Announcements

- TA office hours start next week, times posted on website
- HW1 out now, due October 2
  - Demo.sql files from section and today have useful examples
  - Check the message board for HW questions and up-to-date information
    - https://us.edstem.org/courses/136/discussion/
    - Recommended style guide posted

Recap

- A lot of content was presented last time
  - Data models
  - Relational Model
  - SQL
- Let’s take a step back and reevaluate

Recap - The Relational Model

- Flat tables, static and typed attributes, etc.
  - “It’s a spreadsheet with rules”

Recap - Basic SQL query

- Declarative query language
  - Tell the computer what you want, not how to get it
- Languages like Java/Python are procedural
- Declarative query language allows physical data independence

Structured Query Language - SQL

- Declarative query language
  - Tell the computer what you want, not how to get it
- Languages like Java/Python are procedural
- Declarative query language allows physical data independence
Select basic SQL query

SELECT *
FROM Payroll

Select P.Name, P.UserID
FROM Payroll AS P
WHERE P.Job = 'TA';

How does a computer understand abstract SQL text?

Wait!

How does a computer understand abstract SQL text?

Parser

Relational Database Management Systems (RDBMSs) use Relational Algebra (RA)
Database Internals

- Code has to boil down to instructions at some point.
- Relational Database Management Systems (RDBMSs) use Relational Algebra (RA).

SELECT P.Name, P.UserID
FROM Payroll AS P
WHERE P.Job = 'TA';

Details will be covered later.

For each semantics

SELECT P.Name, P.UserID
FROM Payroll AS P
WHERE P.Job = 'TA';

Tuples “flow” up the query plan, getting filtered and modified.

- It’s important to define the semantics (meaning) of a query.

SELECT P.Name, P.UserID
FROM Payroll AS P
WHERE P.Job = 'TA';

Payroll

Name | UserID | Job | Salary
---|---|---|---
Jack | 123 | TA | 50000
Allison | 345 | TA | 60000
Magdi | 567 | Prof | 90000
Dan | 789 | Prof | 100000

For each row in P:
if (row.Job == 'TA'):
output (row.Name, row.UserID)
For each semantics

SELECT P.Name, P.UserID
FROM Payroll AS P
WHERE P.Job = 'TA';

September 27, 2019
SQL Basics

19

For each semantics

SELECT P.Name, P.UserID
FROM Payroll AS P
WHERE P.Job = 'TA';

September 27, 2019
SQL Basics

20

For each semantics

SELECT P.Name, P.UserID
FROM Payroll AS P
WHERE P.Job = 'TA';

September 27, 2019
SQL Basics

21

Recap – SQL and RA

- SQL
  - “What data do I want”
- RA
  - “How do I get the data”

(Next few lectures)
(After SQL)

24
What’s Next?

- Creating tables
- Keys → Identification
- Foreign Keys → Relationships
- Joins in SQL and RA
  - Inner joins
  - Outer joins
  - Self joins

Create Table Statement

```
CREATE TABLE Payroll (UserID INT, Name VARCHAR(100), Job VARCHAR(100), Salary INT);
```

Data types

- Each attribute has a type.
- Examples types:
  - Strings: CHAR(20), VARCHAR(50), TEXT
  - Numbers: INT, SMALLINT, FLOAT
  - MONEY, DATETIME, ...
- Few more that are DBMS specific
  - Statically and strictly enforced

- Generally you will use:
  - VARCHAR(N) for strings where N is the maximum character length
  - INT, FLOAT for numbers (INTEGER works in SQLite)
  - DATETIME (or VARCHAR(N) in SQLite) for dates

Everything is case-insensitive, but having your own guidelines is useful for readability.
A Key is one or more attributes that uniquely identify a row.

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Definitely not a key

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Definitely not a key

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Good candidate for a key

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
<tr>
<td>913</td>
<td>Peter</td>
<td>TA</td>
<td>60000</td>
</tr>
</tbody>
</table>

Is this a good candidate for a key?
**Key**

A Key is one or more attributes that uniquely identify a row.

Data comes from the real world so models ought to reflect that.

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
<tr>
<td>913</td>
<td>Peter</td>
<td>TA</td>
<td>60000</td>
</tr>
</tbody>
</table>

CREATE TABLE Payroll (UserID INT, Name VARCHAR(100), Job VARCHAR(100), Salary INT);

CREATE TABLE Payroll (UserID INT, Name VARCHAR(100), Job VARCHAR(100), Salary INT);

CREATE TABLE Payroll (UserID INT PRIMARY KEY, Name VARCHAR(100), Job VARCHAR(100), Salary INT);

CREATE TABLE Payroll (UserID INT, Name VARCHAR(100), Job VARCHAR(100), Salary INT, PRIMARY KEY (UserID, Name));
Foreign Keys

- Databases can hold multiple tables
- How do we capture relationships between tables?

Foreign Key

A Foreign Key is one or more attributes that uniquely identify a row in another table.

Payroll

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Regist

<table>
<thead>
<tr>
<th>UserID</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Charger</td>
</tr>
<tr>
<td>345</td>
<td>Civic</td>
</tr>
<tr>
<td>567</td>
<td>Civic</td>
</tr>
<tr>
<td>789</td>
<td>Pinto</td>
</tr>
</tbody>
</table>

Is this valid?

Nope, 567 is not unique in Regist table

References

- Payroll
- Regist
Joins

• Foreign keys are able to describe a relationship between tables
• Joins are able to realize combinations of data

Inner Joins

• Bread and butter of SQL queries
  • “Inner join” is often interchangeable with just “join”

The Relational Model Revisited

• More complete overview of the Relational Model:
  • Database → collection of tables
  • All tables are flat
  • Keys uniquely ID rows
  • Foreign keys act as a “semantic pointer”
  • Physical data independence

CREATE TABLE Payroll (UserID INT PRIMARY KEY, Name VARCHAR(100), Job VARCHAR(100), Salary INT);
CREATE TABLE Regist (UserID INT REFERENCES Payroll, Car VARCHAR(100));

SELECT Payroll.Name, Regist.Car FROM Payroll AS P JOIN Regist AS R ON P.UserID = R.UserID;

How do we algorithmically get our results?

<table>
<thead>
<tr>
<th>User ID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
<th>User ID</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
<td>123</td>
<td>Charger</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
<td>127</td>
<td>Civic</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
<td>169</td>
<td>Pinto</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SELECT R.Name, R.Car FROM Payroll AS P JOIN Regist AS R ON P.UserID = R.UserID;

<table>
<thead>
<tr>
<th>Name</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack</td>
<td>Charger</td>
</tr>
<tr>
<td>Magda</td>
<td>Civic</td>
</tr>
<tr>
<td>Magda</td>
<td>Pinto</td>
</tr>
</tbody>
</table>
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)

SELECT P.Name, R.Car
    FROM Payroll AS P JOIN Regist AS R
    ON P.UserID = R.UserID

for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
for each row1 in Payroll:
    for each row2 in Regist:
        if (row1.UserID = row2.UserID):
            output (row1.Name, row2.Car)
for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)

for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)

for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)

for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)

for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)

for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)

for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)

for each row1 in Payroll:
for each row2 in Regist:
if (row1.UserID = row2.UserID):
output (row1.Name, row2.Car)
Now I want to include everyone, even if they don’t drive.

### Outer Joins

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UserID</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Charger</td>
</tr>
<tr>
<td>345</td>
<td>Civic</td>
</tr>
<tr>
<td>567</td>
<td>Civic</td>
</tr>
<tr>
<td>789</td>
<td>Pinto</td>
</tr>
</tbody>
</table>

```sql
SELECT P.Name, R.Car
FROM Payroll AS P
LEFT OUTER JOIN Regist AS R
ON P.UserID = R.UserID;
```

### Outer Joins

- **LEFT OUTER JOIN**
  - All rows in left table are preserved
- **RIGHT OUTER JOIN**
  - All rows in right table are preserved
- **FULL OUTER JOIN**
  - All rows are preserved

### Self Joins

Find all people who drive a Civic and Pinto

```sql
SELECT P.Name, R.Car
FROM Payroll AS P, Regist AS R
WHERE P.UserID = R.UserID AND R.Car = 'Civic';
```
Self Joins

Find all people who drive a Civic and Pinto

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
<td>Charger</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
<td>Civic</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
<td>Pinto</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
<td>Pinto</td>
</tr>
</tbody>
</table>

```sql
SELECT P.Name, R.Car
FROM Payroll AS P, Registration AS R
WHERE P.UserID = R.UserID
AND R.Car = 'Civic'
AND R.Car = 'Pinto';
```

Will this work?

Nope, empty set is returned

Discuss with the people around you how you would solve this.

All pairs of cars a person can drive

```sql
SELECT P.Name, R1.Car
FROM Payroll AS P, Registration AS R1, Registration AS R2
WHERE P.UserID = R1.UserID AND
P.UserID = R2.UserID AND
R1.Car = 'Civic';
R2.Car = 'Pinto';
```

A little extra SQL

- ORDER BY - Orders result tuples by specified attributes (default ascending)
  ```sql
  SELECT P.Name, P.UserID
  FROM Payroll AS P
  WHERE P.Job = 'TA'
  ORDER BY P.Salary, P.Name;
  ```

- DISTINCT - Deduplicates result tuples
  ```sql
  SELECT DISTINCT P.Job
  FROM Payroll AS P
  WHERE P.Salary > 70000;
  ```
Takeaways

- We can describe relationships between tables with keys and foreign keys
- Different joining techniques can be used to achieve particular goals
- Our SQL toolbox is growing!
  - Not just reading and filtering data anymore
  - Starting to answer complex questions