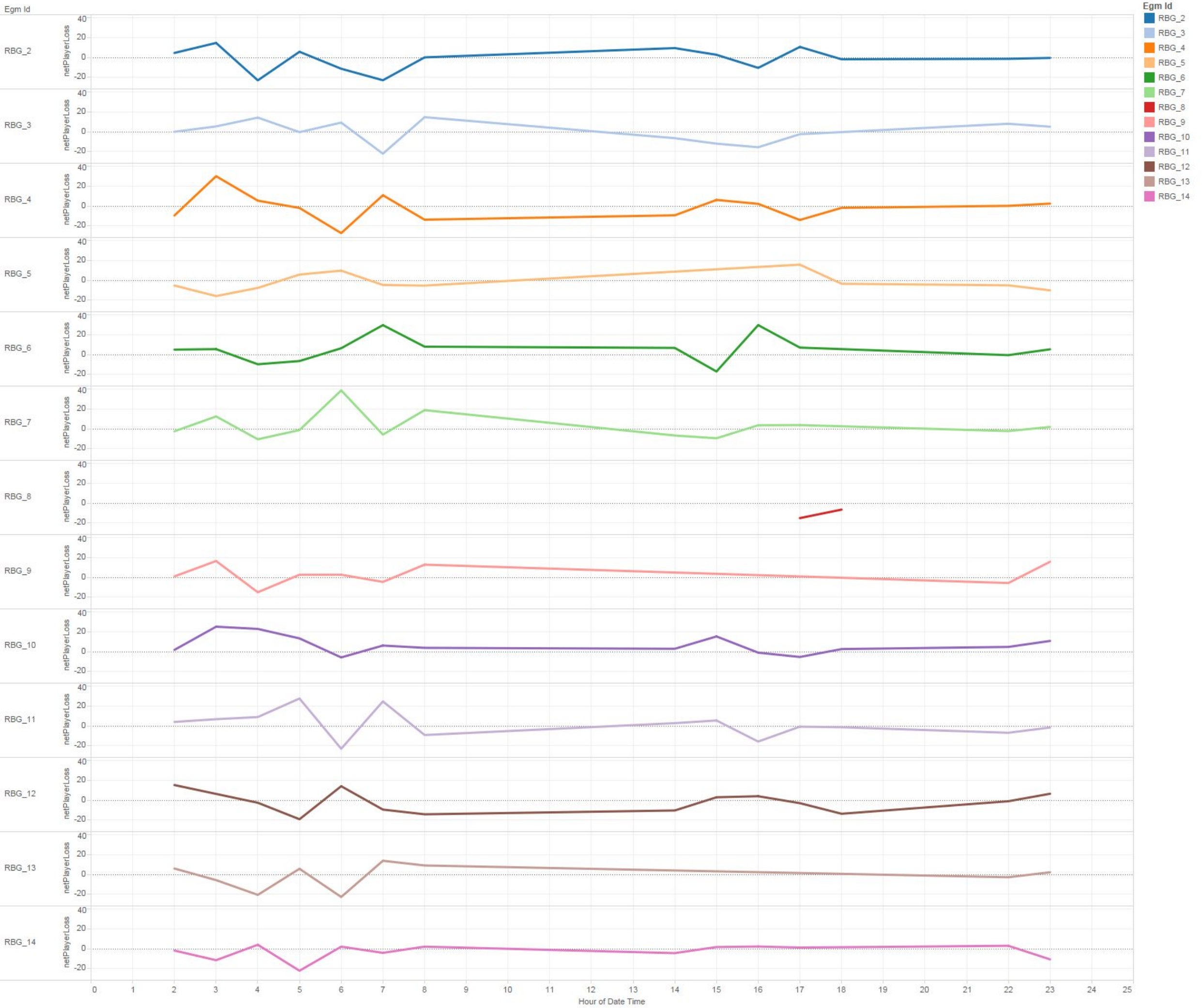
Table | JSON

wagersWinsHistogram

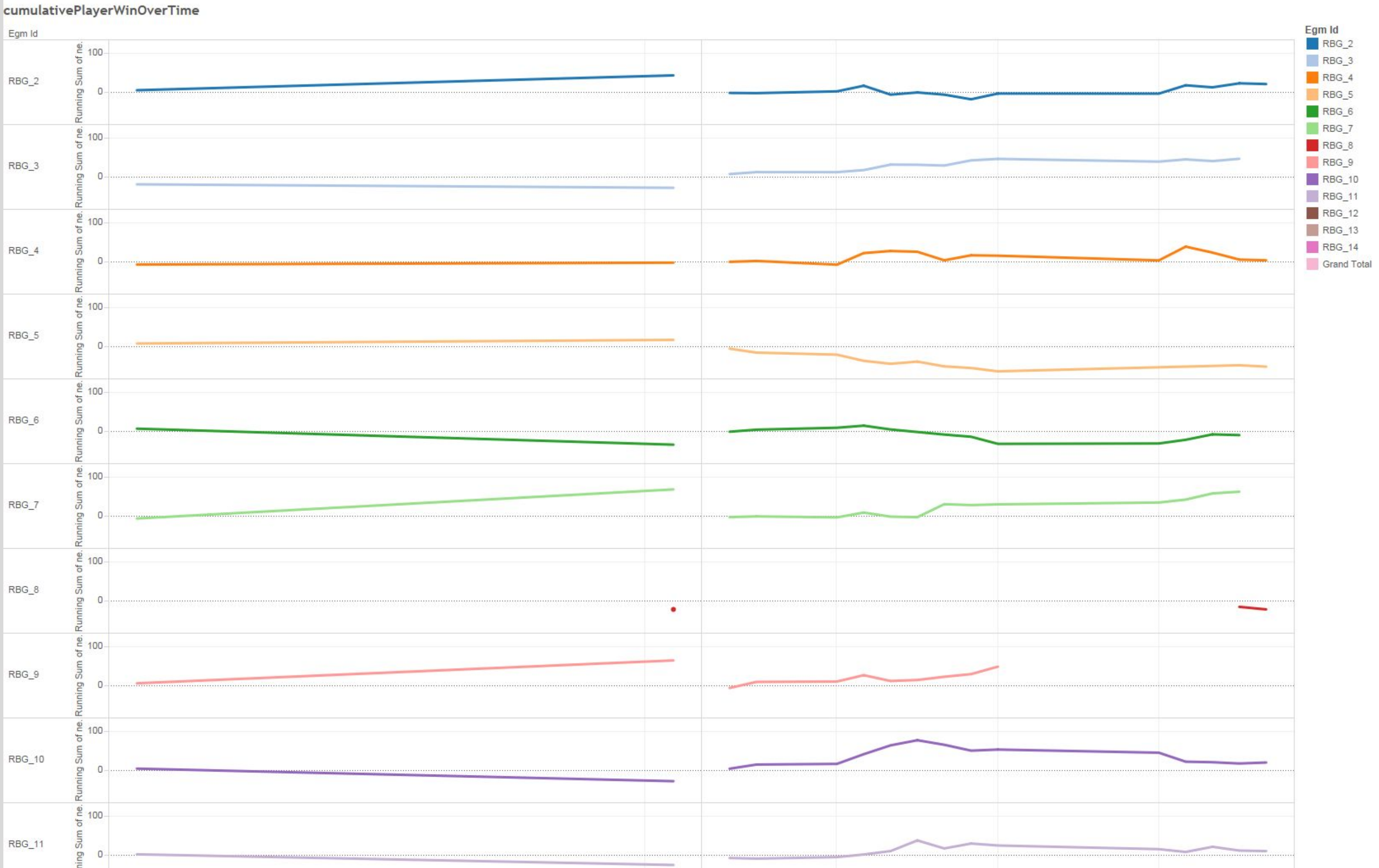



logWinsWagersPerHour
Egm Id
RBG_2
RBG_3
RBG_4
RBG_5
RBG_6
RBG_7
RBG_8
logWager
logWin
1000
500
0

netPlayerWinPerHour



The trend of sum of netPlayerLoss for Date Time Hour broken down by Egm Id. Color shows details about Egm Id.


cumulativePlayerWinOverTime
Egm Id
RBG_2
RBG_3
RBG_4
RBG_5
RBG_6
RBG_7
RBG_8
RBG_9
RBG_10
RBG_11
Running Sum of ne..
100
0
Egm Id
RBG_2
RBG_3
RBG_4
RBG_5
RBG_6
RBG_7
RBG_8
RBG_9
RBG_10
RBG_11
RBG_12
RBG_13
RBG_14
Grand Total