

CSCI 241

Database, MySQL, and SQL

Department of CS & QM
Winthrop University
Fall 2011

Prepared by X. Wang

Topics

- Database basics
- Database design
- MySQL Command Line Client (Interpreter)
- SQL basics

Prepared by X. Wang

2

Database Basics

- A relational database is using **tables** to organize data
- Databases are managed by a Relational Database Management System (RDBMS) or **Database Server**
- The database language used by Database Servers is **SQL**
- SQL uses a set of statements to define and manipulate data (Insert, Update, Delete, Select, etc.)

Prepared by X. Wang

3

Database Basics

- In this course, we learn how to use **MySQL Database Server** to manage databases
- MySQL server can manage multiple **databases**
- In a database we can create multiple **tables**
- A table may have multiple **attributes** (columns, or fields), and each attribute has its own name, and data type
- A table contains the data as **tuples** (rows, or records)
- A table must have a **primary key** that formed by one or more attributes to uniquely identify each row

Prepared by X. Wang

4

Database Terminology

- **Database**: A repository to store data in the table form
- **Table**: A concrete data object that stores related data
- **Attributes**: The columns of a table. All rows in a table have the same attributes
- **Rows**: The data entries stored in a table. Rows contains values for each attribute
- **Primary Key**: One or more attributes that contain values to uniquely identify each row
- **Foreign Key**: One or more attribute of a table, which is a primary key of another related table

Prepared by X. Wang

5

Database Terminology

- **Relational Database Management System (RDBMS)** or **Database Server**: A software application that manages data in a database and is based on the relational model
- **Relation Model**: A formal model that uses database, tables, and attributes to store data and manages the relationship between tables
- **SQL**: A standard query language that interacts with a database server
- **Constraints**: Restrictions or limitations on tables and attributes

Prepared by X. Wang

6

Database Terminology

- **Index:** A data structure used for fast access to rows in a table. An index is usually built for the primary key of each table. Indexes can also be built for other attributes when whose attributes are frequently used in queries
- **Entity-Relationship (ER) Modeling:** A technique used to describe the real-world data in terms of entities, attributes, and relationships

Prepared by X. Wang

7

Database Design and Use

- To design and use a database, we usually need to go through the following steps:
 1. Analyze the requirements of the database application
 2. Design database with **Entity-Relationship Modeling**
 3. List all **relation schema** from the ER model (optional)
 4. Create all **tables** based on the ER model or relational schema (usually create an SQL script file, and execute the file in RDBMS Command Line Client)
 5. Use SQL statements to manipulate database tables from a Command Line Client or programming languages

Prepared by X. Wang

8

Entity-Relationship Modeling

- Entity-Relationship (ER) modeling is a simple and clear method of expressing the design of database
- The first step in ER modeling is to identify entities
- **Entities** are **objects or things** that can be described by their characteristics
- An entity is represented by a **single-lined Rectangle** in the ER diagram
- An entity may have multiple **attributes**
- An attribute is represented by an **Ellipse** in the ER diagram

Prepared by X. Wang

9

Entity-Relationship Modeling

- Each entity should have a **primary key**
- The names of the attributes for the primary key are **underlined** in the ER diagram
- If an entity doesn't have its own primary key, the entity is called **Weak Entity**. The primary key of a weak entity depends on its **strong entity**
- A weak entity is represented by a **double-lined Rectangle** in ER diagram

Prepared by X. Wang

10

Entity-Relationship Modeling

- For example, a **customer** is a strong entity that has a **name**, an **address**, a **phone**, and other details. It may have a primary key attribute – **cust_id**
- An **orders** of a customer can be an entity with attributes: **order_id**, **date**, **creditcard**, **expirydate** and others
- The **order_id** attribute doesn't directly form the primary key of the order entity.
- Instead, the **order_id** depends on the customer entity
- So **orders** is a weak entity
- **For each entity, we need to create a table in database**

Prepared by X. Wang

11

Entity-Relationship Modeling

- The second step in ER modeling is to identify the **relationships** between entities
- The relationships may have different types (cardinalities)
- There are three possible relationships between two entities
- **One-to-one:** There is exactly one instance of the first entity for each one instance of the second entity
- A one-to-one relationship is represented by a **line** labeled with a **1** at each end that joins two entities

Prepared by X. Wang

12

Entity-Relationship Modeling

- One-to-many (or many-to-one): There is one or more instances of the second entity for each one instance of the first entity
- A one-to-many relationship is represented by a line labeled with a 1 and an M (or a 1 and an N) at two ends, respectively
- Many-to-many: Each instance of the first entity is related to one or more instances of the second entity, and each instance of the second entity is related to one or more instances of the first entity

Prepared by X. Wang

13

Entity-Relationship Modeling

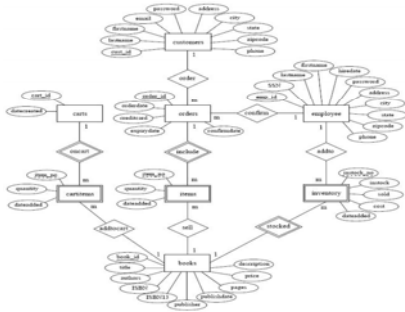
- A many-to-many relationship is represented by a line labeled with an M and an N at two ends, respectively
- A many-to-many relationship may have attributes
- For a many-to-many relationship, we need to create a table in database. The primary key is the combination of two primary keys from two related entities
- Other relationships do not need to have attributes, and also do not need to create tables
- A relationship is represented with a diamond in a ER diagram

Prepared by X. Wang

14

Entity-Relationship Modeling

- An example: the ER diagram of an bookstore database



15

Relation Schema

- After drawing the ER diagram, we can easily list the relation schema for all the tables

1. customers = (cust_id, lastname, firstname, email, password, address, city, state, zipcode, phone)
2. employee = (emp_id, SSN, lastname, firstname, password, hiredate, address, city, state, zipcode, phone)
3. books = (book_id, title, authors, ISBN, ISBN13, publisher, publishdate, pages, price, description)
4. inventory = (book_id, instock_no, instock, sold, cost, emp_id, dateadded)
5. orders = (cust_id, order_id, orderdate, creditcard, expirydate, emp_id, confirmdate)
6. items = (order_id, item_no, quantity, book_id)
7. carts = (cart_id, datecreated)
8. cartitems = (cart_id, item_no, quantity, book_id, dateadded)

Prepared by X. Wang

16

MySQL Command Interpreter

- MySQL command line client or interpreter is a client tool that allows users to use database commands and SQL statements to interactively operate MySQL server
- For Web database applications, the MySQL command interpreter is commonly used to create databases and tables and to test queries
- Once the MySQL server is running, the command interpreter becomes available

Prepared by X. Wang

17

MySQL Command Interpreter

- On Windows, start MySQL Command Line Client from the Start menu, or enter `mysql -uuser -p` from a Command Prompt window
- If a password is required, enter the password
- After entering the MySQL command interpreter, we can see the special prompt as follows:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.5.15-rc-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Prepared by X. Wang

18

MySQL Command Interpreter

- To list help information, type: `help, \h` or `?`
- To quit the command interpreter, type: `exit, quit` or `\q`
- To list all databases, enter: `show databases;`
- To select a database, enter: `use database-name;`
- To list all tables in current database, enter: `show tables;`
- In the interpreter, commands can be entered in either uppercase or lowercase
- The arrow keys can be used to select an executed command

Prepared by X. Wang

19

Creating Databases

- The `CREATE DATABASE` statement is used to create an empty database in MySQL
- This statement only works for the account who has the administrator's (root's) access right
- For example:

```
CREATE DATABASE bookstore;
```

 creates an empty database called `bookstore`

Prepared by X. Wang

20

Creating Tables

- To create a table, we use the statement:
`CREATE TABLE`
- This statement will create an empty table structure with all listed attributes
- To define an attribute, we need to specify its `data type`, and some constraints with `modifiers` if needed

Prepared by X. Wang

21

Creating Tables

- Common SQL data types for attributes in MySQL

Data Type	Comments
<code>int(length)</code>	Integer with a maximum <code>length</code>
<code>decimal(width[,decimal_digits])</code>	A floating-point number with a width and an optional number <code>decimal_digits</code> for the decimal place
<code>datetime</code>	Store a date and time in <code>YYYY-MM-DD HH:MM:SS</code>
<code>time</code>	Store a time in <code>HH:MM:SS</code>
<code>date</code>	Store a date in <code>YYYY-MM-DD</code>
<code>timestamp</code>	Store a timestamp
<code>varchar(length)</code>	A variable-length string with a maximum <code>length</code>
<code>char(length)</code>	A fixed-length string with a <code>length</code>
<code>blob</code>	A binary large object that stores up to 64KB

Prepared by X. Wang

22

Creating Tables

- An example to create the table `customer`

```
CREATE TABLE customers (
  cust_id int(8) NOT NULL auto_increment,
  lastname varchar(50) NOT NULL,
  firstname varchar(50) NOT NULL,
  email varchar(50) NOT NULL,
  password varchar(32) NOT NULL,
  address varchar(50),
  city varchar(30),
  state varchar(20),
  zipcode varchar(10),
  phone varchar(15),
  PRIMARY KEY (cust_id)
) ENGINE=MyISAM;
```

Prepared by X. Wang

23

Creating Tables

- When creating a table, we can specify some constraints with modifiers
- `NOT NULL` is a modifier to specify that the attribute must be given for any a row
- `auto_increment` is a modifier to specify the values for the attribute will be automatically incremented when an insert happens
- `DEFAULT` modifier specifies an default value when no value is provided in an insert

Prepared by X. Wang

24

Creating Tables

- The primary key of the table `customer` is specified with the constraint `PRIMARY KEY (cust_id)`
- If a table's primary key has more than one attributes, separate the attributes' names with comma
- A foreign key can be defined with the constrain `FOREIGN KEY` and reference
- Other non-primary keys can also be specified
- The complete syntax for the `CREATE TABLE` statement, please read [MySQL Manual](#)

Prepared by X. Wang

25

Creating Tables

- We can specify a table's `storage engine` in the `CREATE TABLE` statement
- MySQL Supports several storage engines such as `MyISAM`, `MEMORY`, `InnoDB`, `BDB`, and so on
- There are two types for these engines: *transaction-safe table* (TST) and *not-transaction-safe table* (NTST)
- InnoDB and BDB belong to TST
- MyISAM, and MEMORY belong to NTST
- By default, a table has `MyISAM` storage engine
- Chapter 13 of the textbook discussed `Storage Engine`

Prepared by X. Wang

26

Dropping Database and Tables

- To drop a database, use: `DROP DATABASE DB-name;`
`DROP DATABASE bookstore;`
- To drop a table, use: `DROP TABLE table-name;`
`DROP TABLE customers;`
- Both `DROP DATABASE` and `DROP TABLE` support an optional `IF EXISTS` keyword which can be used to prevent an error being reported
`DROP DATABASE IF EXISTS bookstore;`
`DROP TABLE IF EXISTS customers;`

Prepared by X. Wang

27

Creating a User

- The `GRANT` statement is used to `create a new user` or `grant an existing user` onto resources
- An example to create a new user `abcd` with password `1234`, and grant all privileges on all databases and all tables to the user

```
GRANT ALL PRIVILEGES ON *.* TO 'abcd'@'localhost'
IDENTIFIED BY '1234';
```
- The complete syntax is listed in the MySQL manual

Prepared by X. Wang

28

Inserting Rows

- The `INSERT` statement is used to `insert a new row` into a table
- The complete syntax is listed in the MySQL manual
- An example to insert a customer into the `customers` table

```
INSERT INTO customers VALUES
(1, 'Wang', 'Xusheng', 'wangx@abc.com', MD5('abcd'),
'1001 University Dr', 'Rock Hill', 'SC', '29730',
'(803)123-4567');
```

Prepared by X. Wang

29

Inserting Rows

- If the value for an attribute is unknown or don't want provide a value for an attribute, we can specify a `NULL` for the corresponding attribute
- If only a small part of attributes have values, we can use `INSERT ... SET` statement
- For the previous example, if we only know the customers name, we can insert the row with:

```
INSERT INTO customers SET
  lastname = 'Wang', firstname = 'Xusheng';
```

Prepared by X. Wang

30

Deleting Rows

- The **DELETE** statement is used to **delete rows** from a table
- **DELETE FROM customers;**
will delete all rows from the customer table
- With a **WHERE** clause, we can remove specific rows
- **DELETE FROM customers WHERE lastname = 'Smith';**
will remove all rows for **customers** whose lastname is Smith

Prepared by X. Wang

31

Updating Rows

- The **UPDATE** statement is used to **update attribute values for certain rows** in a table
- **UPDATE customers SET state = upper(state);**
replaces the string values of all **state** attributes with the same string in uppercase. Here **upper()** is one of MySQL functions we can find it in MySQL Manual
- With a **WHERE** clause, we can update specific rows
- **UPDATE customers SET lastname = 'Smith' WHERE cust_id = 7;**
updates the **lastname** attribute of customer #7 to Smith

Prepared by X. Wang

32

Querying with SQL SELECT

- The **SELECT** statement is used to **query and retrieve one or more rows (returned as a table)** from one table or multiple tables with join in a database
- **SELECT * FROM customers;**
lists all rows with all attributes from the **customer** table
- **SELECT cust_id, lastname, firstname FROM customers;**
lists all rows with only three attributes from the customer table

Prepared by X. Wang

33

Querying with SQL SELECT

- The **SELECT** statement can also be used to output data that is not from a database, but from an arithmetic expression that may contain MySQL functions
- **SELECT curtime();**
displays the current time
- **SELECT pi()*(4*4);**
does an arithmetic calculation

Prepared by X. Wang

34

Querying with SQL SELECT

- With a **WHERE** clause, we can query specific rows
- **SELECT cust_id, lastname, firstname, city, state FROM customers WHERE state = 'SC';**
lists five attributes for the customers who live in SC
- **SELECT cust_id, lastname, firstname FROM customers WHERE lastname='Smith' AND firstname LIKE 'M%';**
lists three attributes for the customers whose **lastname** is Smith and **firstname** begins with a letter M

Prepared by X. Wang

35

Querying with SQL SELECT

- With an **ORDER BY** clause, we can sort the output
- **SELECT cust_id, lastname, firstname, city, state FROM customers WHERE state = 'SC' ORDER BY lastname, firstname;**
lists five attributes for the customers who live in SC with sorting on **lastname** and **firstname**

Prepared by X. Wang

36

Querying with SQL SELECT

- With a **GROUP BY** clause, we can create aggregations in a query

```
➤ SELECT city, COUNT(*)
   FROM customers
   GROUP BY city;
```

creates an output as

city	COUNT(*)
Alexandra	14
Armidale	7
Athlone	9
Bauple	6
Belmont	11

Prepared by X. Wang

37

Querying with SQL SELECT

- There are several functions that can be used in aggregation with the **GROUP BY** clause
- **AVG()** finds the average value of a numeric attribute in a set
- **MIN()** finds the minimum value of a string or numeric attribute in a set
- **MAX()** finds the maximum value of a string or numeric attribute in a set
- **SUM()** finds the sum total of a numeric attribute
- **COUNT()** counts the number of rows in a set

Prepared by X. Wang

38

Querying with SQL SELECT

- A **HAVING** clause permits conditional aggregation in a query

```
➤ SELECT city, COUNT(*)
   FROM customers
   GROUP BY city HAVING COUNT(*) > 10;
```

creates an output as

city	COUNT(*)
Alexandra	14
Belmont	11

Prepared by X. Wang

39

Querying with SQL SELECT

- The **HAVING** clause can do the same thing as the **WHERE** clause does

```
➤ SELECT cust_id, lastname
   FROM customers HAVING lastname = 'Marzalla';
and
```

```
SELECT cust_id, lastname
FROM customers WHERE lastname = 'Marzalla';
```

generate the same output, but the **WHERE** clause provides a better performance

Prepared by X. Wang

40

Querying with SQL SELECT

- The **DISTINCT** clause can generate output with only unique values

```
➤ SELECT DISTINCT city
   FROM customers;
```

finds out the unique cities in the customer table

```
➤ SELECT city
   FROM customers
   GROUP BY city;
```

can do the same thing

Prepared by X. Wang

41

Querying with SQL SELECT

- The **LIMIT** operator is MySQL-specific and is used to control the size of the output

```
➤ SELECT * FROM customers LIMIT 5;
only returns the first 5 rows
```

- We can also specify which row to begin at, and how many rows to be returned

```
➤ SELECT * FROM customers LIMIT 100, 5;
returns the 100th and 104th rows
```

- Row numbering begins at 0

- The **LIMIT** operator is included at the end of an SQL statement

Prepared by X. Wang

42

Join Queries

- So far our queries were only from *one table*
- Sometimes we want to have a query that is related to two or more tables
- For example, a query to find all customers who have placed orders, requires two tables *customers* and *orders*
- A query that matches rows from two or more tables is called *join query*
- To finish a join query, we need to use *primary key* and *foreign keys* in related tables in the *WHERE* clause

Prepared by X. Wang

43

Join Queries

- The following statement creates the *orders* table

```
CREATE TABLE orders (
  cust_id int(5) NOT NULL,
  order_id int(8) NOT NULL auto_increment,
  orderdate datetime,
  creditcard char(16),
  expirydate char(4),
  emp_id int(5),
  confirmdate datetime,
  PRIMARY KEY (order_id),
  FOREIGN KEY (cust_id) REFERENCES customers(cust_id)
  ON DELETE CASCADE,
  FOREIGN KEY (emp_id) REFERENCES employee(emp_id)
) ENGINE=MyISAM;
```

Prepared by X. Wang

44

Join Queries

- The following join query (find the customers who have placed orders) lists all possible combinations between *customers* and *orders* tables
- ```
SELECT lastname, firstname, order_id, orderdate
FROM customers, orders
```
- A join query *without conditions* will generate the *Cartesian Product* (list all combinations of records from two tables)

Prepared by X. Wang

45

## Join Queries

- To create a reasonable join, we can use the *NATURAL JOIN* operation

```
SELECT lastname, firstname, order_id, oederdate
FROM customers NATURAL JOIN orders
```

- The *NATURAL JOIN* operator requires both tables have the same attribute name
- For example, both *customers* and *orders* tables have the same attribute name – *cust\_id*
- If no same attribute name in two tables, a *Cartesian Product* will be generated

Prepared by X. Wang

46

## Join Queries

- To create a join query, we can also explicitly specify the conditions in the *WHERE* clause
- ```
SELECT lastname, firstname, order_id, orderdate
FROM customers, orders
WHERE customers.cust_id = orders.cust_id;
```
- This join query will generate the same result as the previous *Natural Join* query
 - Specifying the join conditions explicitly in the *WHERE* clause is *safer* and *clearer* than the *NATURAL JOIN* operator

Prepared by X. Wang

47

Join Queries

- The join query can be applied on more than two tables
- The following statement creates the *items* table

```
CREATE TABLE items (
  order_id int(8) NOT NULL,
  item_no int(3) NOT NULL,
  quantity int(3) NOT NULL,
  book_id int(8) NOT NULL,
  dateadded datetime,
  PRIMARY KEY (order_id,item_no),
  FOREIGN KEY (order_id) REFERENCES orders(order_id)
  ON DELETE CASCADE,
  FOREIGN KEY (book_id) REFERENCES books(book_id)
) ENGINE=MyISAM;
```

Prepared by X. Wang

48

Join Queries

- A query: find the orders and the corresponding items for the customer whose cust_id is 2
- The following statement will create the result

```
SELECT lastname,firstname,orderdate,quantity,  
       book_id  
FROM customers, orders, items  
WHERE customers.cust_id = 2 AND  
       customers.cust_id = orders.cust_id AND  
       orders.order_id = items.order_id;
```

Prepared by X. Wang

49

SQL Script File

- We can interactively execute SQL statements one by one in the Command Interpreter
- The Command Interpreter also allows executing an SQL Script File
- An SQL Script File contains all SQL statements that will be executed in order when the file is run from the Command Interpreter
- An SQL Script File usually names with an extension .sql
- To execute an SQL Script File, we use:

```
source SQL-Script-File-name;
```

Prepared by X. Wang

50