SQL: Data Definition Language

CSC 343

Winter 2018

MICHAEL LIUT (MICHAEL.LIUT@UTORONTO.CA)

DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL SCIENCES

UNIVERSITY OF TORONTO MISSISSAUGA

Database Schemas in SQL

SQL is primarily a query language, for getting information from a database.

- DML: Data Manipulation Language

But SQL also includes a *data-definition* component for describing database schemas.

- DDL: Data Definition Language

Creating (Declaring) a Relation

Simply:

- CREATE TABLE <table_name> (
  <list of elements>
  );

Deleting a relation:

- DROP TABLE <table_name>;

Elements of Table Declarations

Most basic element: an attribute and its type.

Common Types:

- INT or INTEGER (synonyms).
  - Also: SMALLINT, MEDIUMINT, and BIGINT
- REAL or FLOAT or DOUBLE (synonyms).
- CHAR(*n*) = fixed-length string of *n* characters.
- VARCHAR(*n*) = variable-length string of up to *n* characters.

Additional data type found here:
https://dev.mysql.com/doc/refman/5.7/en/data-types.html

Example: Create Table

```
CREATE TABLE Sells (
        bar     CHAR(20),
        beer    VARCHAR(20),
        price   REAL
);
```

SQL Values

Integers and reals are represented as you would expect.

Strings are too, except they require single quotes.

- Two single quotes = real quote

  e.g. 'Joe''s Bar'.

Any value can be NULL

- Unless attribute has NOT NULL constraint

  e.g. price REAL not null,

Dates and Times

DATE and TIME are types in SQL

- MySQL even has DATETIME
  - Ranging from 1000-01-01 00:00:00 to 9999-12-31 23:59:59

The form of a date value is:

DATE 'yyyy-mm-dd'

e.g. DATE '2007-09-30' is September 30th, 2007.

Times and Values

The form of a time value is:

- TIME ‘hh:mm:ss’ – with an optional decimal point for a fraction of a second.

e.g. TIME ‘15:30:02.5’ = two and a half seconds after 3:30pm.

Declaring Keys

An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE.

Either says that no two tuples of the relation may agree in all the attribute(s) on the list.

Example

Beers (name, manf)
Bars (name, addr, license)
Drinkers (name, addr, phone)
Likes (drinker, beer)
Sells (bar, beer, price)
Frequents (drinker, bar)

Underline = *key* (tuples cannot have the same value in all key attributes)

Declaring Single-Attribute Keys

Place PRIMARY KEY or UNIQUE KEY after the type in the declaration of the attribute.

Example:

```
CREATE TABLE Beers (
        name CHAR(20) UNIQUE,
        manf CHAR(20)
);
```

Declaring Multi-Attribute Keys

A key declaration can also be another element in the list of elements of a CREATE TABLE statement.

This form is essential if the key consists of more than one attribute.

- Could be used for one-attribute keys.

Example

The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
        bar     CHAR(20),
        beer    VARCHAR(20),
        price   REAL,
        PRIMARY KEY (bar, beer)
);
```

Primary Key vs. Unique

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.

2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

Types of Constraints

1. **Keys**
2. **Foreign-key** (referential-integrity) constraint
3. **Domain** constraint
   - Constrain values of a particular attribute.
4. **Tuple-based** constraint
   - Relationship among components.
5. **Assertions**: any SQL Boolean expressions

Foreign Keys

Values appearing in attributes of one relation must appear together in certain attributes of another relation.

Example: in Sells(bar, beer, price), we might expect that a beer value also appears in Beers.name

Expressing Foreign Keys

Use keyword REFERENCES, either:

1. After an attribute (for one-attribute keys).
2. As an element of the schema:

   FOREIGN KEY (<list of attributes>)

   REFERENCES <relation> (<attributes>)

Referenced attributes must be declared PRIMARY KEY or UNIQUE.

Example: with Attribute

```
CREATE TABLE Beers (
        name   CHAR(20)        PRIMARY KEY,
        manf   CHAR(20)
);
CREATE TABLE Sells (
        bar    CHAR(20),
        beer   CHAR(20)        REFERENCES    Beers(name),
        price  REAL
);
```

Example: As Schema Element

```
CREATE TABLE Beers (
        name   CHAR(20)        PRIMARY KEY,

        manf   CHAR(20)

);

CREATE TABLE Sells (
        bar    CHAR(20),
        beer   CHAR(20),

        price  REAL

        FOREIGN KEY (beer)   REFERENCES   Beers(name)

);
```

Enforcing Foreign-Key Constraints

If there is a foreign-key constraint from relation *R* to relation *S*, two violations are possible:

1. An insert or update to *R* introduces values not found in *S*.
2. A deletion or update to S causes some tuples of *R* to "dangle."

Actions Taken

Example: suppose R = Sells, S = Beers.

An insert or update to **Sells** that introduces a nonexistent beer must be rejected.

A deletion or update to **Beers** that removes a beer value found in some tuples of **Sells** can be handled in three ways.

Actions Taken

Default: Reject the modification.

Cascade: Make the same changes in Sells.

- Deleted beer: delete Sells tuple.
- Updated beer: change value in Sells.

Set Null: Change the beet to NULL.

Example: Cascade

Delete the Bud tuple from Beers:

- Then delete all tuples from Sells that have beer = ‘Bud’.

Update the Bud tuple by changing ‘Bud’ to ‘Budweiser’

- Then change all Sells tuples with beer = ‘Bud’ to beer = ‘Budweiser’

Example: Set NULL

Delete the Bud tuple from Beers:

- Change all tuples of Sells that have beer = ‘Bud’ to have beer = NULL.

Update the Bud tuple by changing ‘Bud’ to ‘Budweiser’:

- Same change as for deletion.

Choosing a Policy

When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates.

Follow the foreign-key declaration by: ON [UPDATE, DELETE][SET NULL CASCADE]

Two such clauses may be used.

Otherwise, the default (reject) is used.

Example: Setting Policy

```
CREATE TABLE Sells (
        bar  CHAR(20),
        beer  CHAR(20),
        price  REAL,
        FOREIGN KEY(beer)
                REFERENCES Beers(name)
                ON DELETE SET NULL
                ON UPDATE CASCADE
);
```

Attribute-Based Checks

Constraints on the value of a particular attribute.

Add CHECK(<condition>) to the declaration for the attribute.

The condition may use the name of the attribute, but any other relation or attribute must be in a sub-query.

Example: Attribute-Based Check

```
CREATE TABLE Sells (
        bar CHAR(20),
        beer CHAR(20) CHECK (
                beer IN (SELECT name FROM Beers)),
        price REAL CHECK (price <= 5.00)
);
```

Timing of Checks

Attribute-based checks are performed only when a value for that attribute is inserted or updated.

- **Example:** CHECK (price <= 5.00) checks every new price and rejects the modification (for that tuple) if the price is more than $5.
- **Example:** CHECK (beer IN (SELECT name FROM Beers)) not checked if a beer is deleted from Beers (unlike foreign-keys).

Tuple-Based Checks

CHECK (<condition>) may be added as a relation-schema element.

The condition may refer to any attribute of the relation.

- But other attributes or relations require a sub-query.

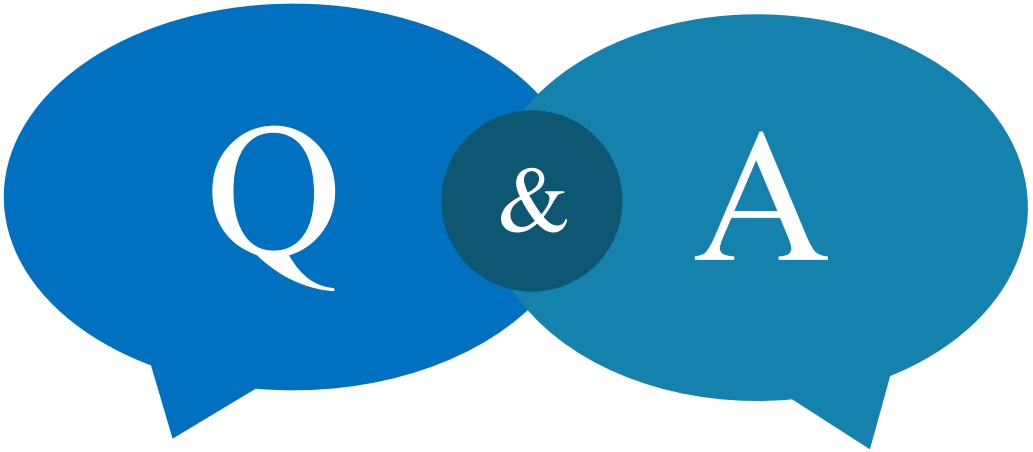
Checked on insert or update only.

Example: Tuple-Based Check

- Only Joe's Bar can sell beer for more than $5.

```
CREATE TABLE Sells (
        bar     CHAR(20),
        beer    CHAR(20),
        price       REAL,
        CHECK (bar = 'Joe''s Bar' OR price <= 5.00)
);
```

Questions?



THANKS FOR LISTENING

I'LL BE ANSWERING QUESTIONS NOW

Citations, Images and Resources

Database Management Systems (3rd Ed.), Ramakrishnan & Gehrke

Some content is based off the slides of Dr. Fei Chiang - http://www.cas.mcmaster.ca/~fchiang/

http://csharpcorner.mindcrackerinc.netdna-cdn.com/UploadFile/BlogImages/06112016031910AM/sql.png