# Tables and Data

Class 29

# Data Types

- databases are optimized to store data efficiently
- the type of data is very significant
- the broad categories are
  - character data
  - numeric data
  - time data
- MySQL does not have a Boolean data type
  tinyint unsigned is used instead

# Character Types

varchar(n) stores up to n characters ($n \leq 255$)
n is always 255; no point in any other value

char(n) stores exactly n characters, right-padded with spaces
to n characters ($n \leq 255$)

- these match HTML `<input type="text" />`
- and other input types (e.g., phone, password, etc.)

# Character Types

- text types

| | |
|---|---|
| tinytext | up to $2^8$ bytes |
| text | up to $2^{16}$ bytes |
| mediumtext | up to $2^{24}$ bytes |
| longtext | up to $2^{32}$ bytes |

- these match HTML `<textarea>`

# Character Types

**enum** a string-like object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time

- very compact storage, while simultaneously
- utilizing readable queries and results
- matches `<input type="text" />`

```
create table shirt
(
  name varchar(255),
  size enum('x-small', 'small', 'medium',
            'large', 'x-large')
);
```

# Integer Types

| | |
|---|---|
| tinyint | $2^8$ values |
| smallint | $2^{16}$ values |
| mediumint | $2^{24}$ values |
| int | $2^{64}$ values |
| bigint | $2^{128}$ values |

- the default is signed
- use unsigned for unsigned things e.g., counts
- use unsigned auto-increment for unique keys
- these match HTML `<input type="number" />`

# Exact Fixed-Point Types

decimal(n,m) stores n digits, m of which are to the right of the decimal point ($n \leq 65$)

```
salary decimal(7, 2)
```

can store every value from $-99999.99$ to $99999.99$ with no approximation error

# Floating Point Types

float uses 4 bytes for storage (rarely used)

double uses 8 bytes for storage

# Time Types

date stores dates in yyyy-mm-dd format from 1000-01-01 to 9999-12-31

time stores times in hh:mm:ss format from -838:59:59 to 838:59:59

datetime stores times in yyyy-mm-dd hh:mm:ss format

- these match HTML `<input type="text" />`

# Creating a Table

- create a simple table from the command line

```
create table pet
(
  id int unsigned not null auto_increment,
  name varchar(255),
  breed varchar(255),
  sex enum('F', 'M', 'U'),
  birth date
);
```

- but for reproducibility use a .sql file, e.g., create_pet.sql
- comments begin with “– ”
- use source command to read in the file

```
mysql> source create_pet.sql
```

- beware column names that are reserved words (e.g., order)

# Entity Tables

entity tables store fundamental objects in your system
the equivalent of a class

- the name is a singular concrete noun
- each table row is one object
- columns store only singleton data (e.g., birthdate)
- no multi-value data (e.g., address) or derived data (e.g., age)
- each row is identified by a unique primary key (auto-increment or inherent)
- examples:
  - person
  - product
  - song
  - book

# Primary Key

- each row of an entity table must have a primary key
- unique across all rows, not null, never changes
- some entities have natural primary keys (ISBN for a book, banner ID for a student)
- but never SSN for a person
- an entity without a natural primary key uses an auto_increment column
- example:

```
create table pet
(
  id int unsigned not null auto_increment,
  name varchar(255),
  breed varchar(255),
  sex enum('F', 'M', 'U'),
  birth date,
  primary key(id)
);
```

# One-to-Many Tables (Attributes)

one-to-many tables store attributes of entities when an entity can have multiple values of an attribute

- the name is a concatenation of the entity name followed by the attribute name
- the attribute name is a concrete noun, abstract noun, sometimes derived from an intransitive verb
- not used when an attribute is itself an entity
- the entity is represented by its primary key
- pay attention to which columns can and cannot be null
- examples: person_address, player_score, pet_vaccination, plant_growth

# One-to-Many Attribute Tables

```
create table pet_vaccination
(
  id int unsigned not null auto_increment,
  pet_id int unsigned not null,
  name varchar(255) not null,
  vet varchar(255),
  administered date,
  primary key(id),
  constraint pet_fk foreign key(pet_id)
    references pet(id)
);
```

# Redundant Data

- a key concept of RDBMS is that every datum is stored in exactly one place
- there are no duplicate or redundant data in the system
- so, if we store birthdate, we do not store age, because that is redundant

# Duplicate Data

- consider the words.txt file:

```
abate    verb      to put an end to; nullify
abeyance noun      a lapse in succession
abjure   verb      to renounce upon oath
abrogate verb      to abolish by authoritative
abstruse adjective difficult to comprehend
acarpous adjective effete; no longer fertile
```

- the parts of speech are duplicate data
- even without the parts.txt file, but especially if that file exists

# Eliminating Duplicate Data

instead we need three tables

```
 id   part
+---+-----------+
| 1 | adjective |
| 2 | adverb    |
| 3 | noun      |
| 4 | verb      |
+---+-----------+
```

```
 id    word       definition
+-- +----------+------------------------------+
| 1 | abate    | to put an end to; nullify    |
| 2 | abeyance | a lapse in succession        |
| 3 | abjure   | to renounce upon oath        |
| 4 | abrogate | to abolish by authoritative  |
| 5 | abstruse | difficult to comprehend      |
| 6 | acarpous | effete; no longer fertile    |
+---+----------+------------------------------+
```

```
 word_id   part_id
+---------+--------+
| 1       |  4     |
| 2       |  3     |
| 3       |  4     |
| 4       |  4     |
| 5       |  1     |
| 6       |  1     |
+---------+--------+
```

# Entity-Entity Tables

- express a relationship between two entities
- may be one-to-one (a person has one biological mother)
- may be one-to-many (a person places many orders)
- may be many-to-many (a physician has many patients, and a patient may see many physicians)
- the name is the concatenation of the two entity names (e.g., person_order)
- or the name of the relationship (e.g., role for the relationship between actor and movie)
- columns consist of the primary keys of the two entities, and possibly other columns which are attributes of the relationship

# Entity-Entity Tables

example many-to-many table

```
create table physician_patient
(
  physician_id int unsigned not null,
  patient_id int unsigned not null,
  first_visit date default null,
  primary key(physician_id, patient_id),
  constraint physician_fk foreign key(physician_id)
    references person(id),
  constraint patient_fk foreign key(patient_id)
    references person(id)
);
```

- no auto_increment column
- note the foreign key constraints

# Entity-Entity Tables

Example one-to-many table

```
create table movie_director
(
  director_id int unsigned not null,
  movie_id int unsigned not null,
  salary decimal(8,0) default null,
  primary key(director_id, movie_id),
  unique key(movie_id),
  constraint director_fk foreign key(director_id)
    references director(id),
  constraint movie_fk foreign key(movie_id)
    references movie(id)
);
```

# Inserting Data

data are inserted into only one table at a time

```
insert into actor
  (first_name, last_name, sex, film_count)
values
  ('Shailene', 'Woodley', 'F', 13);
```

- any attributes (columns) left out explicitly get set either to the default value (if any) or to NULL
- an auto-increment column automatically gets set to the next-highest value for that column
- it is a logical error to leave out a column that is declared as not null and which has no default value

# Retrieving Information: One Table

a query that involves only one table is easy:

- what columns have the information you need?
- do any column data need to be transformed?
- what rows have the information you need?
- do any rows need to be aggregated?
- in what order should the information be delivered?

Example: can we see the movies of the 90's listed by rank?

```
select name, rank
from movie
where year between 1990 and 1999
order by rank desc;
```

# Two Tables

- some queries require information from two different tables
- this is where a DB blows away text files

Example: the names of comedy movies

- the movie table has names
- the movie_genre table has has genres
- the two tables need to be joined to get both pieces of information at the same time
- almost always, the join condition will involve primary and foreign keys

```
select movie.name
from movie
join movie_genre on movie.id = movie_genre.movie_id
where movie_genre.genre = 'Comedy';
```

# Ambiguous Column Names

- as soon as more than one table is involved, column names can be ambiguous
- for example, movie, actor, and director each have a column named "id"

  actor and director both have columns named "first_name" and "last_name"
- even if right now there's only one table with a column named "rank" (movie), tomorrow someone might add a new column in, say, the actor table named "rank" and then "rank" would be ambiguous
- therefore, every column name in a query with more than one table must be fully qualified

```
select movie.name
from movie
join movie_genre on movie.id = movie_genre.movie_id
where movie_genre.genre = 'Comedy';
```

# Shortcut Table Names

- fully qualified table names can become tedious
- so within each query, it is common to use shortcut names

```
select m.name
from movie as m
join movie_genre as mg on m.id = mg.movie_id
where mg.genre = 'Comedy';
```

# Two Tables

the names of directors who have a better than even chance of directing a comedy

- the director table has directors' names
- the director_genre table has genre and probability information

```
select concat_ws(' ', d.first_name, d.last_name) as name,
       format(dg.prob, 3) as prob
from director as d
join director_genre as dg on d.id = dg.director_id
where dg.genre = 'Comedy' and dg.prob > 0.5
order by dg.prob desc;
```

# Three Tables

the names of actresses and their characters who played in movies in the 1990s

- the names of actresses are in the actor table
- the names of characters are in the role table
- the year of movies is in the movie table

```
select concat_ws(' ', a.first_name, a.last_name) as actorname,
       r.role
from actor as a
join role as r on a.id = r.actor_id
join movie as m on r.movie_id = m.id
where m.year between 1990 and 1999
      and a.sex = 'F'
order by a.last_name asc;
```

# Four Tables

the names of actresses and their characters who played in comedies in the 1990s, along with the name of the movie

- the names of actresses are in the actor table
- the names of characters are in the role table
- the year and title of movies are in the movie table
- the genre of movies is in the movie_genre table

```
select concat_ws(' ', a.first_name, a.last_name) as actname,
       r.role, m.name
from actor as a
join role as r on a.id = r.actor_id
join movie as m on r.movie_id = m.id
join movie_genre as mg on mg.movie_id = m.id
where m.year between 1995 and 2005
      and a.sex = 'F'
      and mg.genre = 'Comedy'
order by a.last_name asc;
```