SQL interface

The meat of AzureCosmosR is a suite of methods to work with databases, containers (tables) and documents (rows) using the SQL API.

```
library(dplyr)
library(AzureCosmosR)

# endpoint object for this account
endp <- cosmos_endpoint(
    "https://myaccount.documents.azure.com:443/",
    key="mykey"
)

# all databases in this account
list_cosmos_databases(endp)

# a specific database
db <- get_cosmos_database(endp, "mydatabase")

# create a new container and upload the Star Wars dataset from dplyr
cont <- create_cosmos_container(db, "mycontainer", partition_key="sex")
bulk_import(cont, starwars)

query_documents(cont, "select * from mycontainer")

# an array select: all characters who appear in ANH
query_documents(cont,
    "select c.name
        from mycontainer c
        where array_contains(c.films, 'A New Hope')")
```

You can easily create and execute JavaScript stored procedures and user-defined functions:

```
proc <- create_stored_procedure(
    cont,
    "helloworld",
    'function () {
        var context = getContext();
        var response = context.getResponse();
        response.setBody("Hello, World");
    }'
)

exec_stored_procedure(proc)

create_udf(cont, "times2", "function(x) { return 2*x; }")

query_documents(cont, "select udf.times2(c.height) from cont c")
```

Aggregates take some extra work, as the Cosmos DB REST API currently only has limited

support for cross-partition queries. Set `by_pkrange=TRUE` in the `query_documents` call, which will run the query on each partition key range (physical partition) and return a list of data frames. You can then process the list to obtain an overall result.

```
# average height by sex, by pkrange
df_lst <- query_documents(cont,
    "select c.gender, count(1) n, avg(c.height) height
        from mycontainer c
        group by c.gender",
    by_pkrange=TRUE
)

# combine pkrange results
df_lst %>%
    bind_rows(.id="pkrange") %>%
    group_by(gender) %>%
    summarise(height=weighted.mean(height, n))
```

Full support for cross-partition queries, including aggregates, may come in a future version of AzureCosmosR.

Other client interfaces

MongoDB

You can query data in a MongoDB-enabled Cosmos DB instance using the mongolite package. AzureCosmosR provides a simple bridge to facilitate this.

```
endp <- cosmos_mongo_endpoint(
    "https://myaccount.mongo.cosmos.azure.com:443/",
    key="mykey"
)

# a mongolite::mongo object
conn <- cosmos_mongo_connection(endp, "mycollection", "mydatabase")
conn$find("{}")
```

For more information on working with MongoDB, see the mongolite documentation.

Table storage

You can work with data in a table storage-enabled Cosmos DB instance using the AzureTableStor package.

```
endp <- AzureTableStor::table_endpoint(
    "https://myaccount.table.cosmos.azure.com:443/",
    key="mykey"
)

tab <- AzureTableStor::storage_table(endp, "mytable")
AzureTableStor::list_table_entities(tab, filter="firstname eq 'Satya'")
```