Today …

- Creating Tables
- Inserting Rows

Homework

- HW 4 due
- HW 5 out
- No class Thurs (10/14)
Basic Table Declarations (CREATE TABLE)

```
CREATE TABLE tablename (
    att1 type1,  
    att2 type2,  
    ...  
);
```

where:

- `att1` is the name of the attribute and `type1` is the data type (domain)
- table is created from within an existing database

We can also add various additional information:

- constraints (e.g., NOT NULL, keys, foreign keys, UNIQUE)
- default values
- in MySQL, the underlying DB engine
- and so on ...
Defining the branch table:

branch(branch_name, address, phone)

Q: What would be reasonable data types?

A simple version of a CREATE TABLE statement ...

```sql
/* basic account information */
CREATE TABLE branch (  
  branch_name VARCHAR(50),  
  address TINYTEXT,  
  phone VARCHAR(12) -- e.g., format: NNN-NNN-NNNN
);
```

• Hint: In MariaDB, use `explain branch;` to see the schema

• Hint: Use `show create table branch;` to see the SQL

Adding "branch name" as a primary key ...

```sql
/* basic branch information */
CREATE TABLE branch (  
  branch_name VARCHAR(50),  
  address TINYTEXT,  
  phone VARCHAR(12),  
  PRIMARY KEY (branch_name) -- a PK "constraint"
);
```
Q: How would we create this table? ... ignoring FKs for now

- account(acct_id, acct_name, main_branch)

```sql
CREATE TABLE account (
    acct_id INT UNSIGNED,
    acct_name TINYTEXT,
    main_branch VARCHAR(50),
    PRIMARY KEY (acct_id)
);

FOREIGN KEY (atts) REFERENCES table (atts)
```

Adding the `account` foreign key:

```sql
CREATE TABLE account (
    acct_id INT UNSIGNED,
    acct_name TINYTEXT,
    main_branch VARCHAR(50),
    PRIMARY KEY (acct_id),
    FOREIGN KEY (main_branch) REFERENCES branch (branch_name)
);
```

Another common constraint: Requiring non-null values (NOT NULL)

```sql
CREATE TABLE account (
    acct_id INT UNSIGNED NOT NULL, -- Redundant with PK
    acct_name TINYTEXT NOT NULL, -- NULL not allowed
    main_branch VARCHAR(50) NOT NULL, -- must have branch
    PRIMARY KEY (acct_id),
    FOREIGN KEY (main_branch) REFERENCES branch (branch_name)
);
```
Adding non-primary candidate keys: **UNIQUE** constraints

**CREATE TABLE** branch (  
    branch_name VARCHAR(50),  
    address TINYTEXT NOT NULL,  
    phone VARCHAR(12) NOT NULL,  
    PRIMARY KEY (branch_name),  
    UNIQUE (phone) -- phone also identifies branch  
);

- **UNIQUE** is similar to a primary key constraint, but allows NULL values

It is also possible to name constraints:

**CREATE TABLE** account (  
    acct_id INT UNSIGNED NOT NULL, -- Redundant with PK  
    acct_name TINYTEXT NOT NULL, -- NULL not allowed  
    main_branch VARCHAR(50) NOT NULL, -- must have branch  
    PRIMARY KEY (acct_id),

    CONSTRAINT account_fk  
    FOREIGN KEY (main_branch) REFERENCES branch (branch_name)  
);

Can then remove constraints “after the fact” (by name)

**ALTER TABLE** account **DROP FOREIGN KEY** account_fk;

Can also add constraints “after the fact”

**ALTER TABLE** account **ADD CONSTRAINT** account_fk  
FOREIGN KEY (main_branch) REFERENCES branch (branch_name);
General constraints using CHECK statements ...

```
CREATE TABLE loan (
    acct_id INT UNSIGNED NOT NULL, -- the account
    barcode INT UNSIGNED NOT NULL, -- the book id
    checkout_date DATE NOT NULL, -- when book was checked out
    due_date DATE NOT NULL, -- when book due is due
    return_date DATE, -- when book was returned or NULL

    PRIMARY KEY (acct_id, barcode, checkout_date),

    CONSTRAINT loan_fk
        FOREIGN KEY (acct_id) REFERENCES account (acct_id),

    CONSTRAINT valid_barcode_value
        CHECK (barcode > 0 and barcode <= 1000000),

    CONSTRAINT valid_due_date
        CHECK (checkout_date < due_date),

    CONSTRAINT valid_return_date
        CHECK (return_date IS NULL or return_date >= checkout_date)
);
```

Q: Give an example row that violates the above ...
More on Modifying Tables using SQL

Drop a column from an existing table ...

   ALTER TABLE my_table DROP COLUMN my_column;

Change name and type of an attribute ...

   ALTER TABLE my_table CHANGE my_column my_new_column data_type;

Change type of an attribute (leave name unchanged) ...

   ALTER TABLE my_table MODIFY my_column new_data_type;

Add a new attribute ...

   ALTER TABLE my_table ADD my_new_column data_type;
Inserting Values

Basic forms of row insertion ...

\[
\begin{align*}
\text{INSERT INTO table VALUES (v1, v2, ...);} \\
\text{INSERT INTO table VALUES (v1, v2, ...), (v3, v4, ...), ...;} \\
\text{INSERT INTO table(a1, a2) VALUES (v1, v2);} \\
\text{INSERT INTO table SET a1 = v1, a2 = v2, ...;}
\end{align*}
\]

For example:

\[
\text{INSERT INTO account VALUES (1, 'Alice', 'Downtown');}
\]

Using default values

\[
\begin{align*}
\text{CREATE TABLE account (} \\
\hspace{1em} \text{acct_id INT UNSIGNED NOT NULL,} \\
\hspace{1em} \text{acct_name TINYTEXT NOT NULL,} \\
\hspace{1em} \text{main_branch VARCHAR(50) NOT NULL DEFAULT 'Downtown',} \\
\hspace{1em} \text{...} \\
\hspace{1em} \text{);} \\
\end{align*}
\]

Default values in insert ...

\[
\text{INSERT INTO account(acct_id, acct_name) VALUES (102, 'Bob');}
\]

Producing the row:

\[
\begin{array}{|c|c|c|}
\hline
| 102 | Bob | Downtown |
\hline
\end{array}
\]
Removing rows

Removing all rows from tables

```sql
DELETE FROM loan;
```

Can also remove specific rows using an SQL query (more later):

```sql
DELETE FROM loan
WHERE acct_id = 11;
```

Updating Rows

Update the return_date of all rows

```sql
UPDATE loan SET return_date = '2021-10-12';
```

Update the return_date of specific rows

```sql
UPDATE loan SET return_date = '2021-10-12' WHERE acct_id = 11;
```

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');
```
Removing Entire Tables

Removing a table that already exists

```
DROP TABLE loan;
```

- Note: this removes the entire table!

Checking that the table exists before removing it

```
DROP TABLE IF EXISTS loan;
```

- only removes table if it has already been created
- good for avoiding errors in a script

Note: if you have foreign keys, the order you drop tables matters!

Q: Why? ...

- In our example:

```
DROP TABLE IF EXISTS loan; -- must be before account (FK)
DROP TABLE IF EXISTS account; -- can now drop account
...
```
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
Using “Scripts” with MariaDB

Much easier/better to create tables via script file
- which is just a text file with SQL statements
- can be “executed” in MariaDB using the `source` command
- which reads and executes each statement

File path is relative to where you run `mysql` command

- On `ada`:

```
bowers@ada:~/cpsc321$ ls
accounts.sql
```

- Execute `mysql`

```
bowers@ada:~/cpsc321$ mysql -p -h cps-database
Enter password:
...
mysql>
```

- Select DB (can also be added to script):

```
mysql> use bowersDB;  -- use your DB here
...
```

- Execute your script (stored on `ada`):

```
mysql> source accounts.sql
...
```