


RDB Lesson 2 Reference Notes



Karl Krueger

Last modified Jun 22, 2017 by Karl Krueger

This resource is deprecated and no longer being maintained.

Please refer to the materials in the Classroom and Discussion Forum.

These reference pages for the Relational Databases course have been moved into the Classroom.

- [SQL Data Types](#)
 - [Text and string types](#)
 - [Numeric types](#)
 - [Date and time types](#)
- [Select statement](#)
 - [where](#)
 - [limit / offset](#)
 - [order by](#)
 - [group by](#)
 - [having](#)
- [All the tables in the zoo database](#)
 - [animals](#)
 - [diet](#)
 - [taxonomy](#)
 - [ordernames](#)
- [Insert statement](#)

SQL Data Types

Here's just a sampling of the many data types that SQL supports. We won't be using most of these types in this course, though.

The exact list of types differs from one database to another. For a full list of types, check the manual for your database, such as [this one](#) for PostgreSQL.

Text and string types

text — a string of any length, like Python **str** or **unicode** types.

char(n) — a string of exactly *n* characters.

varchar(n) — a string of up to *n* characters.

Numeric types

integer — an integer value, like Python **int**.

real — a floating-point value, like Python **float**. Accurate up to six decimal places.

double precision — a higher-precision floating-point value. Accurate up to 15 decimal places.

decimal — an exact decimal value.

Date and time types

date — a calendar date; including year, month, and day.

time — a time of day.

timestamp — a date and time together.

Select statement

The most basic form of the **select** statement is to select a single scalar value:

select 2 + 2 ;

More usefully, we can select one or more columns from a table. With no restrictions, this will return all rows in the table:

select name, species **from** animals ;

Columns are separated by commas; use ***** to select all columns from the tables.

Quite often, we don't want *all* the data from a table. We can restrict the rows using a variety of *select clauses*, listed below. There are also a wide variety of functions that can apply to columns; including *aggregation* functions that operate on values from several rows, such as **max** and **count**.

where

The **where** clause expresses *restrictions* — filtering a table for rows that follow a particular rule. **where** supports equalities, inequalities, and boolean operators (among other things):

- **where species = 'gorilla'** — return only rows that have 'gorilla' as the value of the species column.
- **where name >= 'George'** — return only rows where the name column is alphabetically after 'George'.
- **where species != 'gorilla' and name != 'George'** — return only rows where species isn't 'gorilla' and name isn't 'George'.

limit / offset

The **limit** clause sets a limit on how many rows to return in the result table. The optional **offset** clause says how far to skip ahead into the results. So **limit 10 offset 100** will return 10 results starting with the 101st.

order by

The **order by** clause tells the database how to sort the results — usually according to one or more columns. So **order by species, name** says to sort results first by the species column, then by name within each species.

Ordering happens before limit/offset, so you can use them together to extract pages of alphabetized results. (Think of the pages of a dictionary.)

The optional **desc** modifier tells the database to order results in descending order — for instance from large numbers to small ones, or from Z to A.

group by

The **group by** clause is only used with aggregations, such as **max** or **sum**. Without a **group by** clause, a select statement with an aggregation will aggregate over the whole selected table(s), returning only one row. With a **group by** clause, it will return one row for each distinct value of the column or expression in the **group by** clause.

having

The **having** clause works like the **where** clause, but it applies after **group by** aggregations take place. The syntax is like this:

select columns from tables group by column having condition ;

Usually, at least one of the *columns* will be an aggregate function such as **count**, **max**, or **sum** on one of the tables' columns. In order to apply **having** to an aggregated column, you'll want to give it a name using **as**. For instance, if you had a table of items sold in a store, and you wanted to find all the items that have sold more than five units, you could use:

select name, count(*) as num from sales having num > 5;

You can have a **select** statement that uses only **where**, or only **group by**, or **group by** and **having**, or **where** and **group by**, or all three of them!

But it doesn't usually make sense to use **having** without **group by**.

If you use both **where** and **having**, the **where** condition will filter the rows that are going into the aggregation, and the **having** condition will filter the rows that come out of it.

You can read more about **having** here:

<http://www.postgresql.org/docs/9.4/static/sql-select.html#SQL-HAVING>

All the tables in the zoo database

animals

This table lists individual animals in the zoo. Each animal has only one row. There may be multiple animals with the same name, or even multiple animals with the same name and species.

- name — the animal's name (example: 'George')
- species — the animal's species (example: 'gorilla')
- birthdate — the animal's date of birth (example: '1998-05-18')

diet

This table matches up species with the foods they eat. Every species in the zoo eats at least one sort of food, and many eat more than one. If a species eats more than one food, there will be more than one row for that species.

- species — the name of a species (example: 'hyena')
- food — the name of a food that species eats (example: 'meat')

taxonomy

This table gives the (partial) biological taxonomic names for each species in the zoo. It can be used to find which species are more closely related to each other evolutionarily.

- name — the common name of the species (e.g. 'jackal')
- species — the taxonomic species name (e.g. 'aureus')
- genus — the taxonomic genus name (e.g. 'Canis')
- family — the taxonomic family name (e.g. 'Canidae')
- t_order — the taxonomic order name (e.g. 'Carnivora')

If you've never heard of this classification, don't worry about it; the details won't be necessary for this course. But if you're curious, Wikipedia articles [Taxonomy](#) and [Biological classification](#) may help.

ordernames

This table gives the common names for each of the taxonomic orders in the **taxonomy** table.

- t_order — the taxonomic order name (e.g. 'Cetacea')
- name — the common name (e.g. 'whales and dolphins')

Insert statement

The basic syntax for the **insert** statement:

```
insert into table ( column1, column2, ... ) values ( val1, val2, ... );
```

If the values are in the same order as the table's columns (starting with the first column), you don't have to specify the columns in the **insert** statement:

```
insert into table values ( val1, val2, ... );
```


For instance, if a table has three columns (**a**, **b**, **c**) and you want to insert into **a** and **b**, you can leave off the column names from the **insert** statement. But if you want to insert into **b** and **c**, or **a** and **c**, you have to specify the columns.

A single **insert** statement can only insert into a single table. (Contrast this with the **select** statement, which can pull data from several tables using a join.)


Karl's rainbow cat T-shirt is [The Time is Meow](#) by Rasabi, printed by Woot.

No labels

2 Comments



Anonymous
Ref2
Reply • Edit • Nov 30, 2016



Anonymous
Cool
Reply • Edit • Feb 07, 2017