Outline

- What is a database?
- What is a database management system?
- An Introduction to SQL
  - How to retrieve data from a database I
    - Basic SQL Queries
  - How to retrieve data from a database II
    - Calculations using multiple columns or rows
  - How to retrieve data from a database III
    - Combining tables
  - How to create a database (optional)
Table Operations

• Aggregate columns: \( \text{col1 op col2 AS col3} \)

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

• Aggregate rows: \( \text{Group BY} \)

Where \( D = \text{function}(A,B,C) \)
Examples of function are
\( \text{Sum}(A,B,C) \) \( \text{Avg}(A,B,C) \) \( \text{Max}(A,B,C) \) \( \text{Min}(A,B,C) \) \( \text{Count}(A,B,C) \)
Compute Columns

- Find discount amount
  - SELECT patientID, (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
- String vars: concatenate "||"
  - fname ||" "|| lname AS name
- Find total paid amount (payments Table)
  - Total = copay+pat_pd+insur_pd

```
SELECT calculate AS NewColumnName
FROM Table;
```
Grouping and Aggregates I

- 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.

```
SELECT f(Column2) AS ColumnName
FROM Table
GROUP BY Column1;
--f(x) : sum, avg, max, min, count;
```
Grouping and Aggregates II

- Can make calculations on groups of rows
  - sum, avg, max, min, count
- How many visits did each patient have?
  - SELECT patientID, count(visitID) AS nvisits
  - FROM visits
  - GROUP BY patientID;
- TRY: What is total billed by patient?
  - Payments table

```sql
SELECT f(Column2) AS ColumnName
FROM Table
GROUP BY Column1;
-- f(x) : sum, avg, max, min, count ;
```
Take Away II

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 (AS)</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>
<tr>
<td>;</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- What is a database?
- What is a database management system?
- An Introduction to SQL
  - How to retrieve data from a database I
    - Basic SQL Queries
  - How to retrieve data from a database II
    - Calculations using multiple columns or rows
  - How to retrieve data from a database III
    - Combining tables: Joins
  - How to create a database (optional)
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
- Give discount amount by last name of patient
  - SELECT patientID, patients.lname, (billed-covered) AS discount
  - FROM payments, patients
  - Where payments.patientID=patients.patientID;
- Give primary diagnosis (diag1) description for each patient visit
  - Tables Visits & lu_diag

```
SELECT Table1.Column1, Table2.Column2
FROM Table1, Table2
WHERE Table1.Column=Table2.Column;
```
Practice Problems

- Give primary diagnosis (diag1) description for each patient visit
  - Tables Visits & lu_diag
- P1: for only patients that saw Dr. Gaines
- P2: for only patients seen in dec 2012
- P3: for only patients seen in dec 2012 by Dr. Gaines
- P4: for only patients that saw Dr. Gaines or Dr. Fry
Different Syntax: Joins

```
SELECT Table1.Column1, Table2.Column2
FROM Table1, Table2
WHERE Table1.Column=Table2.Column;

-- Does same thing;
SELECT Table1.Column1, Table2.Column2
FROM Table1
JOIN Table2
ON Table1.Column=Table2.Column;
```
Different 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 panel as
SELECT
    providers.fname AS dr_first,
    providers.lname AS dr_last,
    patients.fname,
    patients.lname,
    patientID
FROM providers, patients
WHERE
    providers.providerID=patients.primary_dr
ORDER BY
    providers.providerID;
Practice Problems

- How many patients does each doctor have?
  - Hint: use the view you just created
- How did you identify the doctor?
  - Last name? first name?
  - What if there are two doctors with the same name?
  - What is the BEST way to refer to entities in a RDB?
Create (Optional)

- Primary Key
- Data Type
  - Text, Integer, Real
  - Data types might have different names in different database. And even if the name is the same, the size and other details may be different! *Always check the documentation!*
  - [http://www.w3schools.com/sql/sql_datatypes_general.asp](http://www.w3schools.com/sql/sql_datatypes_general.asp)
- Example: kumdb.sql
### Advanced conditionals (Optional)

#### Like & Wildcard

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%</code></td>
<td>A substitute for zero or more characters</td>
</tr>
<tr>
<td><code>_</code></td>
<td>A substitute for a single character</td>
</tr>
<tr>
<td><code>[charlist]</code></td>
<td>Sets and ranges of characters to match</td>
</tr>
<tr>
<td><code>[^charlist]</code> or <code>[^charlist]</code></td>
<td>Matches only a character NOT specified within the brackets</td>
</tr>
</tbody>
</table>

```sql
SELECT.fname, lname
FROM patients
WHERE lname LIKE 's%';
```
Indexing

- Can have many per table
- Mapping to locations in storage, for quick lookup
  - Example: index patientID
    - PID=1 in memory 04
    - PID=2 in memory 08 ...etc...
- Pros: Faster to find and retrieve data
- Cons: Slow to enter and save data
ACID: Atomic, Consistent, Isolated & Durable

- four most desirable basic characteristics of a transaction system
- **Atomic** transactions are such that the transaction is either entirely completed or makes no change to the database; even if an error or a hardware fault occurs mid-transaction the database will not be left with a half-completed transaction.
- **Consistent** transactions ensure that the database is left in a consistent state after the transaction is complete, meaning that any integrity constraints (unique keys, foreign keys, CHECK constraints etc.) must be satisfied or the transaction will be rejected.
- **Isolated** transactions are invisible to other users of the database while they are being processed
- **Durable** transactions guarantee that they will not be rolled back after the caller has committed them
- Early RDBMSes couldn't always guarantee all four of these requirements with their transactions, but modern counterparts can usually provide ACID transactions even in the event of power or hardware failure.
Take Away I
Advanced SQL queries

• Why write SQL queries?
  ◦ To answer real world questions from the database

• Calculating new columns from other columns

• Combining multiple rows
  ◦ Grouping and aggregating

• Views: Permanent Queries
  ◦ CREATE VIEW
Take Away II
Different 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
Indexing

- Can have many per table
- Mapping to locations in storage, for quick lookup
  - Example: index patientID
    - PID=1 in memory 04
    - PID=2 in memory 08 ...etc...
- Pros: Faster to find and retrieve data
- Cons: Slow to enter and save data

four most desirable characteristics of a transaction system

- ACID: Atomic, Consistent, Isolated & Durable