SQL Reference Guide

Fundamentals

Creating a Table

```
CREATE TABLE cities (
  city VARCHAR(30) NOT NULL,
  state VARCHAR(30) NOT NULL,
  population INT
);
```

The `CREATE TABLE cities` statement creates and names a table within a database.

Within the parentheses, the data to be inserted is specified. Similar to headers in an Excel spreadsheet, `city`, `state`, and `population` define this data.

Inserting Values

```
INSERT INTO cities (city,
state, population)
VALUES ('Alameda',
'California', 79177);
```

The `INSERT INTO` statement first specifies the table and columns, then the `VALUES` line inserts the corresponding data.

Viewing the Table

Tables are easily viewed by using the `SELECT` statement.

```
SELECT *
FROM cities;
```

The `*` signifies selecting all available data, while `FROM` indicates the specific table to view.

Syntax note: the use of a semicolon signifies the completion of a code set, called a *statement terminator*. While most interpreters are smart enough to work without one, not all will and it is best practice to end a SQL statement in this manner.

Queries

Queries retrieve specific data from within a table. The query can be customized to be as broad or specific as the user wishes with the use of conditional operators.

WHERE:

```
// Return rows with a specific
value in column_a
SELECT *
FROM table_1
WHERE column_a = 'value';
```

WHERE and AND:

```
// Include multiple values with
the AND operator
SELECT *
FROM table_1
WHERE column_a = 'value'
AND column_b = 'value';
```

WHERE and OR:

```
// Return data containing
either one value or another
using the OR operator
SELECT *
FROM table_1
WHERE column_a = 'value'
OR column_b = 'value';
```

WHERE and IN:

```
// Return data containing
multiple values in a column
using the IN operator
SELECT *
FROM table_1
WHERE column_a IN ('value_a',
'value_b');
```

WHERE NOT:

```
// Exclude certain data with
the WHERE NOT operators
SELECT *
FROM table_1
WHERE NOT column_a = 'value';
```

WHERE and NOT IN:

```
// Exclude certain values from
a query using the NOT IN
operators
SELECT *
FROM table_1
WHERE column_a NOT IN
('value_a', 'value_b');
```

Joins

Data combined from different sets of data, or tables, is referred to as a join. Joins are completed using a column that is common between tables.

`LEFT JOIN`: returns all records from the left table and the matched records from the right table.

```
SELECT column(s)
FROM table_1
LEFT JOIN table_2
ON table_1.column_name =
table_2.column_name;
```

`RIGHT JOIN`: returns all records from the right table, and the matched records from the left table.

```
SELECT column(s)
FROM table_1
RIGHT JOIN table_2
ON table_1.column_name =
table_2.column_name;
```

`INNER JOIN`: returns records that have matching values in both tables.

```
SELECT column(s)
FROM table_1
INNER JOIN table_2
ON table_1.column_name =
table_2.column_name;
```

`CROSS JOIN`: returns records that match every row of the left table with every row of the right table. This type of join has the potential to make very large tables. Note that there are no additional conditions to be met to join the data.

```
SELECT column(s)
FROM table_1
CROSS JOIN table_2;
```

`FULL OUTER JOIN`: after an inner join is performed, null values are placed within the columns that do not match between the two tables. Note that the `OUTER` keyword is optional when using this join.

```
SELECT column(s)
FROM table_1
FULL OUTER JOIN table_b
ON table_1.column_name =
table_2.column_name;
```

Primary Keys

Primary Keys are a means of creating unique values for the data contained in tables.

```
CREATE TABLE people (
  id SERIAL PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  has_pet BOOLEAN DEFAULT false,
  pet_type VARCHAR(10) NOT NULL,
  pet_name VARCHAR(30),
  pet_age INT
);
```

The code block above creates the `people` table. The line `id SERIAL PRIMARY KEY,` indicates a column titled `id`, `SERIAL` signifies that each row will be auto-incremented, and `PRIMARY KEY` stipulates that this column contains unique identifiers for this table.

When inserting data into a table containing this line, the `id` column does not need to be included in the insert statement because it automatically increments with each row.

Wildcards

Wildcards are used to substitute from zero to many characters in a string when performing queries. The keyword `LIKE` indicates the use of a wildcard.

`%` (percentage):

```
// Substitute zero to multiple characters in a
query
SELECT *
FROM actor
WHERE last_name
LIKE 'Will%';
```

In this example, all last names beginning with "Will" will be returned, including **Will**, **Willa**, and **Willows**.

`_` (underscore):

```
// Substitute a single character in a query
SELECT *
FROM actor
WHERE first_name
LIKE '_AN';
```

In the above code block, the underscore represents a single character. After executing this query, all actors whose first name contains three letters, the second and third of which are "AN", will be returned.