# Terminology

SQL and RethinkDB share very similar terminology. Below is a table of terms and concepts in the two systems.

| **SQL** | **RethinkDB** |
| --- | --- |
| database | database |
| table | table |
| row | document |
| column | field |
| table joins | table joins |
| primary key | primary key (by default id) |
| index | index |

# INSERT

This is a list of queries for inserting data into a database.

| **SQL** | **ReQL** |
| --- | --- |
| **INSERT**  **INTO**  users(user\_id,  age,  name)  **VALUES**  ("f62255a8259f",  30,  Peter) | r**.**table("users")**.**insert({  "user\_id":  "f62255a8259f",  "age":  30,  "name":  "Peter"  }) |

# SELECT

This is a list of queries for selecting data out of a database.

| **SQL** | **ReQL** |
| --- | --- |
| **SELECT**  **\***  **FROM**  users | r**.**table("users") |
| **SELECT**  user\_id,  name  **FROM**  users | r**.**table("users")  **.**pluck("user\_id",  "name") |
| **SELECT**  **\***  **FROM**  users  **WHERE**  name  **=**  "Peter" | r**.**table("users")**.**filter({  "name":  "Peter"  })  If you have a secondary index built on the field name, you can run a more efficient query:  r**.**table("users")  **.**get\_all("Peter",  index**=**"name") |
| **SELECT**  **\***  **FROM**  users  **WHERE**  name  **=**  "Peter"  **AND**  age  **=**  30 | r**.**table("users")**.**filter({  "name":  "Peter",  "age":  30  }) |
| **SELECT**  **\***  **FROM**  users  **WHERE**  name  **LIKE**  "P%" | r**.**table("users")**.**filter(  r**.**row['name']**.**match("^P")}  ) |
| **SELECT**  **\***  **FROM**  users  **ORDER**  **BY**  name  **ASC** | r**.**table("users")**.**order\_by("name") |
| **SELECT**  **\***  **FROM**  users  **ORDER**  **BY**  name  **DESC** | r**.**table("users")**.**order\_by(  r**.**desc("name")  ) |
| **SELECT**  user\_id  **FROM**  users  **WHERE**  name  **=**  "Peter"  **ORDER**  **BY**  name  **DESC** | r**.**table("users")**.**filter({  "name":  "Peter"  })**.**order\_by(  r**.**desc("name")  )**.**pluck("user\_id") |
| **SELECT**  **\***  **FROM**  users  **LIMIT**  5  SKIP  10 | r**.**table("users")**.**skip(10)**.**limit(5) |
| **SELECT**  **\***  **FROM**  users  **WHERE**  name  **IN**  ('Peter',  'John') | r**.**table("users")**.**filter(**lambda**  doc:  r**.**expr(["Peter",  "John"])  **.**contains(doc["name"])  )  If you have a secondary index built on the field name, you can run a more efficient query:  r**.**table("users")  **.**get\_all("Peter",  "John",  index**=**"name") |
| **SELECT**  **\***  **FROM**  users  **WHERE**  name  **NOT**  **IN**  ('Peter',  'John') | r**.**table("users")**.**filter(**lambda**  doc:  r**.**expr(["Peter",  "John"])  **.**contains(doc["name"])  **.not**()  ) |
| **SELECT**  **COUNT**(**\***)  **FROM**  users | r**.**table("users")**.**count() |
| **SELECT**  **COUNT**(name)  **FROM**  users  **WHERE**  age  **>**  18 | r**.**table("users")**.**filter(  (r**.**row**.**has\_fields("name"))  **&**  (r**.**row["age"]  **>**  18)  )**.**count() |
| **SELECT**  **AVG**("age")  **FROM**  users | r**.**table("users")  **.**avg("age") |
| **SELECT**  **MAX**("age")  **FROM**  users | r**.**table("users")["age"]  **.**max() |
| **SELECT**  **DISTINCT**(name)  **FROM**  users | r**.**table("users")**.**pluck("name")**.**distinct() |
| **SELECT**  **\***  **FROM**  users  **WHERE**  age  **BETWEEN**  18  **AND**  65; | r**.**table("users")**.**filter(  (r**.**row["age"]  **>=**  18)  **&**  (r**.**row["age"]  **>=**  65)  If you have a secondary index built on the field age, you can run a more efficient query:  r**.**table("users")  **.**between(18,  65,  index**=**"age") |
| **SELECT**  name,  'is\_adult'  **=**  **CASE**  **WHEN**  age**>**18  **THEN**  'yes'  **ELSE**  'no'  **END**  **FROM**  users | r**.**table("users")**.**map({  "name":  r**.**row["name"],  "is\_adult":  r**.**branch(  r**.**row["age"]  **>**  18,  "yes",  "no"  )  }) |
| **SELECT**  **\***  **FROM**  posts  **WHERE**  **EXISTS**  (**SELECT**  **\***  **FROM**  users  **WHERE**  posts.author\_id  **=**  users.id) | r**.**table("posts")  **.**filter(**lambda**  post:  r**.**table("users")  **.**filter(  **lambda**  user:  user**.**id  **==**  post**.**author\_id  )**.**count()  **>**  0  ) |

# UPDATE

This is a list of commands for updating data in the database.

| **SQL** | **ReQL** |
| --- | --- |
| **UPDATE**  users  **SET**  age  **=**  18  **WHERE**  age  **<**  18 | r**.**table("users")**.**filter(  r**.**row["age"]  **<**  18  )**.**update({  "age":  18  }) |
| **UPDATE**  users  **SET**  age  **=**  age**+**1 | r**.**table("users")**.**update(  {  "age":  r**.**row["age"]**+**1  }  ) |

# DELETE

This is a list of queries for deleting data from the database.

| **SQL** | **ReQL** |
| --- | --- |
| **DELETE**  **FROM**  users | r**.**table("users")**.**delete() |
| **DELETE**  **FROM**  users  **WHERE**  age  **<**  18 | r**.**table("users")  **.**filter(  r**.**row["age"]  **<**  18)  **.**delete() |

# JOINS

This is a list of queries for performing joins between multiple tables.

| **SQL** | **ReQL** |
| --- | --- |
| **SELECT**  **\***  **FROM**  posts  **JOIN**  users  **ON**  posts.user\_id  **=**  users.id | r**.**table("posts")**.**inner\_join(  r**.**table("users"),  **lambda**  post,  user:  post["user\_id"]  **==**  user["id"]  )**.**zip()  **Note:** zip() will merge the user in the post, overwriting fields in case of conflict.  If you have an index (primary key or secondary index) built on the field of the right table, you can perform a more efficient join with [eq\_join](http://rethinkdb.com/api/python/eq_join/).  r**.**table("posts")**.**eq\_join(  "id",  r**.**table("users"),  index**=**"id"  )**.**zip() |
| **SELECT**  posts.id  **AS**  post\_id,  **user**.name,  users.id  **AS**  user\_id  **FROM**  posts  **JOIN**  users  **ON**  posts.user\_id  **=**  users.id    **SELECT**  posts.id  **AS**  post\_id,  **user**.name,  users.id  **AS**  user\_id  **FROM**  posts  **INNER**  **JOIN**  users  **ON**  posts.user\_id  **=**  users.id | r**.**table("posts")**.**inner\_join(  r**.**table("users"),  **lambda**  post,  user:  post["user\_id"]  **==**  user["id"]  )**.**map({  "post\_id":  r**.**row["left"]["id"],  "user\_id":  r**.**row["right"]["id"],  "name":  r**.**row["right"]["name"]  }) |
| **SELECT**  **\***  **FROM**  posts  **RIGHT**  **JOIN**  users  **ON**  posts.user\_id  **=**  users.id    **SELECT**  **\***  **FROM**  posts  **RIGHT**  **OUTER**  **JOIN**  users  **ON**  posts.user\_id  **=**  users.id | r**.**table("posts")**.**outer\_join(  r**.**table("users"),  **lambda**  post,  user:  post["user\_id"]  **==**  user["id"]  )**.**zip()  **Note**: You can perform more efficient OUTER JOINoperations with the [concat\_map](http://rethinkdb.com/api/python/concat_map/) command.  r**.**table("posts")**.**concat\_map(**lambda**  post:  r**.**table("users")  **.**get\_all(post["id"],index**=**"id")  **.**do(  **lambda**  results:  r**.**branch(  results**.**count()  **==**  0,  [{"left":  post}],  results**.**map(  **lambda**  user:  {  "left":  post  "right":  user  }  )  )  )  )**.**zip() |
| **SELECT**  **\***  **FROM**  posts  **LEFT**  **JOIN**  users  **ON**  posts.user\_id  **=**  users.id  **SELECT**  **\***  **FROM**  posts  **LEFT**  **OUTER**  **JOIN**  users  **ON**  posts.user\_id  **=**  users.id | r**.**table("users")**.**outer\_join(  r**.**table("posts"),  **lambda**  user,  post:  post**.**user\_id  **==**  user**.**id  )**.**zip()  r**.**table("users")**.**concat\_map(**lambda**  user:  r**.**table("posts")  **.**get\_all(user["id"],index**=**"id")  **.**do(  **lambda**  results:  r**.**branch(  results**.**count()  **==**  0,  [{"left":  user}],  results**.**map(  **lambda**  post:  {  "left":  user  "right":  post  }  )  )  )  )**.**zip() |

# AGGREGATIONS

This is a list of queries for performing data aggregation.

| **SQL** | **ReQL** |
| --- | --- |
| **SELECT**  category  **FROM**  posts  **GROUP**  **BY**  category | r**.**table("posts")**.**map(  r**.**row["category"]  )**.**distinct() |
| **SELECT**  category,  **SUM**('num\_comments')  **FROM**  posts  **GROUP**  **BY**  category | r**.**table('posts')  **.**group('category')  **.**sum('num\_comments') |
| **SELECT**  category,  status,  **SUM**('num\_comments')  **FROM**  posts  **GROUP**  **BY**  category,  status | r**.**table("posts")  **.**group('category',  'status')  **.**sum('num\_comments') |
| **SELECT**  category,  **SUM**(num\_comments)  **FROM**  posts  **WHERE**  num\_comments  **>**  7  **GROUP**  **BY**  category | r**.**table("posts")  **.**filter(r**.**row['num\_comments']**>**7)  **.**group('category')  **.**sum('num\_comments') |
| **SELECT**  category,  **SUM**(num\_comments)  **FROM**  posts  **GROUP**  **BY**  category  **HAVING**  num\_comments  **>**  7 | r**.**table("posts")  **.**group('category')  **.**sum('num\_comments')  **.**ungroup()  **.**filter(r**.**row["reduction"]  **>**  7) |

# TABLE and DATABASE manipulation

This is a list of queries for creating and dropping tables and databases.

| **SQL** | **ReQL** |
| --- | --- |
| **CREATE**  **DATABASE**  my\_database; | r**.**db\_create('my\_database') |
| **DROP**  **DATABASE**  my\_database; | r**.**db\_drop('my\_database') |
| **CREATE**  **TABLE**  users  (id  INT  **IDENTITY**(1,1)  **PRIMARY**  **KEY**,  name  VARCHAR(50),  age  INT); | r**.**table\_create('users',  primary\_key**=**"id")  **Note:** RethinkDB is a NoSQL database and does not enforce schemas.  **Note:** The default primary key is id |
| **TRUNCATE**  **TABLE**  users; | r**.**table("users")**.**delete() |
| **DROP**  **TABLE**  users; | r**.**table\_drop("users") |