L5: SQL's Data Definition Language

CS1106/CS6503: Intro to Relational Databases

Dr Kieran T. Herley

Semester One, 2023-24

School of Computer Science & Information Technology
University College Cork

Summary

SQL's data definition language. Data types: numerical, temporal and textual. The CREATE command. Primary keys.

Data Definition

Data Definition and Data Manipulation

Data manipulation language (DML)

- SELECT, INSERT, UPDATE etc.

Data definition language (DDL)

- Specify the structure of database's table(s)
- CREATE a table with this structure

A Table Definition for Our Running Example

```
CREATE TABLE students
(
    id_number CHAR(9),
    first_name VARCHAR(20),
    last_name VARCHAR(30),
    date_of_birth DATE,
    hometown VARCHAR(30),
    course CHAR(5),
    points INTEGER,
    . . .
);
```

- Specifies names (table and cols) and col. types
- Creates table with this structure

SQL's main data types

Textual CHAR, VARCHAR, TEXT

Numerical – Integers INT/INTEGER [1]

Numerical – Reals

- Exact values: DEC/DECIMAL (aka NUMERIC)
- Approx. values: FLOAT, DOUBLE (aka REAL)

Temporal DATE, TIME, DATETIME

Others

- Also other types: BLOB (Binary Large Object)
- Often system dependent; MySQL shown

[1] Various "sizes": TINYINT, SMALLINT, MEDIUMINT, BIGINT

Data type variation among DBMSs

- Most standard DBMS software support above, with variations
- SQLite *recognizes* these main types, but may not implement them all faithfully.

Numerical types

INTEGER/INT e.g. 12345 or -67890

DECIMAL(n, d)

- Reals: *n*-digit number with a *d*-digit mantissa

FLOAT, DOUBLE

- Scientific notation (e.g. `1.34E+12` for $1.34 \times 10^{+12}$)
- system-dependent limits on size

Typically system dependent limits on “size” or precision

Temporal Types

DATE

- Dates in `YYYY-MM-DD` format

TIME

- Time (24-hour clock) in `hh:mm:ss` format

DATETIME

- Combined date and time in format `YYYY-MM-DD hh:mm:ss`

DBMSs support useful *functions* for manipulating dates and times e.g. extracting year. Typically system dependent.

Textual types

CHARACTER(n), CHAR(n)

- Shortish, fixed-length strings
- Space for exactly *n* characters allocated
- Shorter strings right-padded with blanks, longer ones truncated

VARCHAR(n)

- Strings of any any length up to max of *n* characters
- May be more space-efficient than CHAR
- Useful where string length not known e.g. addresses

TEXT

- Larger blocks of text e.g. book chapters
- Stored externally in file system – may be less efficient

MySQL vs SQLite types

MySQL Types (Some)	SQLite Types
INTEGER	INTEGER
CHAR(n), VARCHAR(n), TEXT	TEXT
FLOAT, DOUBLE, REAL	REAL
DECIMAL(n, p), NUMERIC, BOOLEAN	NUMERIC
DATE, TIME, DATETIME	TEXT (or NUMERIC)
BLOB	BLOB

- SQLIte *recognizes* standard data types but *maps* them to its own more restrictive set of types.
- We will use the standard (MySQL) types

Meanwhile Back At The Example

```
CREATE TABLE students
(
   id_number CHAR(9),
   first_name  VARCHAR(20),
   last_name VARCHAR(30),
   date_of_birth  DATE,
   hometown VARCHAR(30),
   course CHAR(5),
   points INTEGER,
   . . .
);
```

Should only execute CREATE once when table is first set up

Structure-Altering SQL commands – use sparingly

- To expunge a table:

```
DROP TABLE X;
DROP TABLE IF EXISTS Y;
```

  - Careful– deletes table and contents
- Altering table structure
  - Adding an attribute/column:

```
ALTER TABLE students ADD gender CHAR(1);
```

  - Deleting an attribute/column:

```
ALTER TABLE students DROP hometown;
```

  - If you design your DB properly, you should rarely need these

Keys

students

id_number	*first_name*	*last_name*	*date_of_birth*	*hometown*	*course*	*points*
112345678	Aoife	Ahern	1993-01-25	Cork	ck401	500
112467389	Barry	Barry	1980-06-30	Tralee	ck402	450
112356489	Ciara	Callaghan	1993-03-14	Limerick	ck401	425
112986347	Declan	Duffy	1993-11-03	Cork	ck407	550
112561728	Eimear	Early	1993-07-18	Thurles	ck406	475
112836467	Fionn	Fitzgerald	1994-06-13	Bandon	ck405	485

- Each table should have one or more attributes (collectively known as the *key*) the values of which uniquely identify each row i.e. no two rows should have the same key e.g. id_number above
- Table definition should specify key as shown

```
CREATE TABLE students
(   . . .
    PRIMARY KEY (id_number)
)
```

Constraints (system dependent – SQLite versions)

- Rules to “police” which values are legit within col
- Checked on INSERT/UPDATE
- SQLite constraints [2]:

  NOT NULL values cannot be NULL

  DEFAULT specifies default value when none provided

  UNIQUE all values in col must be different

  CHECK specifies (simple) condition values must satisfy
- SQLite disables checking by default so will ignore hereafter

[2] Also PRIMARY KEY – prefer PRIMARY KEY (. . .) version

An Example

A Simple Database

- Suppose we want to design a DB to hold information about some people
- Information about each person:
  - Name
  - Birth date
  - Address
  - Favourite foods

A Simple Database

- Suppose we want to design a DB to hold information about some people
- Information about each person:
  - Name
  - Birth date
  - Address
  - Favourite foods

  First stab at a DB design:

Column	Type
name	VARCHAR(?)
gender	CHAR(1)
birth_date	DATE
address	VARCHAR(?)
likes	?????

A Simple Database

- Suppose we want to design a DB to hold information about some people
- Information about each person:
    - Name
    - Birth date
    - Address
    - Favourite foods

  First stab at a DB design:

Column	Type
name	VARCHAR(?)
gender	CHAR(1)
birth_date	DATE
address	VARCHAR(?)
likes	?????
- Reasonable first stab, but several imperfections

A second try

Column	Type
person_id	CHAR(6)
first_name	VARCHAR(20)
last_name	VARCHAR(20)
gender	CHAR(1)
birth_date	DATE
street	VARCHAR(30)
town	VARCHAR(30)
county	VARCHAR(30)
favourite_foods	?????

A second try

Column	Type
person_id	CHAR(6)
first_name	VARCHAR(20)
last_name	VARCHAR(20)
gender	CHAR(1)
birth_date	DATE
street	VARCHAR(30)
town	VARCHAR(30)
county	VARCHAR(30)
favourite_foods	?????

Better but what about favourite_foods?

- VARCHAR– difficult to access individual food items
- Separate columns (`fav1`, `fav2`, . . .), but how many?
- Better to use second separate table

The `likes` table

- Introduce seperate table `likes` to capture relationship between persons and their favourite foods.
-

persons

person_id	*first_name*	*last_name*	. . .
⋮	⋮	⋮	⋮
112356489	Ciara	Callaghan	⋯
112986347	Declan	Duffy	⋯
⋮	⋮	⋮	⋮

likes

person_id	*food*
⋮	⋮
112356489	Ice cream
112356489	Chocolate
112986347	Pizza
112986347	Beer
112986347	Crisps
⋮	⋮

- This models the fact that person `112986347` (aka Declan Duffy) likes pizza, beer and crisps
- The "link" between the two tables is `person_id`, a *foreign key*.

Our final design

```
CREATE TABLE persons
(
    person_id   CHAR(6),
    first_name   VARCHAR(20),
   last_name   VARCHAR(20),
   gender   CHAR(1),
    birth_date  DATE,
    street   VARCHAR(30),
   town  VARCHAR(30),
   county   VARCHAR(30),

   PRIMARY KEY (person_id)
);
```

```
CREATE TABLE likes
(
    person_id   CHAR(6),
   food VARCHAR(20),

   PRIMARY KEY (person_id, food)

);
```

Note: two-attribute key

Foreign Keys

Foreign keys

An attribute or group of attributes in one table used to indicate a row in another table is known as a foreign key.

Consider simplified version of persons-likes database:

```
persons(person_id, first_name, last_name)
likes(person_id, food)
```

The `person_id` in `likes` is a foreign key.

Foreign keys cont'd

Ideally should indicate foreign keys in CREATEs:

```
CREATE TABLE persons
(
    person_id CHAR(6),
    first_name VARCHAR(20),
    last_name VARCHAR(20),

    PRIMARY KEY (person_id)
);
```

```
CREATE TABLE likes
(
    person_id CHAR(6),
    food VARCHAR(20),

    PRIMARY KEY (person_id, food),
    FOREIGN KEY(person_id) REFERENCES persons(person_id)
);
```

- which *should* prevent
    - INSERTing row into `likes` unless `person_id` already in `persons`
    - DELETing row into `persons` while `person_id` lingers in `likes`
- SQLite ignores foreign keys by default; we will omit this

Working with Multi-Table DBs

Working With Our New DB

- Easy one-table queries:

```
SELECT *
FROM persons
WHERE first_name = 'Ciara';
```

```
SELECT *
FROM likes
WHERE person_id = '112986347';
```

- What about queries like the following?

  List names of all persons who like pizza

- Need mechanism to reach across both tables!?

What We Have Covered So Far

- Setting up a simple database (CREATE)
- Adding content to the database (INSERT, UPDATE)
- Posing (simple) queries to extract information from database

Notes and Acknowledgements

The favourite foods example is taken from “Learning SQL” by Alan Beaulieu (O’Reilly, 2009). If you are looking for a nice, compact and affordable introduction to SQL, this is a good choice.