# Relational Databases II



**Course URL:**
http://pinformatics.org/phpm631

# 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: col1 op col2 AS col3

| col1 | col2 |
|---|---|
| a | d |
| b | e |
| c | f |

→

| col1 | col2 | col3 |
|---|---|---|
| a | d | a+d |
| b | e | b+e |
| c | f | c+f |

- Aggregate rows: Group BY

| A |
|---|
| B |
| C |

→

| D |
|---|

Where D=function(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)

# 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

```
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!

| Keyword | parameters |
|---|---|
| **select** | columns and/or expressions (**AS**) |
| **from** | Tables |
| **where** | conditions on the rows |
| **group by** | group rows together |
| **order by** | order the rows |
| **;** | |

# 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

```
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)

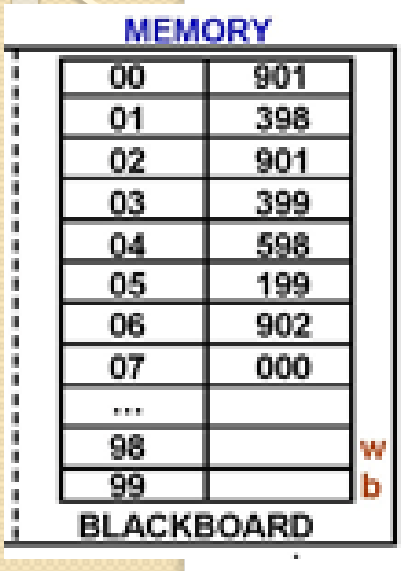
- Primary Key
- Data Type
  - Text, Integer, Real
  - Data types might have different names in different database. And even if the name is the same, the size and other details may be different! **Always check the documentation!**
  - http://www.w3schools.com/sql/sql_datatypes_general.asp
- Example: kumdb.sql

# Advanced conditionals (Optional) Like & Wildcard

| Wildcard | Description |
|---|---|
| % | A substitute for zero or more characters |
| _ | A substitute for a single character |
| [charlist] | Sets and ranges of characters to match |
| [^charlist] or [!charlist] | Matches only a character NOT specified within the brackets |

```
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.

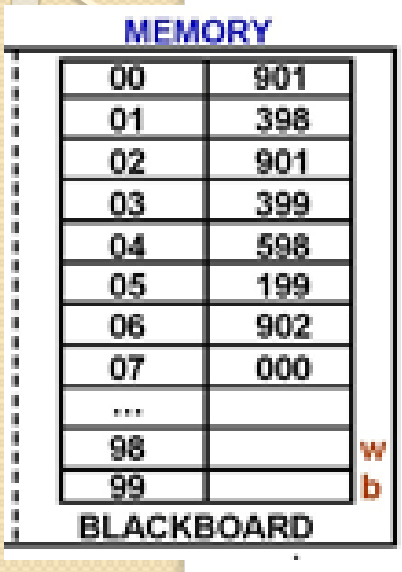
# Take Away I
# Advanced SQL queries

- 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

# Take Away III




- 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