Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
    - Insert
    - Delete
    - Update
    - Select
      - from, where
      - order by
      - set operations
      - joins (implicit, explicit)
      - sub-queries
      - aggregation and grouping
Aggregation and Grouping

- Open sqlite3, and create the following table:

```sql
create table Employees(
    ENO INTEGER PRIMARY KEY,
    Dept TEXT,
    Country TEXT,
    Name TEXT,
    Wage INTEGER
);
insert into Employees values (0, 'A', 'US', 'John', 50);
insert into Employees values (1, 'A', 'China', 'Lynn', 75);
insert into Employees values (3, 'B', 'US', 'Ross', 60);
insert into Employees values (7, 'C', 'US', 'Julia', 95);
insert into Employees values (8, 'B', 'China', 'David', 25);
insert into Employees values (9, 'A', 'China', 'Ned', 65);
insert into Employees values (NULL, 'A', 'US', 'John', 50);
insert into Employees values (NULL, 'B', 'China', 'John', 30);
insert into Employees values (NULL, 'C', 'China', 'David', 30);
insert into Employees values (NULL, 'A', 'US', 'David', 50);
insert into Employees values (NULL, 'C', 'US', 'Sara', 45);
```
Aggregation and Grouping

- Recall the R.A. Syntax \[ g_1, \ldots, g_j \bigcup f_1(a_1), \ldots, f_k(a_k)(R) \]

### Employees

<table>
<thead>
<tr>
<th>ENO</th>
<th>Dept</th>
<th>Country</th>
<th>Name</th>
<th>Wage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>US</td>
<td>John</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>China</td>
<td>Lynn</td>
<td>75</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>US</td>
<td>Ross</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>US</td>
<td>Julia</td>
<td>95</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
<td>China</td>
<td>David</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>China</td>
<td>Ned</td>
<td>65</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>US</td>
<td>John</td>
<td>50</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>China</td>
<td>John</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>China</td>
<td>David</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>A</td>
<td>US</td>
<td>David</td>
<td>50</td>
</tr>
<tr>
<td>14</td>
<td>C</td>
<td>US</td>
<td>Sara</td>
<td>45</td>
</tr>
</tbody>
</table>

- **Q1**: Get the total number of employees, the min wage, and max wage
- **Q2**: Get the number of employees, min wage, and max wage by department and country.
Aggregation and Grouping in SQL

- **Syntax:**
  ```sql
  SELECT f1(DISTINCT a_1), ..., fk(DISTINCT a_k), 
  g_1, ..., g_j, 
  a_1, ..., a_n 
  FROM ... 
  WHERE ... 
  GROUP BY g_1, g_2, ..., g_j;
  ```

- **Q1:** Get the total number of employees, the min wage, and max wage
  ```sql
  select count(*) as total, max(Wage), min(Wage) from employees;
  ```
Aggregation and Grouping

- Q2: Get the number of employees, min wage, and max wage by department and country.

  ```sql
  select dept, country, count(*) as total, max(Wage), min(Wage)
  from employees
  group by dept, country;
  ```

- Do I need to project dept and country? Try this:

  ```sql
  select count(*) as total, max(Wage), min(Wage)
  from employees
  group by dept, country;
  ```

- What if GROUP BY attributes were swapped in order?
Q3: Find all countries whose average wage is < $60.

Why won't the following work?

```sql
select *
from (select country, avg(wage)
     from employees
     group by country)
where avg(wage) < 60;
```
Aggregation and Grouping (Cont.)

- Reminder: aggregated columns have no label

- **Q3:** Find all countries whose average wage is < $60.

```sql
select *
from (select country, avg(wage) as AVGWAGE
     from employees
     group by country)
where AVGWAGE < 60;
```
Q4: Count the number of passengers with the same last name. Sort results in descending order of the count.

Solution:

```sql
SELECT name, count(*) AS num
FROM Employees
GROUP BY name
ORDER BY num DESC;
```
Q4b: Now get last names that occur the *least* frequently.

Solution?? (FIX ME!!!)

WITH tmp AS ( -- tmp stores the previous query results to get counts
    SELECT name, count(*) AS num
    FROM Employees
    GROUP BY name)
SELECT name, min(num)
FROM tmp;
WITH tmp AS ( -- tmp stores the previous query results to get counts
  SELECT name, count(*) AS num
  FROM Employees
  GROUP BY name),
lowestCount AS (  
  SELECT min(num) AS num
  FROM tmp
)
SELECT *
FROM tmp NATURAL JOIN lowestCount;

<table>
<thead>
<tr>
<th>name</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>3</td>
</tr>
<tr>
<td>John</td>
<td>3</td>
</tr>
<tr>
<td>Julia</td>
<td>1</td>
</tr>
<tr>
<td>Lynn</td>
<td>1</td>
</tr>
<tr>
<td>Ned</td>
<td>1</td>
</tr>
<tr>
<td>Ross</td>
<td>1</td>
</tr>
<tr>
<td>Sara</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julia</td>
<td>1</td>
</tr>
<tr>
<td>Lynn</td>
<td>1</td>
</tr>
<tr>
<td>Ned</td>
<td>1</td>
</tr>
<tr>
<td>Ross</td>
<td>1</td>
</tr>
<tr>
<td>Sara</td>
<td>1</td>
</tr>
</tbody>
</table>
Problem: Selecting Certain Groups

Q5: Determine all departments that have more than 2 employees.

```
SELECT Dept, count(*) AS num
FROM Employees
WHERE num >= 2
GROUP BY Dept;
```

Won't parse!
num doesn't exist here

Let's write in relational algebra to see why
HAVING Clause

- We need to *delay* the selection of groups to the end
  - HAVING class

- Q5: *(continued)*

```sql
SELECT Dept, count(*) AS num
FROM Employees
GROUP BY Dept
HAVING num >= 2;
```
Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
- Triggers (DDL)
- Transactions
- Conclusion
SQLite Triggers

- **Triggers** are actions that automatically occur based on a database event
  - User defined, each trigger associates with a relation

- Useful for:
  - Leaving a trail for security audits
  - Replicating data onto another database
  - ...
Motivating Scenario

- Want to track when users are:
  - Created, deleted, or when they update their information

```sql
create table users (  
    user_id TEXT PRIMARY KEY,  
    passwd TEXT NOT NULL,  
    f_name TEXT NOT NULL,  
    l_name TEXT NOT NULL
)

-- created just for audits of the user table
create table auditlog (  
    time TEXT,  
    event TEXT,  
    PRIMARY KEY(time)
)
```
Creating a Trigger

- Syntax:

```sql
CREATE TRIGGER trigger_name
BEFORE | AFTER event ON table_name
BEGIN
    sql_stmt1; sql_stmt2; ...
END;
```

- event may be: insert, update, or delete

For our example:

```sql
CREATE TRIGGER new_user_t
AFTER INSERT ON users
BEGIN
    INSERT INTO auditlog
    VALUES (date('now'), 'account created!');
END;
```
sqlite> insert into users values ('brichards', 'test123', 'Brad', 'Richards');
sqlite> insert into users values ('aasmith', '5nakes', 'Adam', 'Smith');

sqlite> select * from users;
user_id     passwd      f_name      l_name
----------  ----------  ----------  ----------
brichards   test123     Brad        Richards
aasmith     5nakes      Adam        Smith

sqlite> select * from auditlog;
time        event
----------  --------------------------
2018-09-20  account created!
2018-09-20  account created!
Better Logs?

- Those logs weren't very descriptive.
  - We want to know *who* was created

```sql
create table users (
    user_id TEXT PRIMARY KEY,
    passwd TEXT NOT NULL,
    f_name TEXT NOT NULL,
    l_name TEXT NOT NULL
);

-- created just for audits of the user table
create table auditlog (
    user_id TEXT,
    time TEXT,
    event TEXT
);
```
Creating a Trigger

- Syntax:

```
CREATE TRIGGER trigger_name
BEFORE | AFTER event ON table_name
BEGIN
    sql_stmt1; sql_stmt2; ...
END;
```

- event may be: insert, update, or delete

- For our example:

```
CREATE TRIGGER new_user_t
AFTER INSERT ON users
BEGIN
    INSERT INTO auditlog
    VALUES (NEW.user_id, date('now'), 'account created!');
END;
```

NEW and OLD are references to event-causing tuple
sqlite> insert into users values ('brichards', 'test123', 'Brad', 'Richards');
sqlite> insert into users values ('aasmith', 'c4t5', 'Adam', 'Smith');

sqlite> select * from users;
user_id     passwd      f_name      l_name
----------  ----------  ----------  ----------
brichards   test123     Brad        Richards
aasmith     5nakes      Adam        Smith

sqlite> select * from auditlog;
user_id     time        event
----------  ----------  --------------------------
brichards   2018-09-20  account created!
aasmith     2018-09-20  account created!
When to Use the NEW vs. OLD Reference?

- **On INSERT:**
  - NEW.A refers to attribute A of the new tuple
  - OLD is undefined

- **On UPDATE:**
  - NEW.A refers to the new value for attribute A
  - OLD.A refers to the old value for A

- **On DELETE:**
  - NEW.A is undefined
  - OLD.A refers to attribute A of the deleted tuple
Another Example

- Log when user makes a change to their account!

```
CREATE TRIGGER upd_user_t
AFTER UPDATE ON users
BEGIN
    INSERT INTO auditlog
    VALUES (NEW.user_id, date('now'), 'account changed: ' ||
            OLD.user_id || '=>' || NEW.user_id || ',' ||
            OLD.passwd || '=>' || NEW.passwd);
END;
```

No joke, || means concatenate in sqlite

```
sqlite> update users set passwd='p4ssw0rd' where user_id='aasmith';
sqlite> select * from auditlog;

<table>
<thead>
<tr>
<th>user_id</th>
<th>time</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>brichards</td>
<td>2018-09-21</td>
<td>account created: brichards</td>
</tr>
<tr>
<td>aasmith</td>
<td>2018-09-21</td>
<td>account created: aasmith</td>
</tr>
<tr>
<td>aasmith</td>
<td>2018-09-21</td>
<td>account changed: aasmith=&gt;aasmith,5nakes=&gt;p4ssw0rd</td>
</tr>
</tbody>
</table>
```
Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
- Triggers
- Transactions
- Conclusion
Consider the following Bank database:

- Example: David sold a copy of **DavidDB Pro** to Brad for $175.
  - Brad wants transfer funds from his savings to David's checking...

```
update checking set balance=275 where acct_id=9291;  -- give David the money
update savings set balance=25 where acct_id=1111;   -- subtract from Brad's account
```
Dealing with Transactions (Cont.)

- Crashes halfway into the transaction
  - David is richer, Brad's account is never deducted -- we both win!
  - Bank loses --- problem!

```sql
update checking set balance=275 where acct_id=9291; -- give David the money

System crashes here, n0000ooooooo..!!! :-(

update savings set balance=25 where acct_id=1111; -- subtract from Brad's account
```

- How should we resolve this issue?
Transactions

- **Database Transactions**: sometimes, a "unit of work" on the database requires a sequence of DML statements.

- A transactions allows multiple statements to be run **atomically**
  - *(i.e., it cannot be split up)*

- Two operations:
  - **COMMIT**
    - Makes all updates to the DB permanent
  - **ROLLBACK**
    - Un-does every statement in the transaction
Using Transactions in SQLite

```
sqlite> BEGIN TRANSACTION;
sqlite> update checking set balance=275 where acct_id=9291; -- give David the money

System crashes here, n0000ooooooooo..!!! :~(

(restart sqlite...)

sqlite> select * from checking;
1110|Henry|200.0
2320|Emily|300.0
9291|David|100.0

-- Yayyyyyy... database is consistent! :^)
-- We never committed the first update!
```
Using Transactions in SQLite

- No fail execution

```sql
sqlite> BEGIN TRANSACTION;
sqlite> update checking set balance=275 where acct_id=9291; -- give David the money
sqlite> update savings set balance=25 where acct_id=1111; -- subtract from Brad's
sqlite> COMMIT TRANSACTION;

sqlite> select * from checking;
1110|Henry|200.0
2320|Emily|300.0
9291|David|275.0

sqlite> select * from savings;
1111|Brad|25.0
2321|America|300.0
9292|David|100.0
```
When to Use ROLLBACK?

- Use ROLLBACK to undo the entire transaction

```sql
sqlite> BEGIN TRANSACTION;
sqlite> -- make hundreds of valid updates and insertions
sqlite> -- make hundreds more valid updates and insertions
...
sqlite> delete from savings; -- OH GOD
sqlite> ROLLBACK; -- phew
```
Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
- Triggers
- Transactions
- Conclusion
In Conclusion...

- SQL first appeared in **IBM System-R** (1976)
  - Now the standard relational data query language
  - Same expressivity as relational algebra, but user-friendly

- Two parts:
  - DDL: Deals with structure of database
  - DML: Deals with data

- Next: How to design databases?
Reminders

- Hwk 3 due Monday 10/7
- Exam Friday 10/11

Next Monday: SQL Lab

- Please bring laptop with you (with SQLite installed)

Last time...

- Outer joins
- AS, WITH clauses
Administrivia 10/2

- Reminders
  - Hwk 3 due Monday 10/7
  - Exam Friday 10/11

- SQL lab days
  - Allowed to bring laptop (with SQLite installed and DB Browser)
  - This Friday and Next Monday

- Last time...
  - Subqueries using IN, EXISTS

- Today:
  - Aggregation and grouping, triggers, and transactions