CS 61A DISCUSSION 10

SQL

Raymond Chan
Discussion 134
UC Berkeley Fall 16

AGENDA

- Announcements
- SQL
- Joins
- Recursive Queries
- Appendix
  - Aggregation

ANNOUNCEMENTS

- Scheme Project due tonight.
- Homework 12 due 11/23 (next Wed).
- Lab 12 due Friday.
- Guerrilla Section Saturday.

SQL

- Declarative programming: tells the interpreter **what** we want.
- Describe the result, not the behavior.
- Data in SQL are stored in tables with a fixed number of named columns.
- Each row represent a single data record with a value in each column.

SQL

- We use **select** statements to create tables.
- Each **select** creates a new row.
- A row by itself is considered a table.

```
> select "Ben" as first, "Bitdiddle" as last;
Ben | Bitdiddle
```

SQL

- Multiple tables can have the same number of columns.
- We can combine the rows of the tables with **union**, creating a larger table.
- Column headings do not have to be repeated.

```
> select "Ben" as first, "Bitdiddle" as last union
> select "Louis",               "Reasoner";
Ben | Bitdiddle
Louis | Reasoner
```

SQL

- To save newly created tables, we use **create table.**
- **create table** [table name] **as** [select statements]

```
CREATE TABLE records AS
  SELECT "Ben Bitdiddle" AS name, "Computer" AS division, "Wizard" AS
           title, 60000 AS salary, "Oliver Warbucks" AS supervisor UNION
  SELECT "Alyssa P Hacker", "Computer", "Programmer", 40000,
                                                         “Ben Bitdiddle" UNION
  SELECT …
```

SQL

- We can now make queries to the table.
- **select *** means select all from table.
- **select * from** records;
    - Prints out the contents of the table.

SQL

select [column1], [column2], … **from** [table_name]

where [condition] **order by** [criteria] **limit** [number of entries]

- There must be at least 1 column and a table to select from.
- Everything else is optional.
- [condition] is <u>one</u> conditional expression.

SQL

- SQL expressions.
  - Comparators: =, >, <, <=, >=, !=, <> ("not equal")
  - Booleans: **and**, **or**
  - Arithmetic: +, -, *, /
- We use || to concatenate strings.

```
> select "hello" || "world"
hello world
```

JOINS

select [column1], [column2], … **from** [table1], [table2] …

where [condition] **order by** [criteria]

- Data can be combined by joining multiple tables together.
- The result table contains a new row for each combination of rows in the input tables.

JOINS

Table_1

A
B
C
D
…

select … from Table_1, Table_2

Table_2

1
2
3
4
…

A	1
A	2
A	3
A	…
B	1
B	2
B	…
…	…

JOINS

Table_1 *m* rows

A
B
C
D
…

select … **from** Table_1, Table_2

Table_2 *n* rows

1
2
3
4
…

mn rows

A	1
A	2
A	3
A	…
B	1
B	2
B	…
…	…

JOINS

- Notice that there are “duplicates” because we have filtered out the rest of the data for the rows.

```
> select name, day from records, meetings;
Ben Bitdiddle | Monday
Ben Bitdiddle | Wednesday
Ben Bitdiddle | Monday
...
Alyssa P Hacker | Monday
...
```

Division	Day	Time
Accounting	Monday	9am
Computer	Wednesday	4pm
Adminstration	Monday	11am
Administration	Thursday	1pm

JOINS

- Notice that there are "duplicates" because we have filtered out the rest of the data for the rows.
- Adding another column back in…

```
> select name, day, division from records, meetings;
Ben Bitdiddle | Monday | Accounting
Ben Bitdiddle | Wednesday | Computer
Ben Bitdiddle | Monday | Administration
...
Alyssa P Hacker | Monday
...
```

Division	Day	Time
Accounting	Monday	9am
Computer	Wednesday	4pm
Adminstration	Monday	11am
Administration	Thursday	1pm

JOINS

- Tables can have the same column names.
- Tables can also be joined with themselves.
- To distinguish between columns, we give *aliases* to tables in the **from** clause.
- To refer to a specific table's column, we use dot notation.

JOINS

select [some_alias].[column1], [some_alias].[column2], …

from [table1] **as** [alias1], [table2] **as** [alias2] …

where [condition] **order by** [criteria]

JOINS

select [some_alias].[column1], [some_alias].[column2], …

  from [table1] **as** [alias1], [table2] **as** [alias2] …

  where [condition] **order by** [criteria]

Filling in what tables you want to select from and the filter condition before thinking about the columns you want.

Goal is to obtain the correct information and then outputting the relevant information

JOINS

```
> select b.name, b.title from records as a, records as b
...     where a.name = "Louis Reasoner" and
...       a.supervisor = b.name;
Alyssa P Hacker | Programm
```

RECURSIVE QUERIES

- We can create local tables using the **with** clause.
- They cannot be used outside of the select statement.
- Can be thought of as “helper” tables.
- Use the local tables to compute the final result.

RECURSIVE QUERIES

with [local-tables] **select** [columns] **from** [tables]

where [condition] **order by** [criteria]

RECURSIVE QUERIES

with [local-tables] **select** [columns] **from** [tables]

where [condition] **order by** [criteria]

with [local-table-name] **as** (

select … <row 1>… **union**

select … <row 2> … **union**

…

)

select [columns] **from** [tables] **where** [condition] **order by** [criteria]

RECURSIVE QUERIES

```
WITH schedule(day, dresscode) as (
  SELECT "Monday", "Sports" UNION
  SELECT "Tuesday", "Drag" UNION
  SELECT "Wednesday", "Regular" UNION
  SELECT "Thursday", "Throwback" UNION
  SELECT "Friday", "Casual"
  )
SELECT a.name, b.dresscode
  from records as a, schedule as b, meetings as c
  where a.division = c.division and
  b.day = c.day order by a.name;
```

RECURSIVE QUERIES

```
WITH schedule(day, dresscode) as (
  SELECT "Monday", "Sports" UNION
  SELECT "Tuesday", "Drag" UNION
  SELECT "Wednesday", "Regular" UNION
  SELECT "Thursday", "Throwback" UNION
  SELECT "Friday", "Casual"
  )
SELECT a.name, b.dresscode
  from records as a, schedule as b, meetings as c
  where a.division = c.division and
  b.day = c.day order by a.name;

Alyssa P Hacker | Regular
Ben Bitdiddle | Regular
Cy D Fect | Regular
DeWitt Aull | Sports
...

> select * from schedule;
Error
```

RECURSIVE QUERIES

- Using the **with** clause, we can create recursive tables.
- The local table has base case(s) and recursive case(s).

```
with [local-table-name] as (
  select … <base case(s)> … union
  select … <recursive case(s)> …
)
select [columns] from [tables] where [condition] order by [criteria]
```

RECURSIVE QUERIES

- Using the **with** clause, we can create recursive tables.
- The local table has base case(s) and recursive case(s).

```
create table naturals as
  with num(n) as (
    select 0 union
    select n + 1 from num where n < 5
    )
  select * from num;
```

RECURSIVE QUERIES

- The initial table initially has a column with 1 row and value of 0.
- In the recursive case we add 1 to a value of the table entries that has *not* been used before.

```
create table naturals as
  with num(n) as (
    select 0 union
    select n + 1 from num where n < 5
    )
  select * from num;
```

RECURSIVE QUERIES

- The condition that *stops* the recursive occurs in the **where** clause of the recursive case.

```
create table naturals as
  with num(n) as (
    select 0 union
    select n + 1 from num where n < 5
    )
  select * from num;
```

RECAP

- In SQL we tell the interpreter what we want.
- Tables are created with select statements that can filter information.
- We can join tables and use alias to distinguish column names.
- Recursive queries can be created when using local tables.
- Aggregation looks at multiple entries of the table. (Appendix; Will be covered in Friday's lecture)

AGGREGATION

APPENDIX - SP'16

- Aggregation operations are performed over multiple rows.
- **min, max, average, sum, count**
- They all take in 1 argument: a column name or *
- These functions retrieve more information from initial tables.

AGGREGATION

APPENDIX - SP’16

- Find name and salary of the person that makes the most money.

```
> select name, max(salary) from records;
Oliver Warbucks | 150000
```

AGGREGATION

APPENDIX - SP'16

- We can count the number of rows to determine the number of employees.

```
> select count(*) from records;
9
```

AGGREGATION

APPENDIX - SP’16

- Aggregation can be performed on specific sets of rows.
- **group by** [column name] groups all the rows that have the same value in column name.

AGGREGATION

APPENDIX - SP'16

- Find the minimum salary earned in each division of the company.

```
> select division, min(salary)
...       from records group by division;
Computer | 25000
Administration | 25000
Accounting | 18000
```

AGGREGATION

APPENDIX - SP'16

- Groups can be filtered by the **having** clause.
- This is similar to the **where** clause.

AGGREGATION

APPENDIX - SP’16

- Find all titles that are held by more than one person

```
> select title from records
...      group by title having count(*) > 1;
Programmer
```

AGGREGATION

APPENDIX - SP'16

- Aliases can also be used with aggregation results

```
> select title, count(*) as count from records
...       group by title having count > 1;
Programmer
```