SQL II

By Richard Roggenkemper and Shide Dehghani

Review of Tables and Join

Table

A table stores data. It consists of...

- a fixed number of **columns**.
- data entries stored in **rows**.

To make a table in SQL, use a **CREATE TABLE** statement:

```
CREATE TABLE [name] AS ...;
```

To create rows of data, **UNION** together **SELECT** statements:

```
SELECT [expr] AS [name], [expr] AS [name], ... UNION
SELECT [expr] AS [name], [expr] AS [name], ... UNION
SELECT [expr] AS [name], [expr] AS [name], ...;
```

To create rows of data from existing tables, use a **SELECT** statement with a **FROM** clause:

```
SELECT [columns] FROM [table] WHERE [condition]
    ORDER BY [order] [ASC/DESC] LIMIT [number];
```

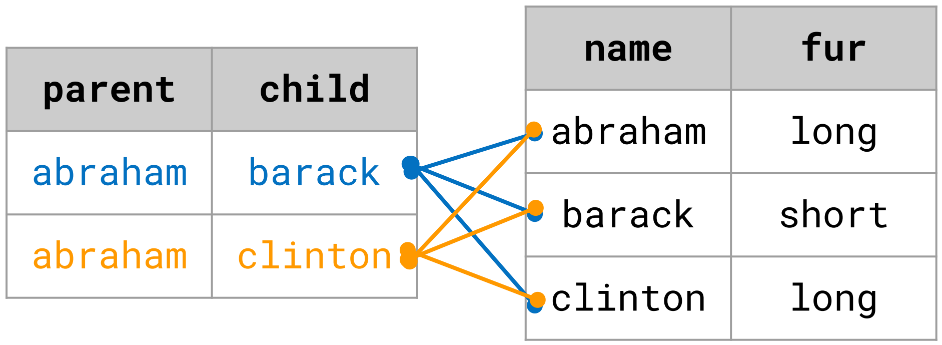
Join

Demo_1

Given multiple tables, we can join them together by specifying their names, separated by commas, in the **FROM** clause of a **SELECT** statement.

```
SELECT * FROM table1, table2;
```

When we join two tables, we get a new table with one row for each combination of rows from the original tables.



parent	child
abraham	barack
abraham	clinton

name	fur
abraham	long
barack	short
clinton	long

parent	child	name	fur
abraham	barack	abraham	long
abraham	barack	barack	short
abraham	barack	clinton	long
abraham	clinton	abraham	long
abraham	clinton	barack	short
abraham	clinton	clinton	long

Check Your Understanding

Table songs:
`name | artist | album`

Table albums:
`name | artist | release_year`

Table artists:
`name | first_year_active`

1. Write an SQL query that outputs the first 10 artists who became active after 2015.

```
SELECT name FROM artists
   WHERE first_year_active > 2015 LIMIT 10;
```

1. Write an SQL query that outputs the names and artists of songs that were released in 2010 ordered by the first year active of the artist.

```
SELECT s.name, s.artist
    FROM songs AS s, artists AS ar, albums AS al
    WHERE album = al.name AND s.artist = ar.name
           AND release_year = 2010
           ORDER BY first_year_active;
```

Aggregation

Single Row Operations: Single-Table Queries

So far, our SQL statements have referred to the values in a single row at a time.

table **dogs**

name	fur
abraham	long
barack	short
clinton	long
delano	long
eisenhower	short
fillmore	curly
grover	short
herbert	curly

Write a query that outputs the name of dogs that either have long fur or are named Grover.

```
SELECT name FROM dogs
    WHERE fur = 'long' OR name = 'grover';
```

output:

name
abraham
clinton
delano
grover

Single Row Operations: Join

table **dogs**

name	fur
delano	long
herbert	curly
grover	short

table **parents**

parent	child
delano	herbert
fillmore	delano
fillmore	grover

Write a query that outputs the names and fur types of all of Fillmore's children.

```
SELECT name, fur FROM dogs, parents
    WHERE parent = 'fillmore' AND
                    name = child;
```

output:

name	fur
delano	long
grover	short

result of cross product:

name	fur	parent	child
delano	long	delano	herbert
delano	long	fillmore	delano
delano	long	fillmore	grover
herbert	curly	delano	herbert
herbert	curly	fillmore	delano
herbert	curly	fillmore	grover
grover	short	delano	herbert
grover	short	fillmore	delano
grover	short	fillmore	grover

Aggregation

Aggregation is the process of doing operations on *groups of rows* instead of just a single row.

SQL provides **aggregate functions** whose return values can be used as entries in a column.

table **dogs**

name	fur	age
delano	long	10
eisenhower	short	7
fillmore	curly	8
grover	short	2
herbert	curly	4

output the average age of all dogs:

```
SELECT AVG(age) AS avg_age FROM dogs;
```

output:

avg_age
6.2

output the total number of rows:

```
SELECT COUNT(*) AS count FROM dogs;
```

output:

count
5

Aggregate Function

Aggregation function	Return value
`MAX([columns])`	The maximum value in the given column(s)
`MIN([columns])`	The minimum value in the given column(s)
`AVG([column])`	The average value in the given column
`COUNT([column])`	The number of values in the given column
`SUM([column])`	The sum of the values in the given column

table **dogs**

name	fur	age
eisenhower	short	7
delano	long	10
grover	short	2

output the sum of ages of all dogs:

```
SELECT SUM(age) AS sum_age FROM dogs;
```

output the name that comes first alphabetically:

```
SELECT MIN(name) AS min_name FROM dogs;
```

Groups

By default, aggregation is performed over all the rows of the table.

We can specify that we want to group rows based on values in a particular column using the **GROUP BY** clause in a **SELECT** statement.

table **dogs**

name	fur	Age
eisenhower	short	7
delano	long	10
grover	short	2
fillmore	curly	8
herbert	curly	4

Write a query that finds the average age of dogs for each fur type.

```
SELECT fur, AVG(age) AS avg_age
    FROM dogs GROUP BY fur;
```

output:

fur	avg_age
short	4.5
long	10
curly	6

More on Group By

Demo_2

You can GROUP BY any valid SQL expression, which includes using multiple column names and operators.

```
SELECT [columns] FROM [table] WHERE [condition]
    GROUP BY [expression]
    ORDER BY [order] [ASC/DESC]
    LIMIT [number];
```

A single group consists of all rows for which [expression] evaluates to the same value.

The output table will have **one row** per group.

Check Your Understanding

table **dogs**

name	**fur**	**age**
abraham	long	9
herbert	curly	4
fillmore	curly	8
delano	long	10
eisenhower	short	3

table **parents**

parent	**child**
delano	herbert
fillmore	abraham
fillmore	delano
eisenhower	fillmore

1. Write a query that outputs a table containing the average age of each parent's children.

parent	**avg_age**
delano	4
fillmore	9.5
eisenhower	8

1. Write a query that outputs a table with 2 rows: one with the number of dogs of even ages and the other with the number of dogs of odd ages (ignore order).

Remember that you can **GROUP BY** expressions containing operators!

count
3
2

Check Your Understanding

table **dogs**

name	fur	age
abraham	long	9
herbert	curly	4
fillmore	curly	8
delano	long	10
eisenhower	short	3

table **parents**

parent	child
delano	herbert
fillmore	abraham
fillmore	delano
eisenhower	fillmore

1. Write a query that outputs a table containing the average age of each parent's children.

```
SELECT parent, AVG(age) AS avg_age
   FROM dogs, parents
   WHERE child = name
   GROUP BY parent;
```

1. Write a query that outputs a table with 2 rows: one with the number of dogs of even ages and the other with the number of dogs of odd ages (ignore order).

```
SELECT COUNT(*) AS count
  FROM dogs
  GROUP BY age % 2 = 0;
```

Filtering Groups

We know how to filter individual rows using the `WHERE` clause.

To filter groups, use the `HAVING [condition]` clause!

Write a query that finds the average age of dogs for each fur type if there are more than one dogs with that fur type.

table **dogs**

name	fur	age
abraham	long	9
herbert	curly	4
fillmore	curly	8
delano	long	10
eisenhowe r	short	3

```
SELECT fur, AVG(age) AS avg_age
    FROM dogs GROUP BY fur
    HAVING COUNT(*) > 1;
```

output:

fur	avg_age
long	9.5
curly	6

```
SELECT fur, AVG(age) AS avg_age
    FROM dogs GROUP BY fur
    HAVING COUNT(*) > 1;
```

table **dogs**

name	fur	age
abraham	long	9
eisenhower	short	3
delano	long	10
fillmore	curly	8
herbert	curly	4

eisenhower	short	3

abraham	long	9
delano	long	10

fillmore	curly	8
herbert	curly	4

fur	avg_age
long	9.5
curly	6

Check Your Understanding

table **dogs**

name	fur	age
abraham	long	9
herbert	curly	3
fillmore	curly	8
delano	long	10
eisenhower	short	3

table **parents**

parent	child
delano	herbert
fillmore	abraham
fillmore	delano
eisenhower	fillmore

Write a query that outputs the average age of each parent's children if that parent's youngest child is at least 5.

parent	avg_age
fillmore	9.5
eisenhower	8

```
SELECT parent, AVG(age) AS avg_age
    FROM dogs, parents
    WHERE name = child
    GROUP BY parent
    HAVING MIN(age) >= 5;
```

Mutating Tables

Databases

In real databases, it’s common practice to initialize empty tables and add rows as new data is introduced.

Demo_3

Create/ remove tables

To create an empty table, use the **CREATE TABLE** statement, specifying the table name and column names (and possible default values):

```
CREATE TABLE [name]([columns]);
```

```
CREATE TABLE parents(parent, child);
```

When adding rows, if no value is provided for the third column, this value will be used.

```
CREATE TABLE dogs(name, fur, phrase DEFAULT 'woof');
```

To remove a table from our database, use the **DROP TABLE** statement:

```
DROP TABLE [IF EXISTS] [name];
```

```
DROP TABLE dogs;
```

```
DROP TABLE IF EXISTS parents;
```

Inserting Records

To insert rows into a table:

```sql
INSERT INTO [table]([columns]) VALUES([values]), ([values]);
```

```sql
CREATE TABLE dogs(name, fur, phrase DEFAULT 'woof');
```

name	fur	phrase
fillmore	curly	woof
delano	long	hi!
	short	bark

```sql
INSERT INTO dogs(name, fur) VALUES('fillmore', 'curly');
INSERT INTO dogs VALUES('delano', 'long', 'hi!');
INSERT INTO dogs(fur, phrase) VALUES('curly', 'bark');
```

Updating Records

To update existing entries in a table:

```
UPDATE [table] SET [column] = [expression] WHERE [condition];
```

To delete existing rows in a table:

```
DELETE FROM [table] WHERE [condition];
```

name	fur	phrase
fillmore	curly	WOOF
delano	short	hi!
	short	bark

```
UPDATE dogs SET phrase = 'WOOF' WHERE fur = 'curly';
DELETE FROM dogs WHERE fur = 'curly' and phrase = 'WOOF';
UPDATE dogs SET fur = 'short';
```

Summary

Create empty table

```
CREATE TABLE [name]([columns]);
```

- **using default values:**

```
CREATE TABLE [name](...,[column n] DEFAULT [value], ...);
```

Remove table from database

```
DROP TABLE [IF EXISTS] [name];
```

Inserting records (new row):

```
INSERT INTO [table]([columns]) VALUES([values]),
([values]);
```

```
INSERT INTO [table] VALUES(...,[values (one for each column)], ...);
```

Updating records (existing row):

```
UPDATE [table] SET [column] = [expression] WHERE [condition];
```

```
DELETE FROM [table] WHERE [condition];
```

Summary

We can use **aggregate functions** to perform operations on a set of rows rather than on individual rows.

To specify an expression by which to group rows, use the **`GROUP BY`** clause.

To filter groups based on a condition over the whole group, use the **`HAVING`** clause.

In real databases, we commonly initialize empty tables and insert, update, or remove records over time.