INFO6210 Data Management and Database Design Final Project : Hands-on Learning Materials to Teach SQL

(Weibo Dai, Chuhong Yu)

Abstract

In this tutorial, we are going to discuss topics namely,

1. What is database
2. What is relational database
3. What is SQL
4. What is MySQL
5. Installation of MySQL and PyCharm (Set up the environment)
6. Frequently used Data type in MySQL database
7. Basic operations on MySQL database
8. Advanced operations on MySQL database
9. ER Models
10. Database normalizations
11. Database transactions
12. Hands on MySQL practices.

We will provide some hands-on practices on database queries at the end of the courses. After going through the entire tutorial, we believe that you will have a clear understanding of Relational database and how the basic concepts of database could be applied in the real world. You would also be able to write some SQL queries on your own. Let's get started.

1. What is database

A database (Abbreviation: DB is a storage of collected electronic data[1], information or documents and it is organized based on specific logics, rules or inner relationship of the collections. A database usually occurs with a specific database managing system (Abbreviation: DBMS) or software, by which a controller or user can interact with the collected data, namely adding, deleting, searching, updating and so on . Usually, the database and its managing system as well as the associated software, are usually called as a database system, or just database.[2] So, when someone mentions a database, that often refers to all of them mentioned above. Among the most popular models for database, hierarchical, network and the relational data model[3], we will focus on the relational data model.

2. What is relational Database

A relational database is one of the most popular types of database that is built by relational data model. It provides an efficient and accessible way for users to store and identify data through the inner relationship with other data.[4] The structure of relational database is usually organized by tables that contain columns and rows, where the detail information of data is stored. The management system for relational database is called relational database management system, (Abbreviation: RDBMS) and serves as the basis for many database-managing software such as MySQL, Oracle and Microsoft Access.

3. What is SQL?

The SQL, which stands for Structured Query Language, is a primary language that allows users to access and interact with the database. It became a standard of the ANSI (the American National Standards Institute) in 1986.With the help of SQL, users can easily add, delete, update, retrieve data for further processes like transaction, statistic analytics, and to manage all aspects of the database.[5]

4.What is MySQL?

MySQL is a free and open-source software for relational database management system and is mostly used for web database.[6] In any database, using tables, primary keys & foreign keys, constraints, triggers, roles, etc., user can perform the database well while with no violation to its referential integrity between two related tables. When compared with Oracle, another famous RDBMS software, MySQL is free and offers basic commands that are enough for most of the users while it do have limitation on supporting files, character types, output analysis and high level of commands. But for users with enough budget and seeking RDBMS that offers extensive commands, more data type availability or strong tools, the Oracle maybe the better choice.[7]

5. Set up the environment and all the tools we need in this tutorial

1. Set up PyCharm

In this tutorial, we use PyCharm to run all the code, so you first need to go to the following link to download PyCharm, follow the installation guide step by step and then you would be able to install it. For this tutorial, the community version is good enough to perform all the operations, but you could download the professional version if you want. (https://www.jetbrains.com/pycharm/)
After installing PyCharm, go to the project terminal, type

"pip install mysql-connector-python"

and run it. We assume that the pip installation engine is already installed in your computer, if not, the following link would lead you to install pip first. (https://pip.pypa.io/en/stable/installing/.)

2. Set up MySQL

(1) Download and install MySQL Community Server (https://dev.mysql.com/downloads/mysql/)

(2) During the installation setup, you will be prompted for a "root" password in the server configuration step.

(3) Download and install MySQL Workbench. The Workbench is GUI tool that allows us to manage your MySQL database. (https://dev.mysql.com/downloads/workbench/.)

(4) Launch workbench, at the home page, setup a new connection profile with the configuration (Connection method: Standard (TCP/IP), Hostname: 127.0.0.1,Port:3306,Username: root, Password: yourpassword) and test your connection.

(5) Double click on your local instance and it should bring you the schemas view where you can see all your databases and tables.

6. Frequently used datatype in SQL

The following are some basic SQL datatype:

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data

7. Basic CRUD operations on MySQL database

Firstly, the CRUD operations stand for Create, Retrieve, Update and Delete. In order to demonstrate the basic operations on jupyter notebook, we first need to install the MySQL connector and set up the connections on jupyter notebook. After all the environment is set up, we could then dive into the

coding part. First, we need to import the MySQL connector library. Secondly, we need to provide the connector with the host name, username and password. If the connection is set up, we would get the memory address of the connection object printed.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  passwd="12345678"
)

print(mydb)
```

If we want to manipulate the database, we first need to get a cursor object. The cursor() function in connection object could be understood as a pointer that we would take advantage of to manipulate our database.

7.1 Create

We create a database called 'test'. In MySQL database , a database is the highest schema, it is a collection of tables.

```
mycursor.execute("create database test")
```

We create a table in the test database. The table in MySQL database contains the information that stored in database. There are id, name and address three fields in the database. The id is the primary key. The type of id is int. Both the type of name and the address type are varchar. The varchar(255) simply means that the length of the field could be 255 bytes. The Auto_increment means that the value of id could be assigned by the MySQL workbench and doesn't need to be specifically assigned.

```
mycursor.execute("CREATE TABLE test.customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255),  address VARCHAR(255))")
```

7.2 Insert

Insert information into the database.

There are two ways here to insert data into the database in python using MySQL connector. The first one is directly insert. We first specify which table we want to insert, and then provides the value. If the insertion is a success, we could see that the record is inserted.

```
In [28]: sql = "INSERT INTO test.customers (name, address) VALUES ('Joy', 'Canada')"

         mycursor.execute(sql)

         # the commit here means to persist the data into database. We could demonstrate it clearly in the following chapters.
         mydb.commit()

         print(mycursor.rowcount, "record(s) inserted.")
```

The second way to insert is to use prepared statement. We first use the placeholder to take the place of the values we want to insert, and then provide the placeholder with the specific value.

```
In [30]: sql = "INSERT INTO test.customers (name, address) VALUES (%s, %s)"
         val = ("David", "California")
         mycursor.execute(sql, val)

         mydb.commit()

         print(mycursor.rowcount, "record(s) inserted.")

         1 record(s) inserted.
```

7.3 Select

Retrieve information from the database.

(1) Retrieve all the information in one table [20]

The 'Select' keyword means picking certain fields in the table. The * means that select all columns in the table.

In [21]:
```
sql = "SELECT * FROM test.customers"
mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
    print(x)
```

```
(2, 'Joy', 'Canada')
(3, 'Helen', 'Beijing')
(4, 'ben', 'England')
(6, 'Alfred Schmidt', 'karamay')
(7, 'Helen', 'Beijing')
(8, 'ben', 'England')
(9, 'Amy', 'Boston')
(10, 'Li', 'San Francisco')
```

(2) Retrieve one specific field from the database

We could use where statement to select one specific field from the table.

In [23]:
```
sql = "SELECT * FROM test.customers WHERE address = %s"
adr = ("Beijing", )
mycursor.execute(sql, adr)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(3, 'Helen', 'Beijing')
(7, 'Helen', 'Beijing')
```

(3) Retrieve one specific column from the database

Instead of using *, we could specify which column we need to retrieve from the database.

In [5]:
```
sql = "SELECT name FROM test.customers"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
('Joy',)
('Helen',)
('ben',)
('Li',)
('Helen',)
('ben',)
('Amy',)
('Li',)
```

(4) Select distinct name from the database

In [6]:
```
sql = "SELECT distinct name FROM test.customers"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
('Joy',)
('Helen',)
('ben',)
('Li',)
('Amy',)
```

(5) And, or, and not operators in select statement

And operation means that only when both conditions are met will the query be executed

```
sql = "SELECT * FROM test.customers \
WHERE name='joy' AND address='Canada';"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(2, 'Joy', 'Canada')
```

Or operation means that either the condition is met will the fields be fetched

```
sql = "SELECT * FROM test.customers \
WHERE name='joy' or address='San Francisco';"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(2, 'Joy', 'Canada')
(6, 'Li', 'San Francisco')
(10, 'Li', 'San Francisco')
```

Not operation means that only when the fields are not met will the fields be matched

```
sql = "SELECT * FROM test.customers \
WHERE not address='San Francisco';"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(2, 'Joy', 'Canada')
(3, 'Helen', 'Beijing')
(4, 'ben', 'England')
(7, 'Helen', 'Beijing')
(8, 'ben', 'England')
(9, 'Amy', 'Boston')
```

(6)Adding more data for following demonstrations

```
sql = "INSERT INTO test.customers (name, address) VALUES (%s, %s)"
val = [("Helen", "Beijing"),
       ("ben", "England"),
       ("Amy", "Boston"),
       ("Li", "San Francisco"),

      ]
mycursor.executemany(sql, val)

mydb.commit()
```

(7) Limit the search result by adding limit

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM test.customers LIMIT 5")

myresult = mycursor.fetchall()

for x in myresult:
    print(x)
```

(8) Sort the data in descending order

```
sql = "SELECT * FROM test.customers ORDER by name desc"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
     print(x)
```

```
(6, 'Li', 'San Francisco')
(10, 'Li', 'San Francisco')
(2, 'Joy', 'Canada')
(3, 'Helen', 'Beijing')
(7, 'Helen', 'Beijing')
(4, 'ben', 'England')
(8, 'ben', 'England')
(9, 'Amy', 'Boston')
```

(9) The COUNT() function

The COUNT() function returns the number of rows that matches a specified criterion. The below statement counts the number of products whose price are more than 1000 USD.

```
sql = "SELECT COUNT(name)\
FROM test.products where price > 1000"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(1,)
```

(10) The AVG() function

The AVG() function returns the average value of a numeric column. The below SQL calculates the

average price in product table.

```
sql = "SELECT AVG(price)\
FROM test.products;"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(1166.6666666666667,)
```

(11)The SUM() function

The SUM() function returns the total sum of a numeric column. The below SQL calculate the total price in product table.

```
sql = "SELECT SUM(price)\
FROM test.products;"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(3500.0,)
```

(12) Group by

The GROUP BY statement groups rows that have the same values into summary rows, the following example demonstrate how many customers are there in each address.

```
sql = "SELECT COUNT(name), address FROM test.customers GROUP BY address;"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(2, 'Canada')
(2, 'Beijing')
(2, 'England')
(1, 'karamay')
(1, 'Boston')
(1, 'San Francisco')
(1, 'California')
```

7.4. Delete a record in one table

The below statement deletes a record whose address is England

```
In [5]: mycursor = mydb.cursor()
        sql = "DELETE FROM test.customers WHERE address = 'England'"

        mycursor.execute(sql)
        mydb.commit()
        print(mycursor.rowcount, "record(s) deleted.")
```

```
2 record(s) deleted.
```

7.5 Update

Update one certain field with given name and address

```
mycursor = mydb.cursor()
sql = "UPDATE test.customers SET name = 'Alfred Schmidt' ,address= 'karamay' WHERE id = 9"

mycursor.execute(sql)
# mydb.commit()
print(mycursor.rowcount, "record(s) updated.")
```

```
1 record(s) updated.
```

We could see the results below.

```
sql = "SELECT * FROM test.customers"
mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
    print(x)
```

```
(2, 'Joy', 'Canada')
(3, 'Helen', 'Beijing')
(4, 'ben', 'England')
(6, 'Li', 'San Francisco')
(7, 'Helen', 'Beijing')
(8, 'Alfred Schmidt', 'karamay')
(9, 'Alfred Schmidt', 'karamay')
(10, 'Li', 'San Francisco')
```

8. Advanced SQL queries

8.1 Join

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. In order to demonstrate further, we need to create two more tables to demonstrate the following queries. One is called products storing information such as product_id, name of product and their prices. Another table would be orders, storing all orders with each containing order_id, product_id and customer_id.

Create products table

```
mycursor = mydb.cursor()
sql = "CREATE TABLE test.products (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), price VARCHAR(255))"
mycursor.execute(sql)
```

Insert information into the database table

```
mycursor = mydb.cursor()
sql = "INSERT INTO test.products (name, price) VALUES (%s, %s)"
val = [
    ("macbook", "2000"),
    ("iphone", "1000"),
    ("apple watch", "500")
]
mycursor.executemany(sql, val)
mydb.commit()
```

See the information of the products table

```
sql = "SELECT * FROM test.products"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
     print(x)
```

Create Orders table

```
mycursor = mydb.cursor()
sql = "CREATE TABLE test.orders (id INT AUTO_INCREMENT PRIMARY KEY, customer_id INT, product_id INT)"
mycursor.execute(sql)
```

Insert information into the orders table

```
mycursor = mydb.cursor()
sql = "INSERT INTO test.orders (customer_id, product_id) VALUES (%s, %s)"
val = [
    ("1", "1"),
    ("1", "2"),
    ("2", "3"),
    ("3", "3")
]
mycursor.executemany(sql, val)
mydb.commit()
```

After the tables are set up, we would like to perform the join operations. There are basically two types of joins. The first one is inner join, the second one is outer join. The former only shows results

in which all criteria of joining are met. We are first going to join the customer table and order table based on the customer id of two tables. The results would show the customer name and their related orders.

8.1.1 Inner join

```
mycursor = mydb.cursor()
sql = "SELECT \
    c.name, o.id\
    FROM test.customers c\
    JOIN test.orders o ON c.id = o.customer_id\
    "
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

The above result shows the result of inner join. We could see that the results didn't show the full list of customers. It only shows the matched field.

8.1.2 Inner join multiple tables.

Since the database contains 3 tables, so we might want to show a table showing order id, customer name, product name and product price, which are all information in three tables. So, we would like to select from orders table and join the other two tables based on the joining conditions.

```
sql = "SELECT \
    o.id, c.name, p.name, p.price\
    FROM test.orders o\
    JOIN test.products p ON o.product_id = p.id\
    JOIN test.customers c ON o.customer_id = c.id"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(3, 'Joy', 'apple watch', '500')
(4, 'Helen', 'apple watch', '500')
```

The above results show the field that met the join condition of products' id = orders' id and orders'customer id = customer id.

8.1.3 Outer join

We first do the left join on customer table and order table. This time we select from customer table.

```
mycursor = mydb.cursor()
sql = "SELECT \
    c.name, o.id\
    FROM test.customers c\
    LEFT JOIN test.orders o ON c.id = o.customer_id\
    "
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
('Joy', 3)
('Helen', 4)
('ben', None)
('Alfred Schmidt', None)
('Helen', None)
('ben', None)
('Amy', None)
('Li', None)
```

From the above results we could see that 'ben' and 'Li' does not have corresponding orders with them. This means that left join shows the result of the records of left table without considering whether the right table has the value. The right join would do the same thing, but it would be meaning less under these circumstances, since the customer and order is one-to-many relationship. One customer could have multiple orders, one order could only be corresponded to one customer. We would discuss entity relationship further in the next chapter.

8.2 Union

The following statement combine the name from customers table and name from products to one single column

```
sql = "SELECT name FROM test.customers union select name from test.products;"
# UNION\
# SELECT products.name\
# FROM test.products;"

mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
     print(x)
```

```
('Joy',)
('Helen',)
('ben',)
('Alfred Schmidt',)
('Amy',)
('Li',)
('macbook',)
('iphone',)
('apple watch',)
```

8.3 Wildcards

Sometimes we only know some information of the database and we would like to perform operations on the database. So, we need to use wildcard to perform the operation. The following SQL statement means that grab the results whose name ends with 'len', '%'means any number of characters.

```
mycursor = mydb.cursor()
sql = "SELECT * FROM test.customers WHERE name LIKE '%len'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(3, 'Helen', 'Beijing')
(7, 'Helen', 'Beijing')
```

The following statement selects all customers with a customer name that have "a" in the second position.

```
mycursor = mydb.cursor()
sql = "SELECT * FROM test.customers WHERE name LIKE '_a%'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
(12, 'David', 'California')
```

8.4 Nested queries [19]

There are cases when we know the information of one table, or we want to get the information of another table. Under these circumstances, we need to use nested queries to do the task.

Basic concept of subqueries

A subquery is a SQL query nested inside a larger query.

A subquery may occur in :

- A SELECT clause
- A FROM clause
- A WHERE clause

The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery. A subquery is usually added within the WHERE Clause of another SQL SELECT statement. You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.

A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select. The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query. You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:

- Compare an expression to the result of the query.
- Determine if an expression is included in the results of the query.
- Check whether the query selects any rows.[6-A]

Let's imagine the situation: we want to get the name of the customer who has bought a MacBook. To do this task, we first need to go to the customers table and find the id of this customer. Then we go to the orders table to get the product id in the orders table. Finally, we go to the products table to find the matched name of the product.

```
: sql = "select name\
        from test.customers\
        where id=(select customer_id\
                  from test.orders\
                  where product_id=(select id\
                                    from test.products\
                                    where name = 'iphone'));"

  mycursor.execute(sql)
  myresult = mycursor.fetchall()
  for x in myresult:
      print(x)

  ('Alfred Schmidt',)
```

From the above result we could see that Alfred Schmidt has ordered a iPhone.

9.Test database

For reference purpose, we show the demo database in MySQL workbench as below.

9.1 Products table

The first picture shows all the information of products table. The second picture shows the description of the products table

	id	name	price
▶	1	macbook	2000
	2	iphone	1000
	3	apple watch	500
	4	cup	4
	5	Marlboro	11
	6	lighter	1
	7	textbook	10
	8	chair	35

```
from IPython.display import Image
Image("products_describe.jpeg")
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(255)	YES		NULL	
price	varchar(255)	YES		NULL	

9.2 Customer table

id	name	address
2	Joy	Canada
3	Helen	Beijing
4	ben	England
6	Alfred Schmidt	karamay
7	Helen	Beijing
8	ben	England
9	Amy	Boston
10	Li	San Francisco

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(255)	YES		NULL	
address	varchar(255)	YES		NULL	

9.3 Order table

id	customer_id	product_id
1	1	1
2	1	2
3	2	3
4	3	3
5	3	5
6	6	7

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
customer_id	int(11)	YES		NULL	
product_id	int(11)	YES		NULL	

10. Introduction to ER diagrams & entities

10.1 What is ER diagram

The ER diagram stands for Entity Relationship Diagram, a diagram that shows the inner relationship of entities stored in the database.[8]

10.2 What is entity

Entity, according to the Cambridge English Dictionary, is "...something that exists apart from other things, having its own independent existence.".[9] Similarly, Entity in real database can be a physical thing such as manager in a Manager database, teacher in a Faculty database, or a conceptual one like a position, a class in related database, etc.[8] In a database, an entity shall have at least one property (also called attribute) and may have a its own table that describe its characteristics.

10.3 What is ER-Diagram and why do we need it in the Relational database

In the ER-diagram, entities are represented in rectangle box and the relationship between them(e.g. a "professor" "teaches" a "course") shall be put in the triangle in the middle of the line links the two entities. Any attributes related to the entities shall be put into an oval-shape box and links with the entities it describes. In general, there are three level of ER diagrams with different level of abstraction, from abstract to detailed, namely the conceptual data model, logical data model and the physical data model.[10]

10.4 What is Conceptual Data Model

The conceptual data model is the basic entity-relationship model that usually contains the least details of the relationships between and attributes contained by interested entities[11]. A conceptual model usually serves as the first step-stone for the development of other models, such as the logical data model and physical data model. However, the conceptual model is not a prerequisite for the development of other models.

10.5 What is Logical model

Similar to conceptual model, the logical data model contains more detailed information about the interested entities. Usually a logical model shall define the type of data (string, integer, varchar, etc.) for each attribute and the numerical relationship (also called cardinality) between entities, such as one-to-one relationship, one-to-many relationship, etc. Besides, normalization process may involve in this model while the definition of primary key or foreign key may not be necessary[12]. Moreover, for each different database system, the logical model shall be built uniquely with independency and cannot simply copy from another database.[11]

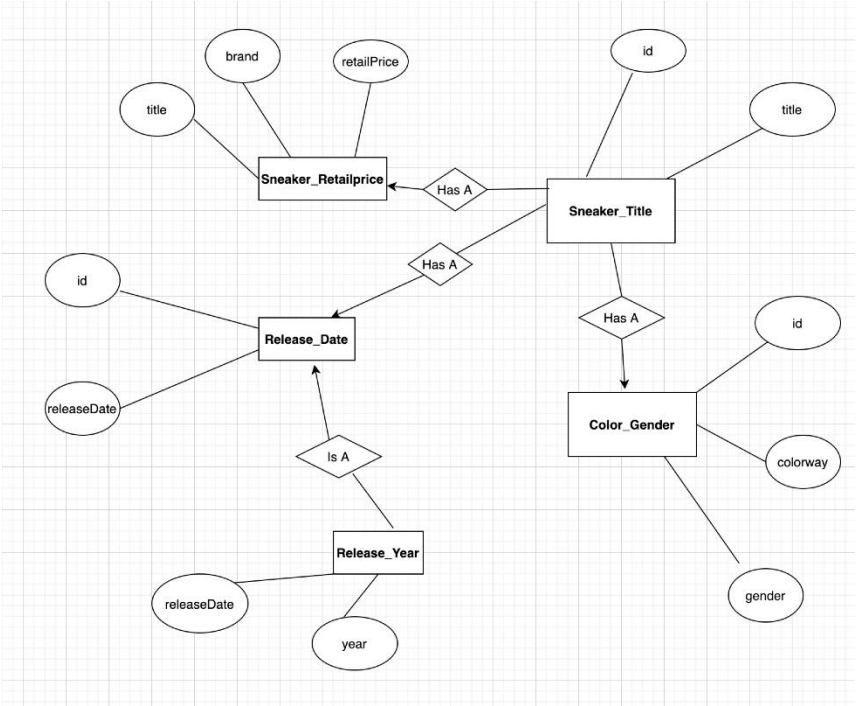
10.6 What is physical model

A physical data model contains the most detailed information about entities and the relationship between them. Because of the thorough details, the unique structure for each database as well as its accordance with normalization rules, the well-developed physical can be regarded as an instantiation of a database.[12] In a physical model, the numerical relationship (cardinality) is required for each table and the data types, length and nullability or even default values are necessities[11]. Compared with other model, the physical one requires specific and defined primary key, foreign key and offers further functions like index, views and so on[12].

10.7 What is UML graph

UML is a common standard language for specifying, visualizing, constructing, and recording software system artifacts, and is a relatively general modeling language. UML is a visual language rather than a programming language. The main purpose of UML is to define a standard method for presenting it in visual system design[13]. We use UML diagrams to describe the behavior and structure of the system. Here follows an example.

The UML diagram below is based on a sneaker database. In this database we have 5 entities, namely Sneaker_Title, Color Gender, Sneaker_Retail price, Release Date and Release Year.From the UML model graph above we could see that the Release_Year 'IS A' Release_Date. Release_Year inherited from the Release_Date, and thus is a Release_Date. For a single Sneaker, it has the attributes of Color_Gender, Release_Date and Sneaker_Retailprice. Thus, the Sneaker_Title 'Has A' Sneaker_Retailprice, Release_Date and Color_gender.



10.8 What is primary key and foreign key

A primary key-foreign key relationship defines a one-to-many relationship between two tables in a relational database. A foreign key is a column or a set of columns in one table that references the

primary key columns in another table. The primary key is defined as a column (or set of columns) where each value is unique and identifies a single row of the table.[14]

11. Database normalization

11.1 Why do we need database normalization?

Normalization is a technique for organizing data in a database. It is important that a database is normalized to minimize redundancy (duplicate data) and to ensure only related data is stored in each table. It also prevents any issues stemming from database modifications such as insertions, deletions, and updates.[15]

11.2 The basic concepts of database normalization

According to Wikipedia, database normalization is a process of "...structuring a relational database accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity..."[15]. The normalization process involves several kinds of key, columns storing attributes and even the relationship of table or its structure. Different normal form has different requirements, we now list from the first to the third normal form.[16]

First normal form (1NF) requirement[16]:

- Each table has a primary key: minimal set of attributes which can uniquely identify a record.
- The value in each column of a table are atomic (No multi-value attributes allowed).
- No repeating groups: two columns do not store similar information in the same table.

Second normal form (2NF) requirement [16]:

- All requirement for 1st normal form must be meet.
- No partial dependencies. 3.No calculated data.

Third normal form (3NF) requirement [16]:

- All requirement for 2nd normal form must be meet.
- Eliminate fields that do not directly depend on the primary key, no transitive dependencies.

12. Database Transaction

12.1 Why do we need database transaction

The primary benefit of using transactions is data integrity. Many databases use require storing data to multiple tables, or multiple rows to the same table in order to maintain a consistent data set. Using transactions ensures that other connections to the same database see either all the updates or none of them.

12.2 The basic concepts of database transaction

According to techopedia.com, a transaction in database is "...a logical unit that is independently executed for retrieval or updates...". In any relational database, the transaction "...must be atomic, consistent, isolated and durable...".[17] The atomicity requires a transaction must be the smallest part of a work and can only be done with fully completion, saved or completely undone (like rolling back). The consistency indicates that no transaction can violate the database's constraints, such as

inputting a varchar into a cell that only for integer values. The third requirement, namely isolation means the transaction cell shall not be available for other attempts until its transaction work is completed or undone.
The last requirement, the durability requires the availability of change in data, even in an event of database failure.[17]
Several terminologies for transaction:

- start, input: start transaction.
- end, input: end transaction.
- commit, input: commit transaction.
- rollback, input: rollback transaction.

If any DML query (like insert, update, delete, etc.) is executed successfully, the following commit transaction will change the data in the database while following a rollback transaction won't change the data stored.[18]

13. Self-assessment

In this section we provided some hands-on questions on SQL queries to give the reader a chance to enhance what they learned in this tutorial. The question would be based on the test database in this tutorial. We would provide the answers directly for references.

Question 1:
Find all customers in customer table in alphabetic descending order.

Question 2:
Find out all the different addresses in customers table.

Question 3:
Find all the products in products table whose price is greater than 500 and the name is end with 'book'.

Question 4:
Find all customers named Joy Helen and ben.

Question 5:
Find out the number of customers.

Question 6:
Find out the customer whose address is end with 'may'.

Question 7:
Count how many orders been made on each product by customers.

Question 8:
Select the product id from the orders table which is ordered by a customer live in Karamay.

Question 9:

Select the name from product table which is ordered by a customer lives in Karamay.

(All answers are attached in jupyter notebook)

References

[1] Definition - What does Database (DB) mean, Techopedia, August 25, 2016
(www.techopedia.com/definition/1185/database-db)

[2] Database, Oracle
(www.oracle.com/database/what-is-database.html)

[3] Everything you need to know about (Relational) Databases, Lucas Olivera, Apr 05, 2019
(https://dev.to/lmolivera/everything-you-need-to-know-about-relational-databases-3ejl)

[4] What is a Relational Database, OmniSci
(www.omnisci.com/technical-glossary/relational-database)

[5] SQL, Amazon Web Services
(https://aws.amazon.com/relational-database/?nc1=h_ls)

[6] What is MySQL and why do I need it, 123Reg
(www.123-reg.co.uk/support/servers/what-is-mysql-and-why-do-i-need-it/)

[6-A] Understanding SQL Subqueries, w3resource
(https://www.w3resource.com/sql/subqueries/understanding-sql-subqueries.php)

[7] What's the difference between Oracle and MySQL, LingaSwamy Parandha, May 25, 2017
(https://www.quora.com/Whats-the-difference-between-Oracle-and-MySQL)

[8] What is an Entity Relationship Diagram (ERD), Smartdraw
(https://www.smartdraw.com/entity-relationship-diagram/)

[9] Entity, Cambridge English Dictionary
(https://dictionary.cambridge.org/dictionary/english/entity)

[10] Chapter 8 The Entity Relationship Data Model, Adrienne Watt
(https://opentextbc.ca/dbdesign01/chapter/chapter-8-entity-relationship-model/)

[11] Entity–relationship model, Wikipedia
(https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model)

[12] What is Data Modelling? Conceptual, Logical, & Physical Data Models, Guru99
(https://www.guru99.com/data-modelling-conceptual-logical.html)

[13] What is Unified Modeling Language (UML), Visual Paradigm
(https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/)

[14] Primary Key-Foreign Key Relationships, Understanding the Physical Database Model, Oracle (https://docs.oracle.com/cd/E12100_01/books/admintool/admintool_DataModeling4.html)

[15] Database normalization, Wikipedia (https://en.wikipedia.org/wiki/Database_normalization)

[16] Normal Forms, Nick Bear Brown, Northeastern University (https://piazza.com/class/k560emam5qywd?cid=46)

[17] Transaction, Techopedia (https://www.techopedia.com/definition/16455/transaction)

[18] MySQL, detailed explanation of transaction, Xulinjiex (https://blog.csdn.net/w_linux/article/details/79666086)

[19] Nested Queries, Mike Dane, December 2017 (https://www.mikedane.com/databases/sql/nested-queries/)

[20] 10.5.6 MySQLCursor.fetchall() Method, MySQL Connector (https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-fetchall.html)

[21] Learn about class diagrams https://openclassrooms.com/en/courses/4191736-design-a-database-with-uml/4191743-learn-about-class-diagrams)

[22] Syntax errors (http://net-informations.com/python/err/eol.htm)

[23] Flowchart Maker and Online Diagram Software (https://app.diagrams.net/)

CONTRIBUTION

Contributed on my Own: 60% By referring to external source: 40%

LICENSE