Lecture 5:
• Quiz 1
• SQL Intro (cont)

Announcements:
• PS-1 due
• HW-1 out, due Thur, Sept 22
• PS-2 out, due Tues, Sept 27

REVIEW: Creating Tables in SQL (MySQL/MariaDB)

Basic **create table** syntax:

```
CREATE TABLE  tablename ( 
   attribute1  type1,  
   attribute2  type2,  
   ...  
);  
```

where `attribute_i` is the name and `type_i` is the data type (e.g., `vt_id INT`)  

Can also add various additional information:
• constraints (e.g., NOT NULL, keys, foreign keys, UNIQUE, etc.)  
• default values  
• etc.
(5) Date and Time values:

- **YEAR** `YYYY` (or `YYYY`) format from 1901 to 2155 (and 0000)
- **DATE** `YYYY-MM-DD` format
- **TIME** multiple formats, e.g., `HH:MM` and `HH:MM:SS`
- **DATETIME** both a date and a time, multiple formats

(6) Others:

- various other string and numeric types
- geometric types
- JSON values
- SET types (e.g., `SET(`value1', `value2', ...))

Examples: Basic Tables and PKs

**Defining a (library) branch table:** `branch(branch_name, address, phone)`

Q: What would reasonable data types be?

Simple version of the CREATE TABLE statement ... 

```sql
/* stores basic account information */
CREATE TABLE branch (
    branch_name VARCHAR(50),
    address TINYTEXT,
    phone VARCHAR(12), -- e.g., format: NNN-NNN-NNNN
    PRIMARY KEY (branch_name) -- syntax for one-attribute PK
);
```

- Hint: In MariaDB, use "explain branch;" to see the schema
- Hint: Use "show create table branch;" to see the SQL
Tables with FKs

Adding an account table: `account(acct_id, acct_name, main_branch)`

• where `main_branch` is a FK to the `branch` table

```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)  
);  
```

In general, FKs take the form:

```sql
FOREIGN KEY (att1, ..., attn) REFERENCES table (att1, ..., attn)
```

NOT NULL Constraints

Another common constraint is 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)  
);  
```

`NOT NULL` is another form of integrity constraint

• e.g., stating the attribute is “required” (as opposed to being “optional”)
Non-Primary Key Constraints

Adding non-primary candidate keys via 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

• in this case, we don’t allow NULL values via NOT NULL

Inserting Rows

Basic forms of row insertion ...

INSERT INTO table VALUES (v1, v2, ...);
INSERT INTO table VALUES (v1, v2, ...), (v3, v4, ...), ...;
INSERT INTO table(a1, a2) VALUES (v1, v2);
INSERT INTO table SET a1 = v1, a2 = v2, ...;

For example:

INSERT INTO account VALUES (101, 'Alice', 'Central');

INSERT INTO account VALUES (102, 'Bob', 'Central'), (103, 'Charlie', 'Shadle');
Modifying Rows

Removing all rows from tables

```sql
DELETE FROM account;
```

Removing specific rows using query conditions (more later):

```sql
DELETE FROM account
WHERE acct_id = 101;
```

Updating attribute value of all rows

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

Updating attribute value of specific rows

```sql
UPDATE loan SET return_date = '2021-10-12' WHERE acct_id = 101;
```

Additional Features: Surrogate Keys and Defaults

**Defining surrogate keys with AUTO_INCREMENT**

```sql
CREATE TABLE pet (  
    pet_id INT UNSIGNED NOT NULL AUTO_INCREMENT, -- surrogate key  
    pet_name TINYTEXT NOT NULL,  
    pet_type TINYTEXT NOT NULL DEFAULT 'dog', -- default pet type  
    PRIMARY KEY (pet_id)  
);
```

Then use named-attribute version of INSERT:

```sql
INSERT INTO pet (pet_name) VALUES ('fido');  
INSERT INTO pet (pet_name, pet_type) VALUES ('bill', 'cat');  
INSERT INTO pet VALUES (4, 'tom', 'cat');
```

Note that AUTO_INCREMENT values start at 1 and increment from highest value.
**Additional Features: Named Constraints**

Sometimes useful to “name” constraints

```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),
    CONSTRAINT account_fk -- named FK constraint
        FOREIGN KEY (main_branch) REFERENCES branch (branch_name)
);
```

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

```sql
ALTER TABLE account DROP FOREIGN KEY account_fk;
```

Can also add constraints “after the fact”

```sql
ALTER TABLE account ADD CONSTRAINT account_fk
    FOREIGN KEY (main_branch) REFERENCES branch (branch_name);
```

---

**Additional Features: CHECK constraints**

Can specify general constraints with `CHECK` statements ...  

```sql
CREATE TABLE item_loan (
    acct_id INT UNSIGNED NOT NULL, -- the account
    barcode INT UNSIGNED NOT NULL, -- the item 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),
    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)
);
```
Additional Features: Modifying Table Schemas

Sometimes need to change existing schemas

   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;