Relational Databases

Hye-Chung Kum (kum@tamu.edu)
Associate Professor
Population Informatics Lab (https://pinformatics.org/)
Course URL: http://pinformatics.org/phpm631
License: Health Information Technology by Hye-Chung Kum is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License

Outline

- What is a database?
- What is a database management system?
- An Introduction to SQL
  - How to retrieve data from a database
  - How to create a database (optional)

What Tables?

- Patient Unique ID
- Visit Unique ID
- Medication Unique ID
- Provider Unique ID

SQL - Structured Query Language

- Every statement yields a table of values as output
  - Sometimes there's only one row in the table!
  - semicolon: Don't Forget.
  - Tells the computer I am done writing my statement

<table>
<thead>
<tr>
<th>Keyword</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>lname, fname</td>
</tr>
<tr>
<td>from</td>
<td>patients</td>
</tr>
<tr>
<td>where</td>
<td>gender='F'</td>
</tr>
<tr>
<td>group by</td>
<td>group rows together</td>
</tr>
<tr>
<td>order by</td>
<td>lname, fname</td>
</tr>
</tbody>
</table>

Choose Columns

- Choosing a subset of columns is sometimes called a “project” operation
- Display first and last name of all patients
  - SELECT name, lname FROM patients;
- TRY: Display diagnosis and visitDate for all patients

Display an Entire Table

- Wildcard
  - * : means all columns
- Regular Expression

SELECT * FROM patients;
Choose Rows

- Find Hispanic patients
  - SELECT `fname`, `lname`, race
  - FROM patients
  - WHERE race = "H";
- TRY: Find visits before Feb 1, 2012
  - YYYY-MM-DD: standard SQL (e.g., '2012-02-01')

Choose Rows: conditional statements

- String: WHERE race = "Hispanic";
- String: WHERE `fname` LIKE "s%";
- Number: WHERE age > 18;
- Date: WHERE `svc_dt` > '1993-01-01';
- YYYY-MM-DD: standard SQL
- Find patients born after Jan. 1, 1993

Conditional Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal. Note: In some versions of SQL this operator may be written as !=</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>Between an inclusive range</td>
</tr>
<tr>
<td>LIKE</td>
<td>Search for a pattern (approximately same)</td>
</tr>
<tr>
<td>IN</td>
<td>To specify multiple possible values for a column</td>
</tr>
</tbody>
</table>

Conditional Logic: Truth Tables

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>NOT</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Try on w3schools

- Which is the correct SQL statement below?
- What is the problem with the incorrect SQL statements?
  - SELECT * FROM patients where race = H;
  - SELECT * FROM patients where race = 'H';
  - SELECT * FROM patients where race = 'h';
- You may also try the free trials at
  - https://academy.vertabelo.com

Sorting

- Can sort output by contents of a column
  - sort in ascending or descending order
  - sort by more than one column (second one breaks ties)
- Sort patients by last name
  - SELECT * FROM patients
  - ORDER BY `lname` DESC;
- TRY: What are our 10 most costly visits?
Take Away I
SQL – Structured Query Language

- Every statement yields a table of values as output
  - Sometimes there’s only one row in the table!

<table>
<thead>
<tr>
<th>Keyword</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>columns and/or expressions</td>
</tr>
<tr>
<td>from</td>
<td>Tables</td>
</tr>
<tr>
<td>where</td>
<td>conditions on the rows</td>
</tr>
<tr>
<td>group by</td>
<td>group rows together</td>
</tr>
<tr>
<td>order by</td>
<td>order the rows</td>
</tr>
</tbody>
</table>

Take Away II
Boolean Logic: Truth Tables (1=T; 0=F)

- WHERE gender='Male' AND age>18;
- WHERE race='Hispanic' OR race='Black';

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>NOT</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Compute Columns

- Find discount amount
  - SELECT patientID, billed, covered, (billed-covered) AS discount
  - FROM payments;
- Nice names for output columns
  - Name following computed column (e.g., discount) will be used to name output column
- Find total paid amount
  - Total = copay+pat_pd+insur_pd

Aggregates (multiple rows)

- Can make calculations on multiple rows
  - sum, avg, max, min, count
- How many charges by bill type
  - SELECT count(charge) as ncharges
  - FROM charges
  - GROUP BY billtype
- TRY: What is total billed by patient?

GROUP BY Column1;

- (Column1) AS ColumnName
  - FROM Table;

Grouping and Aggregates

- Can make calculations on groups of rows
  - sum, avg, max, min, count
- Each different value for the GROUP BY fields defines a new group
  - One row of output is produced for each group
  - Several rows of input table may belong to same group. They are aggregated using aggregation operator.
### Table Operations

- **Aggregate columns:** `col1 op col2 AS col3`

```
<table>
<thead>
<tr>
<th>col1</th>
<th>col2</th>
<th>col3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>d</td>
<td>a+d</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
<td>b+e</td>
</tr>
<tr>
<td>c</td>
<td>f</td>
<td>c+f</td>
</tr>
</tbody>
</table>
```

- **Aggregate rows:** `GROUP BY` 

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a+b</td>
<td>c+f</td>
<td>D</td>
</tr>
</tbody>
</table>
```

Where `D=Sum(A,B,C)`
Examples of function are:
- `Sum(A,B,C)`
- `Avg(A,B,C)`
- `Max(A,B,C)`
- `Min(A,B,C)`
- `Count(A,B,C)`

### Joins

- Combine rows from one table with rows from another
- Usually join on some common column
  - Don’t combine rows unless their value in the common column is the same
- WHERE clause says the common column must be same in each table
- Produce a list of bills for all patients with their name
  - SELECT `patients.fname, patients lname, patients.patientID`, `billed` 
  - FROM `patients`, `payments` 
  - WHERE `patients.patientID = payments.patientID`

#### Different Syntax: Joins

- SQL JOINs
  - `INNER JOIN: Returns all rows when there is at least one match in BOTH tables`
  - `LEFT JOIN: Return all rows from the left table, and the matched rows from the right table`
  - `RIGHT JOIN: Return all rows from the right table, and the matched rows from the left table`
  - `FULL JOIN: Return all rows when there is a match in ONE of the tables`

### Views: Permanent Queries

- Looks and feels like a table
- Saved queries
- Virtual table: not a real table in the DB
- Can treat it like a real table, as if it exists

#### Create View

```
CREATE VIEW panel as
SELECT
    providers.fname as dr_first, providers.lname as dr_last,
    patients.fname, patients.lname
FROM providers, patients
WHERE providers.providerID=patients.primary_dr;
```
Resources

- http://www.w3schools.com/sql/default.asp
- https://academy.vertabelo.com
- https://www.sqlite.org/lang.html
- SQLite DB Browser: portable apps
  - http://portableapps.com/apps/development/sqlite_database_browser_portable