Mastering MySQL

COMP-3077 Web-Based Data Management

Dr. Saja AL-Mamoori

Joining Tables Together

- *customers* table needs to be able to be cross-referenced with publications purchased from the *classics* table.
- CREATE TABLE customers (

  name VARCHAR(128),

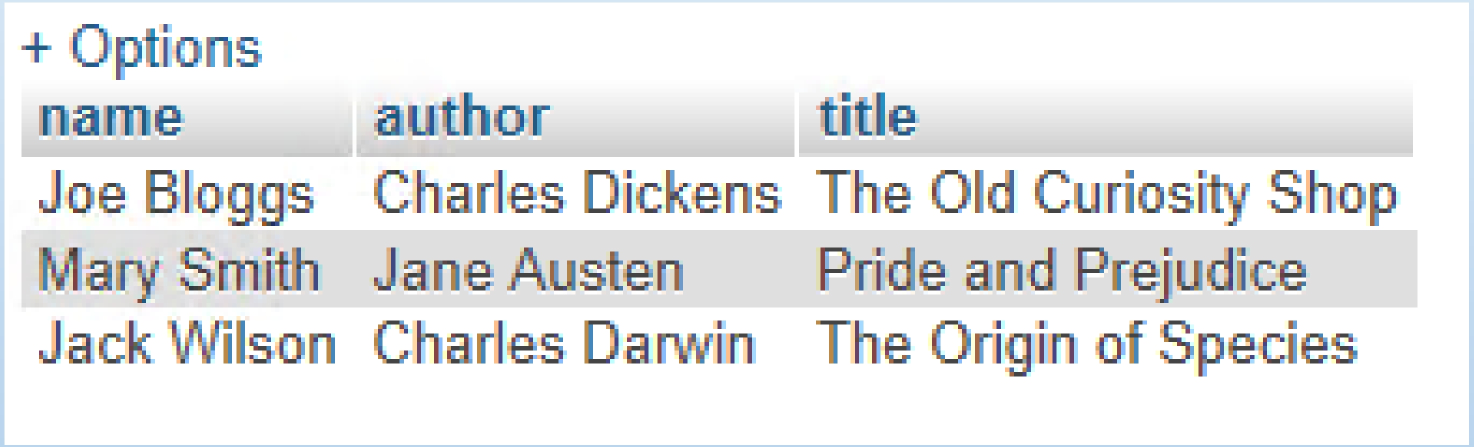
  isbn VARCHAR(13),

  PRIMARY KEY (isbn)) ENGINE MyISAM;
- INSERT INTO customers(name,isbn) VALUES('Joe Bloggs','9780099533474');
- INSERT INTO customers(name,isbn) VALUES('Mary Smith','9780582506206');
- INSERT INTO customers(name,isbn) VALUES('Jack Wilson','9780517123201');
- SELECT * FROM customers;

Joining Tables Together

- Of course, in a proper table containing customers' details there would also be addresses, phone numbers, email addresses, and so on, but they aren't necessary for this explanation.
- SELECT name, author, title from customers, classics
- WHERE customers.isbn=classics.isbn;
- Or **NATURAL JOIN** to join them automatically:
- SELECT name, author, title FROM customers NATURAL JOIN classics;

Joining Tables Together


+ Options

name	author	title
Joe Bloggs	Charles Dickens	The Old Curiosity Shop
Mary Smith	Jane Austen	Pride and Prejudice
Jack Wilson	Charles Darwin	The Origin of Species

Using Logical Operators

- You can also use the logical operators AND, OR, and NOT in your MySQL WHERE queries to further narrow down your selections.
- SELECT author,title FROM classics WHERE author LIKE "Charles%" AND author LIKE "%Darwin";
- SELECT author, title FROM classics WHERE author LIKE "%Mark Twain%" OR author LIKE "%Samuel Langhorne Clemens%";
- SELECT author,title FROM classics WHERE author LIKE "Charles%" AND author NOT LIKE "%Darwin";
- % means anything (zero or more characters)

Database Design

- It’s very important that you design a database correctly before you start to create it;
- Otherwise, you are almost certainly going to have to go back and change it by:
  - splitting up some tables
  - merging others,
  - moving various columns in order to achieve sensible relationships that MySQL can easily use.

online bookstore’s database

- How many authors, books, and customers are in the database?
- Which author wrote a certain book?
- Which books were written by a certain author?
- What is the most expensive book?
- What is the best-selling book?
- Which books have not sold this year?
- Which books did a certain customer buy?
- Which books have been purchased along with the same other books?

Simple rules

1- When you plan to do a lot of searches on something, it can often benefit by having its own table.

2- when couplings between things are **loose**, it’s best to put them in separate tables.

- books and ISBNs can probably be combined into one table, because they are closely linked.
- Books and customers should be in separate tables, because their connection is very loose.

Defining Tables

- *Authors*

There will be lots of searches for authors, many of whom have collaborated on titles, and many of whom will be featured in collections. Listing all the information about each author together, linked to that author, will produce optimal results for searches —hence an *Authors* table.

- *Books*

Many books appear in different editions. Sometimes they change publisher and sometimes they have the same titles as other, unrelated books. So the links between books and authors are complicated enough to call for a separate table.

- *Customers*

It’s even more clear why customers should get their own table, as they are free to purchase any book by any author.

Primary Keys: The Keys to Relational Databases

- How to link between 2 tables?
- Who wrote each book? (Books – Authors)
- Who purchased it (Books – Customers)
- We need an ***unique*** identifier for each table (the *primary key*)
- The **ISBN** is a rare case for which an industry has provided a primary key that you can rely on to be unique for each product.
- For Primary key you may consider creating a primary key with AUTO_INCREMENT

Normalization

- The process of **separating** your data into tables and **creating primary keys**.
- Its main goal is to make sure each piece of information appears in the database only once, (Duplicating data is inefficient) ?
  1. because it makes databases larger than they need to be => slows access.
  2. more important, the presence of duplicates creates a strong risk that you'll update only one row of duplicated data =>inconsistency =>errors.
- It's important not to go too far and create more tables than is necessary, which would also lead to inefficient design and slower access.

Normalizaion

- *Example:*
- If you list the titles of books in the ***Authors* table** as well as the ***Books* table**
- and you have to correct a typographic error in a title?
- you'll have to search through both tables and make sure you make the same change every place the title is listed.

It's better to **keep the title in one place** and use the ISBN in other places

Normalization

- 3 different forms of normalization:
  1. First Normalization
  2. Second Normalization
  3. Third Normalization
- If you modify a database to satisfy each of these forms in order, you will ensure that your database is :
  1. optimally balanced for fast access
  2. minimum memory usage
  3. disk space usage

highly inefficient design for a database table

Author1	**Author2**	**Title**	**ISBN**	**Price $US**	**Customer Name**	**Customer Address**	**Purchase Date**
David Sklar	Adam Trachtenberg	PHP Cookbook	596101015	44.99	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Danny Goodman		Dynamic HTML	596527403	9.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Hugh E	Williams David	lane PHP And MySQL	596005436	44.95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
David Sklar	Adam Trachtenberg	PHP Cookbook	596101015	44.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Rasmus Lerdorf	Kevin Tatroe & Peter MacIntyre	Programming PHP	596006815	39.99	David Miller	3647 Cedar Lane,Waltham, MA 02154	Jan 16 2009

First Normal Form

- For a database to satisfy the *First Normal Form*, it must fulfill three requirements:

1- There should be no repeating columns containing the same kind of data. (repeated Author Columns violate this rule)

2- All columns should contain a single value. (Kevin Tatroe & Peter MacIntyre in Author2 violate this rule).

3- There should be a primary key to uniquely identify each row. (satisfied , by having ISBN).

First Normal Form

Title	ISBN	Price $US	Customer Name	Customer Address	Purchase Date
PHP Cookbook	596101015	44.99	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Dynamic HTML	596527403	9.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
lane PHP And MySQL	596005436	44.95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
PHP Cookbook	596101015	44.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Programming PHP	596006815	39.99	David Miller	3647 Cedar Lane,Waltham, MA 02154	Jan 16 2009

ISBN	Author
596101015	David Sklar
596101015	Adam Trachtenberg
596527403	Danny Goodman
596005436	Hugh E Williams
596005436	David Lane
596006815	Rasmus Lerdorf
596006815	Kevin Tatroe
596006815	Peter MacIntyre

First Normal Form

Title	ISBN	Price $US	Customer Name	Customer Address	Purchase Date
PHP Cookbook	596101015	44.99	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Dynamic HTML	596527403	9.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
lane PHP And MySQL	596005436	44.95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
PHP Cookbook	596101015	44.99	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Programming PHP	596006815	39.99	David Miller	3647 Cedar Lane,Waltham, MA 02154	Jan 16 2009

Author ID	ISBN	Author
1	596101015	David Sklar
2	596101015	Adam Trachtenberg
3	596527403	Danny Goodman
4	596005436	Hugh E Williams
5	596005436	David Lane
6	596006815	Rasmus Le[illegible]
7	596006815	Kevin Tatroe
8	596006815	Peter MacIntyre

Second Normal Form

- The First Normal Form deals with duplicate data (or redundancy) across multiple columns.
- The *Second Normal Form* is all about redundancy across multiple rows.
- Second Normal Form, your tables must already be in First Normal Form.
- Notice how Darren Ryder with the same address (customer) have bought 2 books (purchases)

Second Normal Form

Title	ISBN	Price $US	Customer Name	Customer Address	Purchase Date
PHP Cookbook	596101015	44.99	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Mar 03 2009
Dynamic HTML	596527403	[illegible]	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
lane PHP And MySQL	596005436	[illegible].95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Jun 22 2009
PHP Cookbook	59610101[illegible]	[illegible]9	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Dec 19 2008
Programming PHP	5960068[illegible]	39.99	David Miller	3647 Cedar Lane,Waltham, MA 02154	Jan 16 2009

The new Books table

Title	ISBN	Price $US
PHP Cookbook	596101015	44.99
Dynamic HTML	596527403	9.99
lane PHP And MySQL	596005436	44.95
PHP Cookbook	596101015	44.99
Programming PHP	596006815	39.99

The new Purchases table

CustNo	ISBN	Date
1	596101015	Mar 03 2009
2	596527403	Dec 19 2008
2	596101015	Dec 19 2008
3	596005436	Jun 22 2009
4	596006815	Jan 16 2009

The new Customers table

CustNo	Name	Address	City	State	Zip
1	Emma Brown	1565 Rainbow Road	Los Angeles	CA	9001
2	Darren Ryder	4758 Emily Drive	Richmond	VA	2321
3	Earl B. Thurston	862 Gregory Lane	Frankfort	KY	4060
4	David Miller	3647 Cedar Lane	Waltham	MA	215

Third Normal Form

- The tables must complies with both the First and Second Normal Forms.
- The data that is *not* directly dependent on the primary key but *is* dependent on another value in the table should also be moved into separate tables.

The 3rd Normal Form Customers table

CustNo	Name	Address	Zip
1	Emma Brown	1565 Rainbow Road	90014
2	Darren Ryder	4758 Emily Drive	23219
3	Earl B. Thurston	862 Gregory Lane	40601
4	David Miller	3647 Cedar Lane	2154

The 3rd … Zip codes

Zip	CityID
90014	1234
23219	5678
40601	4321
2154	8765

The 3rd Normal Form cities table

CityID	Name	StateID
1234	Los Angeles	5
5678	Richmond	46
4321	Frankfort	17
8765	Waltham	21

The 3rd Normal Form states table

StateID	Name	Abbreviation
5	California	CA
46	Virginia	VA
17	Kentucky	KY
21	Massachusetts	MA

When Not to Use Normalization

- **high-traffic sites**: you should never fully normalize your tables on sites that will cause MySQL to thrash.
- Normalization requires spreading data across multiple tables, and this **means making multiple calls to MySQL for each query**.
- Few dozens of concurrent users can call hundred of concurrent access to database tables => in this case de-normalize to reduce the number of lookup tables .
- Of course, you have to deal with the downsides previously mentioned, such as using up large amounts of disk space, and ensuring that you update every single duplicate copy of data when one of them needs modifying.

Relationships

- MySQL is called a *relational* database management system because its tables *relationships between data alongside the data*.
- There are three categories of relationships:

1. One-to-One
2. One-to-Many
3. Many-to-Many

One-to-One

- It is like a (traditional) marriage: each item has a relationship to only one item of the other type.
- Its rare.
- Example in this chapter *The 3rd Normal Form states table*
- Usually, when two items have a one-to-one relationship, you just include them as columns
- in the same table. There are two reasons for splitting them into separate tables:

1. You want to be prepared in case the relationship changes later.
2. The table has a lot of columns, and you think that performance or maintenance would be improved by splitting it.

One-to-many

- *One-to-many* (or many-to-one) relationships occur when one row in one table is linked to many rows in another table.
- Example *The 3rd Normal Form Customers table* ***with*** *The new Purchases table*

One-to-many

The 3rd Normal Form Customers table

CustNo	Name	Address	Zip
1	Emma Brown	1565 Rainbow Road	90014
2	Darren Ryder	4758 Emily Drive	23219
3	Earl B. Thurston	862 Gregory Lane	40601
4	David Miller	3647 Cedar Lane	2154

The new Purchases table

CustNo	ISBN	Date
1	596101015	Mar 03 2009
2	596527403	Dec 19 2008
2	596101015	Dec 19 2008
3	596005436	Jun 22 2009
4	596006815	Jan 16 2009

Databases and Anonymity

- An interesting aspect of using relations is that you can accumulate a lot of information about some item—such as a customer—without actually knowing who that customer is.
- Note that in the previous example we went from customers’ zip codes to customers’ purchases, and back again, without finding out the name of a customer.
- Databases can be used to track people, but they can also be used to help preserve people’s privacy while still finding useful information.

Transactions

- Suppose that you are creating a sequence of queries to transfer funds from one bank account to another. You would not want either of the following events to occur:
  - You add the funds to the second account, but when you try to subtract them from the first account, the update fails, and now both accounts have the funds.
  - You subtract the funds from the first bank account, but the update request to add them to the second account fails, and the funds have disappeared into thin air.

Transactions

- Transactions allow concurrent access to a database by many users or programs at the same time.
- MySQL handles this seamlessly by ensuring that all transactions are queued and that users or programs take their turns and don’t tread on each other’s toes.
- The transaction (deducting from first then adding to the second account) will fully successfully completed or fully rejected.

Transaction Storage Engines

- To be able to use MySQL’s transaction facility, you have to be using MySQL’s *InnoDB* storage engine.

-
  ```
  CREATE TABLE accounts (
  number INT,
  balance FLOAT,
  PRIMARY KEY(number)
  ) ENGINE InnoDB;
  ```
- `DESCRIBE accounts;`

Transaction Storage Engines

- INSERT INTO accounts(number, balance) VALUES(12345, 1025.50);
- INSERT INTO accounts(number, balance) VALUES(67890, 140.00);
- SELECT * FROM accounts;
- Transactions in MySQL start with either a BEGIN or a START TRANSACTION statement.
- BEGIN;
- UPDATE accounts SET balance=balance-250 WHERE number=12345;
- UPDATE accounts SET balance=balance+250 WHERE number=67890;
- COMMIT;
- SELECT * FROM accounts;

Using ROLLBACK

- Using the ROLLBACK command, you can tell MySQL to forget all the queries made since the start of a transaction and to end the transaction.
- BEGIN;
- UPDATE accounts SET balance=balance-250 WHERE number=12345;
- UPDATE accounts SET balance=balance+250 WHERE number=67890;
- SELECT * FROM accounts;
- ROLLBACK;
- SELECT * FROM accounts;

References

- Learning PHP, MySQL & JavaScript, 4th Edition. With jQuery, CSS & HTML5, by Robin Nixon, O'Reilly Media, 2014, 978-1-4919-1866-1, chapters 8 - 9