Today ...

- Wrap up table creation
- Basic SPJ Queries

Homework

- HW 5 due
- HW 6 out

Library Schemas:

Branch(branch_name, address, phone)

Account(acct_id, acct_name, main_branch)

Loan(acct_id, barcode, checkout_date, due_date, return_date)
Wrapping up SQL Table Creation

Defining surrogate keys

Use `AUTO_INCREMENT` for surrogate keys (start at 1, ...)

```sql
CREATE TABLE pet (
    pet_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name TINYTEXT NOT NULL,
    PRIMARY KEY (pet_id)
);

INSERT INTO pet (name) VALUES ('dog'), ('cat'), ('bird');

SELECT * FROM pet;
+--------+-------+
| pet_id | name  |
+--------+-------+
| 1      | dog   |
| 2      | cat   |
| 3      | bird  |
+--------+-------+

INSERT INTO pet VALUES (5, 'fish');
INSERT INTO pet (name) VALUES ('hamster');

SELECT * FROM pet;
+--------+-------+
| pet_id | name  |
+--------+-------+
| 1      | dog   |
| 2      | cat   |
| 3      | bird  |
| 5      | fish  |
| 6      | hamster |
+--------+-------+
```
More on Weak Entity Sets

- weak entity as a single table (recall 1..1 constraint)
- key of the strong entity is the foreign key
- key of the new table is key of the strong entity plus the partial key
- when strong entity is deleted, all weak entities also deleted

CREATE TABLE Policy (  
  name VARCHAR (20),  
  dob VARCHAR (10),  
  ssn VARCHAR (11) NOT NULL,  
  PRIMARY KEY (name, ssn),  
  FOREIGN KEY (ssn) REFERENCES Employee (ssn) ON DELETE CASCADE  
);

Foreign Key deletion constraints:

- ON DELETE RESTRICT disallows the deletion (the default)
- ON DELETE CASCADE parent row deletion causes child row deletion
- ON DELETE SET NULL sets child row’s FK to NULL

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>dob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Basic SQL Queries

The account and loan tables:

Account

<table>
<thead>
<tr>
<th>acct_id</th>
<th>acct_name</th>
<th>main_branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Alice</td>
<td>Downtown</td>
</tr>
<tr>
<td>102</td>
<td>Bob</td>
<td>Downtown</td>
</tr>
<tr>
<td>103</td>
<td>Alice</td>
<td>South Hill</td>
</tr>
<tr>
<td>104</td>
<td>Chuck</td>
<td>Downtown</td>
</tr>
</tbody>
</table>

Loan

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
<th>checkout_date</th>
<th>due_date</th>
<th>return_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
<td>8/12</td>
<td>8/26</td>
<td>8/24</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>8/12</td>
<td>8/19</td>
<td>NULL</td>
</tr>
<tr>
<td>102</td>
<td>4242</td>
<td>8/25</td>
<td>9/7</td>
<td>8/29</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>7/10</td>
<td>7/17</td>
<td>7/18</td>
</tr>
</tbody>
</table>

SQL “Select-From-Where” queries ... relational algebra SPJ queries

(1) SELECT corresponds to $\pi$ ... SELECT * for all attributes

(2) FROM lists tables being queried / joins

(3) WHERE corresponds to $\sigma$

Note: SQL doesn’t remove duplicates by default (unlike relational algebra)

Example: Find the barcodes of books loaned by account 101

```
SELECT barcode
FROM Loan
WHERE acct_id = 101
```

Similar to: $\pi_{\text{barcode}}(\sigma_{\text{acct_id}=101}(\text{Loan}))$ ... Q: what is different?
Example: Find all books checked out from Downtown accounts

```sql
SELECT barcode
FROM Loan, Account
WHERE main_branch = 'Downtown' AND
    Loan.acct_id = Account.acct_id
```

- example of a "comma" join ... i.e., a cartesian product
- can use parenthesized expressions involving AND, OR, NOT
- relational comparators =, <, >, <=, >=, != ... or <> for !=
- IS NULL instead of = NULL ... more on NULLs later
- IS NOT NULL instead of != NULL

Equivalent to an explicit cartesian product ... called CROSS JOIN

```sql
SELECT barcode
FROM Loan CROSS JOIN Account
WHERE main_branch = 'Downtown' AND
    Loan.acct_id = Account.acct_id
```

Can also use explicit JOIN USING syntax ... same-named join attributes

```sql
SELECT barcode
FROM Loan JOIN Account USING (acct_id)
WHERE main_branch = 'Downtown'
```

- Note that USING removes the duplicate attribute names in the attribute list

Or more general JOIN ON syntax ... more complex join conditions

```sql
SELECT barcode
FROM Loan JOIN Account ON (Loan.acct_id = Account.acct_id)
WHERE main_branch = 'Downtown'
```
And NATURAL JOIN syntax

```sql
SELECT barcode
FROM Loan NATURAL JOIN Account
WHERE main_branch = 'Downtown'
```
Disambiguate attributes using "correlation names"

```
SELECT barcode
FROM Loan l, Account a
WHERE l.acct_id = a.acct_id AND main_branch = 'Downtown'
```

- especially useful for self-joins
- good to follow same style
- my personal preference is to use correlation names
- also to use join syntax over comma joins

Can "chain" together products ...

```
SELECT l.barcode, a.acct_name, b.address
FROM Loan l, Account a, Branch b
WHERE l.acct_id = a.acct_id AND a.main_branch = b.branch_name
```

Same query but using JOIN syntax:

```
SELECT l.barcode, a.acct_name, b.address
FROM Loan l JOIN Account a USING (acct_id)
    JOIN Branch b ON (a.main_branch = b.branch_name)
```