

Adv Web Dev Arch

Lecture 7

CRUD and Database manipulation

Amir Amintabar, PhD

AJAX Calls

高级网页开发架构

第7讲

CRUD 操作与数据库操作

阿米尔·阿明塔巴尔，博士

AJAX 调用

CRUD and Database Manipulation

for APIs development

SQL Database related knowledge needed for the development of API servers

API 开发中的 CRUD 与数据库操作

开发 API 服务器所需的 SQL 数据库相关知识

Outline (Using Relational DB for Building an API server)

- 0- Review
- Review of SQL statements
- 1- What is a Database?
- 2- Few definitions
- 3- Design a DB by example
- 4- Creating 1:M by example
- 5- Referential integrity **constraint**
- **6- Learn SQL by Example!**
 - Query practice examples using the online hosted NorthWind DB
 - https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
 - Where clauses, like operator
- 7- Sub-queries
- 8- Creating M:M by example

大纲 (使用关系型数据库构建 API 服务器)

- 0- 复习
- SQL 语句复习
- 1- 什么是数据库?
- 2- 若干定义
- 3- 通过示例设计数据库
- 4- 通过示例创建一对多 (1:M) 关系
- **5- 参照完整性 约束**
- **6- 通过示例学习 SQL!**
 - 使用在线托管的 Northwind 数据库进行查询练习示例
 - https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
 - WHERE 子句与 LIKE 运算符
- 7- 子查询
- 8- 通过示例创建多对多 (M:M) 关系

0

Review

0

审核

Review

- setTimeout and order of execution (in what order numbers log?)
- Various web application architectures, ..., their goal ...
- Single or multiple threaded, asynchronous, none-blocking ..
- AJAX calls (Asynchronous JavaScript And XML),
- XMLHttpRequest readyState (0, 1, 2, 3, 4)
- GET, POST, the difference (data size, url, ...)
- Modules in Nodejs,
- built in modules such as http, fs ..,
- installing new modules using NPM install,
- How to host a node js app,
- how to create DB user in a cPannel shared hosting accounts,
 - giving various privileges to DB user,
- restarting a node js app every time we make changes (you can install node-mon module on your local PC that automatically restarts the app every time you save changes).
- How node js connects to DB,
- how to install MySQL engine locally,
- how to use DB admin tools such as phpMyAdmin to create DB/ tables and where to run SQL statements, how to backup and restore.

```
console.log(1);
setTimeout(function () {
    console.log(2);
}, 0);
console.log(3);
```

status of the XMLHttpRequest :
0 (UNSENT): The XHR object has been created, but open() has not been called yet.
1 (OPENED): open() has been called.
2 (HEADERS_RECEIVED): send() has been called, and the headers of the response are available.
3 (LOADING): The response is being received. As data comes in, the responseText property is updated. (3(LOADING) code be updated multiple times to this status if the server is sending a large size of response in multiple chunks)
4 (DONE): The operation is complete, and either the request has been successfully completed (status code 2xx) or an error occurred.

复习

- setTimeout 与执行顺序 (数字将以何种顺序输出?)
- 各类 Web 应用架构，……及其目标……
- 单线程或多线程、异步、非阻塞……
- AJAX 请求 (异步 JavaScript 和 XML)
- XMLHttpRequest 的 readyState (0, 1, 2, 3, 4)
- GET 与 POST 的区别 (数据大小、URL 等)
- Node.js 中的模块，
- 内置模块，例如 http、fs 等，
- 使用 npm install 安装新模块，
- 如何托管一个 Node.js 应用，
- 如何在 cPanel 共享主机账户中创建数据库用户，
 - 为数据库用户授予各种权限，
- 每次修改代码后都需要手动重启 Node.js 应用 (您可在本地计算机上安装 node-mon 模块，该模块会在您保存修改后自动重启应用)。
- Node.js 如何连接数据库，
- 如何在本地安装 MySQL 数据库引擎，
- 如何使用 phpMyAdmin 等数据库管理工具创建数据库/数据表、在何处执行 SQL 语句，以及如何进行数据库备份与恢复

```
console.log(1);
setTimeout(function () {
    console.log(2);
}, 0);
console.log(3);
```

XMLHttpRequest 的状态：
0 (未发送)：已创建 XHR 对象，但尚未调用 open() 方法。
1 (已打开)：已调用 open() 方法。
2 (已接回应头)：已调用 send() 方法，且响应的头部信息已可用。
3 (加载中)：正在接收响应。随着数据陆续到达，responseText 属性将被持续更新。
(当服务器以多个数据块形式发送较大响应时，“3 (加载中)” 状态可能被多次更新为该状态)
4 (已完成)：操作已结束，请求已成功完成 (状态码为 2xx) 或发生了错误。

1

What is a
database?

1

什么是数据
库?

What is a database?

- Q: So what do you think it is?
- A: A database is an organized collection of data, stored electronically in a server
- Typically a computer
- or computers distributed
- Or on RAID (redundant array of independent disks to improve robustness or speed etc)

什么是数据库？

- 问：那么，您认为它是什么呢？
- 答：数据库是存储在服务器中、以电子方式组织起来的数据集合
- 通常为一台计算机
- 或多台分布式计算机
- 或存储在 RAID（独立磁盘冗余阵列，用于提升可靠性或速度等）上

Database example

- Lets see one in action (hosted northWind DB)
- https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- How many tables are there in that DB?
- Run a couple of queries such as
- SELECT * FROM [Customers]
- SELECT country FROM [Customers]
- SELECT distinct country FROM [Customers]
- SELECT * FROM Products, OrderDetails WHERE
Products.ProductID=OrderDetails.ProductID AND Quantity>10
- SELECT country, count(CustomerID)
FROM [Customers]
group by country
- SELECT * FROM Customers WHERE Country='Mexico';

数据库示例

- 让我们来看一个实际运行示例（使用 Northwind 数据库）
- https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- 该数据库中包含多少张表？
- 运行若干条查询语句，例如：
- 从 [Customers] 表中选择所有列
- 从 [Customers] 中选择 country
- SELECT DISTINCT country FROM [Customers]
- 从 Products 和 OrderDetails 表中选择所有列，条件是
Products.ProductID=OrderDetails.ProductID 且 Quantity>10
- SELECT 国家, COUNT(CustomerID)
FROM [Customers]
按国家分组
- 从 Customers 表中选择所有字段，其中 Country='Mexico';

Database Management System (DBMS)

- What we just saw was a database, not Database Management System!

数据库管理系统 (DBMS)

- 我们刚才看到的是一个数据库，而不是数据库管理系统！

DBMS
 DataBase Management System

2

DBMS
数据库管理系统

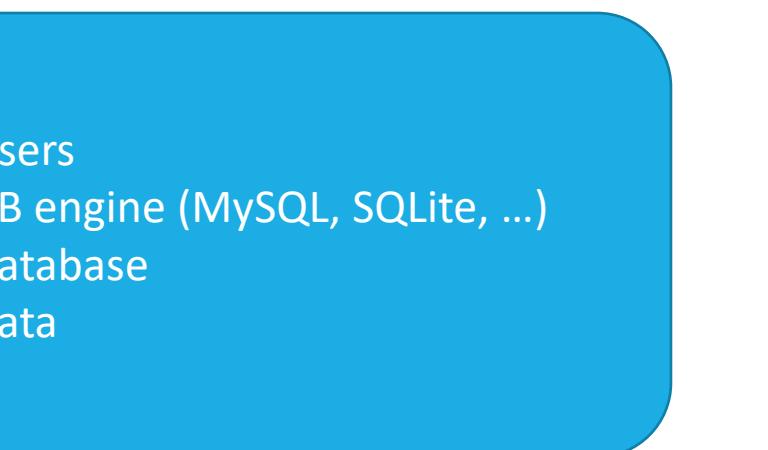
2

What are DBMS?

- The **DBMS** manages incoming, outgoing data, organizes it, and provides ways for the data to be modified or extracted by users or other programs

什么是数据库管理系统
(DBMS) ?

- 数据库管理系统 (DBMS) 负责管理数据的输入与输出, 对数据进行组织, 并为用户或其他程序提供修改或提取数据的方式



- **Massive:**
 - massive data in size! Q: Guess how many terra bytes of data we need to store entire students data?
- **Persistent:**
 - data in the database outlives the programs that execute on that data. Program will finish but the data remains on disk
- **Safe** (data has to stay in consistent state despite hardware failure, **malicious attacks**,
 - run critical applications such as telecommunications and banking systems, have to have guarantees that the data managed by the system
 - will stay in a consistent state, it won't be lost or overwritten when there are failures, and there can be hardware failures.
 - There can be **software failures**.
 - Even simple power outages. You don't want your bank balance to change because the power went out at your bank branch.
 - And of course there are the problem of malicious users that may try to corrupt data.
- **海量:**
 - 数据量极其庞大！问：猜一猜，我们需要多少太字节（TB）的存储空间才能完整保存所有学生的数据？
- **持久化:**
 - 数据库中的数据寿命长于操作该数据的程序。程序会终止运行，但数据仍保留在磁盘上。
- **安全**（即使发生硬件故障或恶意攻击，数据也必须始终保持一致状态）
 - 运行电信和银行系统等关键应用的系统，必须确保其所管理的数据始终处于一致状态，在发生故障时不会丢失或被覆盖，且系统需能应对硬件故障。
 - 可能出现软件故障。
 - 即便是简单的断电情况。您肯定不希望因银行分行停电而导致您的账户余额发生变动。
 - 当然，还存在恶意用户可能试图篡改数据的问题。

DBMSs are also

- **User management**
 - You can create DB users and give them privileges
- **Multi-user**
 - Concurrency control (e.g. at the same time one person can write)
- **Convenient**
 - High level query languages, declarative language (you will see why)
- **Efficient**
 - Performance, No duplication, fast access
- **Reliable**
 - 99.999 uptime! (even missing a single bank transaction could create an issue)

The screenshot shows the MySQL Databases interface with the title 'MySQL® Databases'. Under 'Manage User Privileges', it lists a user and database. A checkbox for 'ALL PRIVILEGES' is checked. Below it, individual privilege checkboxes are listed in pairs across two columns:

| Column 1 | Column 2 |
|-------------------------|----------------|
| ALTER | ALTER ROUTINE |
| CREATE | CREATE ROUTINE |
| CREATE TEMPORARY TABLES | CREATE VIEW |
| DELETE | DROP |
| EVENT | EXECUTE |
| INDEX | INSERT |
| LOCK TABLES | REFERENCES |
| SELECT | SHOW VIEW |

数据库管理系统（DBMS）还具备

- **用户管理**
 - 您可以创建数据库用户并为其分配权限
- **多用户支持**
 - 并发控制（例如，允许多人同时执行写入操作）
- **使用便捷**
 - 高级查询语言，声明式语言（您将明白原因）
- **高效**
 - 性能优异、无数据冗余、访问迅速
- **高可靠性**
 - 99.999% 的正常运行时间！（即使遗漏一笔银行交易也可能引发问题）

The screenshot shows the MySQL Databases interface with the title 'MySQL® Databases'. Under 'Manage User Privileges', it lists a user and database. A checkbox for 'ALL PRIVILEGES' is checked. Below it, individual privilege checkboxes are listed in pairs across two columns:

| Column 1 | Column 2 |
|-------------------------|----------------|
| ALTER | ALTER ROUTINE |
| CREATE | CREATE ROUTINE |
| CREATE TEMPORARY TABLES | CREATE VIEW |
| DELETE | DROP |
| EVENT | EXECUTE |
| INDEX | INSERT |
| LOCK TABLES | REFERENCES |
| SELECT | SHOW VIEW |

Relational DBMS could be categorized as

- Relational DBMs (RDBM) using SQL language
- Non-relational DBMs (also called NoSQL DBMS)
- For our APIs, we only focus on Relational DBMs, in this course
- RDBMS language: SQL
 - SQL: Structured Query Language
- Examples of RDBMS:
 - MySQL (focus on this course for our CRUD on APIs)
 - Sqlite (file base, light weigh and runs in browsers such as
https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
 - Other examples: Oracle, PostgreSQL, SQL server,

关系型数据库管理系统 (RDBMS) 可分为

- 关系型数据库管理系统 (RDBMS) , 使用 SQL 语言
- 非关系型数据库管理系统 (亦称 NoSQL 数据库管理系统)
- 在本课程中, 我们的 API 仅关注关系型数据库管理系统
- RDBMS 使用的语言: SQL
 - SQL: 结构化查询语言
- 关系型数据库管理系统 (RDBMS) 示例:
 - MySQL (本课程重点讲解其在 API 中的 CRUD 操作)
 - SQLite (基于文件、轻量级, 可在浏览器中运行, 例如
https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
 - 其他示例: Oracle、PostgreSQL、SQL Server

Design a DB by example

3

通过示例设计数据库

3

How would you keep records of patients' visits to a clinic?

- Try to keep them in sticky notes? Notebooks or Spreadsheet ?
- For two patients for their clinic visits, create a spreadsheet to keep their records

| Patient name | Age | Clinic Visit |
|--------------|-----|--------------|
| Sarah Brown | 97 | 01/01/2001 |
| John Smith | 56 | 08/08/2018 |

- Now, what if John visited the clinic on another date, 08/01/2019 ? Which one do you keep in the sheet. Or would you add the new one just next to previous one?

| Patient name | Age | Clinic Visit |
|--------------|-----|-----------------------------|
| Sarah Brown | 97 | 01/01/2001 |
| John Smith | 56 | 08/08/2018, 09/09/2019, ... |

Storing multiple values in same column is not a good practice!
Can you guess why?
Aside from reasons you listed, it also violates **1st Normal Form**

- What if there were no more room in that Column to add new visit dates?

您将如何记录患者在诊所的就诊情况？

- 尝试用便签纸记录？还是使用笔记本或电子表格？
- 为两位患者的诊所就诊情况创建一个电子表格以保存其就诊记录

| 患者姓名 | Age | 诊所就诊 |
|--------|-----|-------------|
| 莎拉·布朗 | 97 | 2001年01月01日 |
| 约翰·史密斯 | 56 | 2018年08月08日 |

- 那么，如果约翰于2019年08月01日再次就诊，您应在表格中保留哪一条记录？还是将新记录直接添加在上一条记录旁边？

| 患者姓名 | Age | 诊所就诊日期 |
|--------|-----|----------------------------|
| 莎拉·布朗 | 97 | 2001年1月1日 |
| 约翰·史密斯 | 56 | 2018年8月8日, 2019年9月9日, |

在同一个列中存储多个值并非良好的实践！您能猜出原因吗？除了您所列举的原因之外，这种做法还违反了**1st范式**。

- 如果该列中已无空间添加新的就诊日期，该怎么办？

Exercise 2 cont.

- ... what if there were no more room in that Column to add new visit dates?
- What if we add multiple new columns to record additional visits

| Patient name | Age | Clinic Visit 1 | Clinic Visit 2 | Clinic Visit 3 |
|--------------|-----|----------------|----------------|----------------|
| Sarah Brown | 97 | 01/01/2001 | | |
| John Smith | 56 | 08/08/2018 | 09/09/2019 | 02/02/2020 |

练习2 (续)

- ……如果该列已无剩余空间可添加新的就诊日期，该怎么办？
- 如果我们添加多个新列来记录额外的就诊次数，情况会如何？

| 患者姓名 | Age | 门诊就诊1 | 门诊就诊2 | 门诊就诊3 |
|--------|-----|-----------|-----------|-----------|
| 莎拉·布朗 | 97 | 2001年1月1日 | | |
| 约翰·史密斯 | 56 | 2018年8月8日 | 2019年9月9日 | 2020年2月2日 |

- Sarah is already 97, what if Sarah will never visit the clinic again?

- 莎拉已经97岁了，如果她再也不会去诊所就诊该怎么办？

Exercise 2 cont.

| Patient name | Age | Clinic Visit 1 | Clinic Visit 2 |
|--------------|-----|----------------|----------------|
| Sarah Brown | 97 | 01/01/2001 | |
| John Smith | 56 | 08/08/2018 | 09/09/2019 |

| Patient name | Age | Clinic Visit |
|--------------|-----|--------------|
| Sarah Brown | 97 | 01/01/2001 |
| John Smith | 56 | 08/08/2018 |
| John Smith | 56 | 09/09/2019 |

- Does this work ? Can it be implemented in an spreadsheet?
- Do you notice anything wrong with this approach?
 - Exactly! We have to repeat the name and age of patients everytime!
- Can you suggest a better format for our spreadsheet?

练习 2 (续)

| 患者姓名 | Age | 门诊就诊 1 | 门诊就诊 2 |
|--------|-----|-------------|-------------|
| 莎拉·布朗 | 97 | 2001年01月01日 | |
| 约翰·史密斯 | 56 | 2018年08月08日 | 2019年09月09日 |

| 患者姓名 | Age | 诊所就诊 |
|--------|-----|-------------|
| 莎拉·布朗 | 97 | 2001年01月01日 |
| 约翰·史密斯 | 56 | 2018年08月08日 |
| 约翰·史密斯 | 56 | 2019年09月09日 |

- 这可行吗？能否在电子表格中实现？
- 你是否发现这种方法存在什么问题？
 - 完全正确！我们必须重新输入姓名和年龄 of 患者每次都是如此！
- 你能为我们的电子表格推荐一种更优的格式吗？

Patients visits record-keeping problem with better answer (RDB)

- Or we can simply move to **relational** databases
- Have multiple tables that in some way relate to one another

Unique to each patient

| Patient | | |
|------------|--------------|-----|
| Patient_ID | Patient name | Age |
| 1 | Sarah Brown | 97 |
| 2 | John Smith | 56 |

| Visit | |
|------------|--------------|
| Patient_ID | Clinic Visit |
| 1 | 01/01/2001 |
| 2 | 08/08/2018 |
| 2 | 09/09/2019 |

患者就诊记录管理问题的更优解决方案（关系型数据库，RDB）

- 或者，我们可直接迁移至关系型数据库
- 包含多个在某种程度上相互关联的表

唯一每位患者

| 患者 | | |
|-------|--------|-----|
| 患者ID_ | 患者姓名 | Age |
| 1 | 莎拉·布朗 | 97 |
| 2 | 约翰·史密斯 | 56 |

| 访问 | |
|-------|-------------|
| 患者ID_ | 门诊就诊 |
| 1 | 2001年01月01日 |
| 2 | 2018年08月08日 |
| 2 | 2019年09月09日 |

- Patient table: one row for each patient
- Visit table: one row for each visit. A patient might visit multiple times, thus multiple rows in the visit table would be related to that patient
- Q: We said tables in same relation DB are in some ways related. How these two are related?
- Because we have the column Patient_ID in both tables, we can **relate** them based on that value. This is the **relational** part

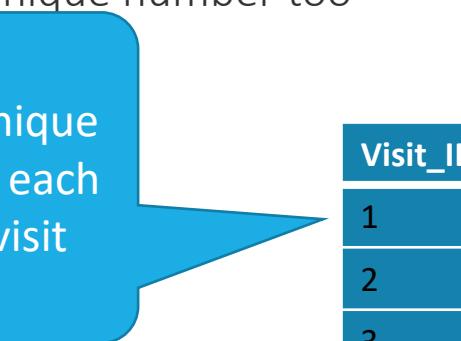
- 患者表：每位患者对应一行
- 访问表：每次访问对应一行。一位患者可能多次就诊，因此访问表中会有多行记录与该患者相关联

- 问：我们之前提到，同一关系型数据库中的表在某种程度上是相互关联的。那么，这两个表具体是如何关联的？
- 因为两个表中都包含 Patient 列，ID 字段同时存在于两张表中，因此我们可以基于该字段将它们关联起来。这是 **关系型** 部分

Patients visits record-keeping Relational DB

- Now lets give each visit in the Visit table a unique number too

| Patient | | |
|------------|--------------|-----|
| Patient_ID | Patient name | Age |
| 1 | Sarah Brown | 97 |
| 2 | John Smith | 56 |



| Visit | | |
|----------|------------|--------------|
| Visit_ID | Patient_ID | Clinic Visit |
| 1 | 1 | 01/01/2001 |
| 2 | 2 | 08/08/2018 |
| 3 | 2 | 09/09/2019 |

- This setup allows me to answer questions like: Who visited the office yesterday
- I can simply match the visit record with the patient names with the help of Patient_ID which exists in both tables.
- Patient_ID in the Patient table is unique and is called **Primary key**
- Q: identify primary key in the Visit table?
- A: Visit_ID
- Patient_ID in the Visit table is called **Foreign key**. A primary key of another table presents in this table
- Q: Did you notice another issue with our database schema? Are you happy with the way we store the Age? Any potential problem you foresee?

患者就诊记录管理关系型数据库

- 现在，我们为“就诊记录”表中的每次就诊也分配一个唯一编号。

| 患者 | | |
|-------|--------|-----|
| 患者ID_ | 患者姓名 | Age |
| 1 | 莎拉·布朗 | 97 |
| 2 | 约翰·史密斯 | 56 |



| 访问 | | |
|-------|-------|-------------|
| 就诊ID_ | 患者ID_ | 门诊就诊 |
| 1 | 1 | 2001年01月01日 |
| 2 | 2 | 2018年08月08日 |
| 3 | 2 | 2019年09月09日 |

- 此设置使我能够回答如下问题：昨天谁访问了办公室？
- 我只需借助同时存在于两张表中的患者_ID，即可轻松将访问记录与患者姓名进行匹配。
- 患者_表中的患者 ID 具有唯一性，称作 **主键**
- 问：请在就诊记录表（Visit 表）中识别主键？
- 答：就诊记录 ID_
- 患者_ID 在就诊记录表（Visit 表）中称作 **外键**。它是另一张表的主键，出现在本表中
- 问：您是否注意到我们数据库模式中存在另一个问题？您对我们是否存储年龄？您预见到哪些潜在问题？

Remarks ... DB Terminologies

Field = Attribute = Column = variable

Table = Entity = Relation = database object

| CustomerID | CustomerName | Address |
|------------|--------------|---------------|
| 2 | Alexa Alino | 123 Burnaby |
| 5 | Bob Brown | 256 Surrey |
| 6 | Clara Cook | 345 Vancouver |
| 13 | David Dail | 467 Richmond |

A cell

Record = Row = tuple = instance

备注……数据库术语

Field = Attribute = Column = variable

表 = 实体 = 关系 = 数据库对象

| CustomerID | CustomerName | 地址 |
|------------|--------------|-------------|
| 2 | Alexa Alino | 123 Burnaby |
| 5 | 鲍勃·布朗 | 萨里256号 |
| 6 | 克拉拉·库克 | 温哥华345号 |
| 13 | 大卫·戴尔 | 列治文467号 |

记录 = 第 = 行
元组 = 实例

- A **table** has rows and columns, where rows represents **records** and columns represent the **attributes**.
- Every **table** is broken up into smaller entities called **fields**. The fields in the Customers table consist of CustomerID, CustomerName and Address

- 答：表由行和列组成，其中行代表记录，而列代表属性。
- 每个数据表均被划分为若干更小的单元，称为字段。客户表（Customers）中的字段包括 CustomerID、CustomerName 和 Address

Remarks ...

- A **field** is a column in a table that is designed to maintain specific information about every record in the table.
- A **record**, also called a **row** or **tuple**, is each individual entry that exists in a table
- Data values are stored in a **Cell**
- "Table Name" is unique (in a database). You cannot have two tables both named "Patient"

备注.....

- 字段 是表中的一列，用于存储该表中每条记录的特定信息。
- 记录 （也称为行 或元组）是指表中存在的每一条独立数据条目。
- 数据值存储在单元格中。
- “表名” 在数据库中必须唯一，您不能同时存在两个名为 “Patient” 的表。

One to Many relationship

- One patient can visit multiple times.
- i.e A single patient (single row) in the Patient table can relate to multiple rows of the Visit table
- That means there is a one to Many (1:M) Relationship from Patient to Visit table

| Patient | | | Visit | | |
|------------|--------------|-----|----------|------------|--------------|
| Patient_ID | Patient name | Age | Visit_ID | Patient_ID | Clinic Visit |
| 1 | Sarah Brown | 97 | 1 | 1 | 01/01/2001 |
| 2 | John Smith | 56 | 2 | 2 | 08/08/2018 |
| | | | 3 | 2 | 09/09/2019 |

一对多关系

- 一个患者可多次就诊。p次。
- 即：患者表（单行记录）与就诊表之间存在一对多关系
- 这意味着患者表与就诊表之间存在一对多(1:M)关系

| 患者 | | | 就诊 | | |
|-------|--------|-----|-------|-------|-------------|
| 患者_ID | 患者姓名 | Age | 就诊编号_ | 患者_ID | 诊所就诊 |
| 1 | 莎拉·布朗 | 97 | 1 | 1 | 2001年01月01日 |
| 2 | 约翰·史密斯 | 56 | 2 | 2 | 2018年08月08日 |
| | | | 3 | 2 | 2019年09月09日 |

- Similarly one to one relationship means a patient could visit the office "only" once!

- 同理，“一对一”关系意味着患者只能到诊所就诊一次！

Creating 1:M
by example

4

创建一对多关系
通过示例

4

Try this at phpMyAdmin

```
CREATE TABLE `patient` (
  `patientid` int(11) NOT NULL,
  `name` varchar(100) DEFAULT NULL,
  PRIMARY KEY(patientid)
);
```

```
CREATE TABLE `visit` (
  `visitid` int(11) NOT NULL,
  `patientid` int(11) NOT NULL,
  `visitDate` date NOT NULL,
  PRIMARY KEY(visitid),
  CONSTRAINT fk_has_patient FOREIGN KEY(patientid)
    REFERENCES patient(patientid)
);
```

```
INSERT INTO patient VALUES
(1, 'Sara Brown'),
(2, 'John Smith'),
(3, 'Jack Ma');
```

```
INSERT INTO visit VALUES
(1, 1, '2002-01-01'),
(2, 2, '2018-08-08'),
(3, 2, '2019-09-09');
```

Q3: Assume we remove row 1 of Patient table; what would each query generate?

Showing rows 0 - 8 (9 total, Query took 0.0005 seconds.)

SELECT * FROM patient, visit

| patientid | name | visitid | patientid | visitDate |
|-----------|------------|---------|-----------|------------|
| 1 | Sara Brown | 1 | 1 | 2002-01-01 |
| 2 | John Smith | 1 | 1 | 2002-01-01 |
| 3 | Jack Ma | 1 | 1 | 2002-01-01 |
| 1 | Sara Brown | 2 | 2 | 2018-08-08 |
| 2 | John Smith | 2 | 2 | 2018-08-08 |
| 3 | Jack Ma | 2 | 2 | 2018-08-08 |
| 1 | Sara Brown | 3 | 2 | 2019-09-09 |
| 2 | John Smith | 3 | 2 | 2019-09-09 |
| 3 | Jack Ma | 3 | 2 | 2019-09-09 |

在 phpMyAdmin 中尝试此操作

创建表 ‘患者’ (

- 患者ID 整数 (11) 非空,
- 可变长度字符串 (100) 默认值为 空值,
- 主键 (患者编号)

);

创建数据表 ‘就诊记录’ (

- 就诊编号 整数 (11) 非空,
- 患者ID 整数(11)非空,
- 就诊日期 日期类型, 非空,
- 主键(就诊ID),
- 约束 fk_关联_患者外键(患者ID)
- 参考文献患者(患者ID)

);

插入到 患者值

```
(1, '莎拉·布朗'),
(2, '约翰·史密斯'),
(3, '马云');
```

插入到 访问记录值

```
(1, 1, "2002-01-01",
(2, 2, "2018-08-08",
(3, 2, "2019-09-09") );
```

问题3：假设我们删除患者表的第1行，各查询将返回什么结果？
每个查询将生成什么结果？

Showing rows 0 - 8 (9 total, Query took 0.0005 seconds.)

SELECT * FROM patient, visit

| patientid | name | visitid | patientid | visitDate |
|-----------|------------|---------|-----------|------------|
| 2 | John Smith | 1 | 1 | 2002-01-01 |
| 3 | Jack Ma | 1 | 1 | 2002-01-01 |
| 1 | Sara Brown | 2 | 2 | 2018-08-08 |
| 2 | John Smith | 2 | 2 | 2018-08-08 |
| 3 | Jack Ma | 2 | 2 | 2018-08-08 |
| 1 | Sara Brown | 3 | 2 | 2019-09-09 |
| 2 | John Smith | 3 | 2 | 2019-09-09 |
| 3 | Jack Ma | 3 | 2 | 2019-09-09 |

Showing rows 0 - 8 (9 total, Query took 0.0005 seconds.)

SELECT * FROM patient, visit

| patientid | name | visitid | patientid | visitDate |
|-----------|------------|---------|-----------|------------|
| 2 | John Smith | 1 | 1 | 2002-01-01 |
| 3 | Jack Ma | 1 | 1 | 2002-01-01 |
| 1 | Sara Brown | 2 | 2 | 2018-08-08 |
| 2 | John Smith | 2 | 2 | 2018-08-08 |
| 3 | Jack Ma | 2 | 2 | 2018-08-08 |
| 1 | Sara Brown | 3 | 2 | 2019-09-09 |
| 2 | John Smith | 3 | 2 | 2019-09-09 |
| 3 | Jack Ma | 3 | 2 | 2019-09-09 |

Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)

SELECT * FROM patient, visit WHERE patient.patientid = visit.patientid

Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)

SELECT * FROM patient, visit WHERE patient.patientid = visit.patientid

| patientid | name | visitid | patientid | visitDate |
|-----------|------------|---------|-----------|------------|
| 1 | Sara Brown | 1 | 1 | 2002-01-01 |
| 2 | John Smith | 2 | 2 | 2018-08-08 |
| 2 | John Smith | 3 | 2 | 2019-09-09 |

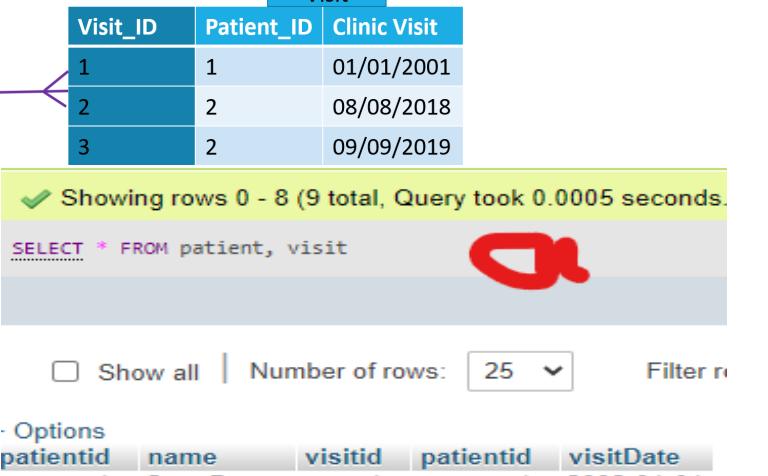
● Q1) What's the difference between
 ● a)
 ● **SELECT * FROM patient, visit**

● b)
 ● **SELECT * FROM patient, visit**
 ● **where patient.patient_ID = visit.patient_ID**

● b would generate =>
 patients and their visits
 Note: Jack Ma is not there, why?
 because
patient.patient_ID = visit.patient_ID
 Does not meet for Jack ma

● Q2: Which one lists patients and each of their visits (if any)?

| Patient | | Visit | | |
|------------|--------------|----------|------------|--------------|
| Patient_ID | Patient name | Visit_ID | Patient_ID | Clinic Visit |
| 1 | Sarah Brown | 1 | 1 | 01/01/2001 |
| 2 | John Smith | 2 | 2 | 08/08/2018 |
| 3 | Jack Ma | 3 | 2 | 09/09/2019 |

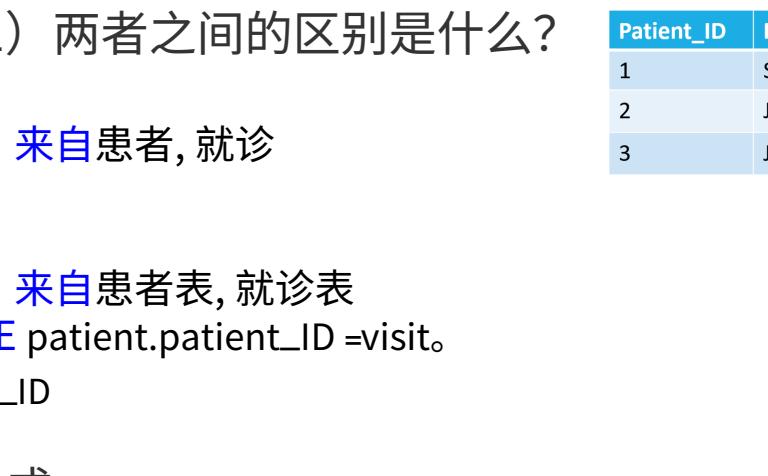


```

    ✓ Showing rows 0 - 8 (9 total, Query took 0.0005 seconds.)
    SELECT * FROM patient, visit
  
```

Filter rows: Search

| Patient | | Visit | | |
|------------|--------------|----------|------------|--------------|
| Patient_ID | Patient name | Visit_ID | Patient_ID | Clinic Visit |
| 1 | Sarah Brown | 1 | 1 | 01/01/2001 |
| 2 | John Smith | 2 | 2 | 08/08/2018 |
| 3 | Jack Ma | 3 | 2 | 09/09/2019 |



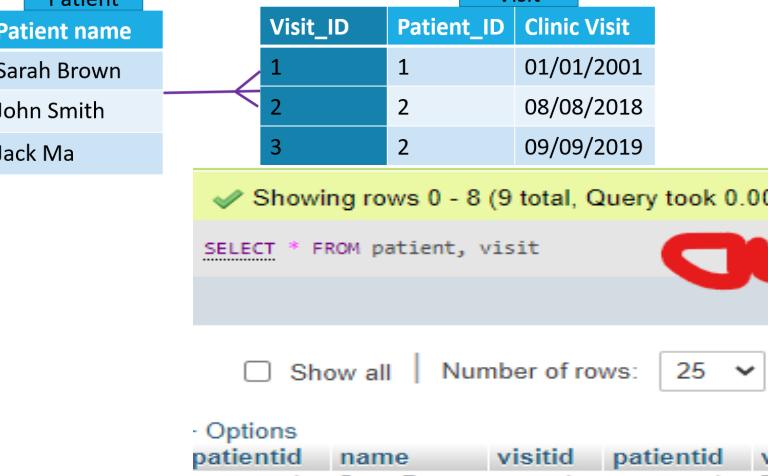
```

    ✓ Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)
    SELECT * FROM patient, visit WHERE patient.patientID = visit.patientID
  
```

Filter rows: Search

- 问题1) 两者之间的区别是什么?
- a)
- 选择 * 来自患者, 就诊**
- b)
- 选择 * 来自患者表, 就诊表**
- WHERE patient.patient_ID = visit.patient_ID**
- b将生成=>
患者及其就诊记录
注意: 杰克·马 (Jack Ma) 不在其中, 为什么?
- 因为
患者.患者_ID = 就诊.患者_ID
不满足杰克·马 (Jack Ma) 的条件
- 问题2: 以下哪一项列出了所有患者及其各自的就诊记录 (如果任意?)

| Patient | | Visit | | |
|------------|--------------|----------|------------|--------------|
| Patient_ID | Patient name | Visit_ID | Patient_ID | Clinic Visit |
| 1 | Sara Brown | 1 | 1 | 2002-01-01 |
| 2 | John Smith | 1 | 1 | 2002-01-01 |
| 3 | Jack Ma | 1 | 1 | 2002-01-01 |
| 1 | Sara Brown | 2 | 2 | 2018-08-08 |
| 2 | John Smith | 2 | 2 | 2018-08-08 |
| 3 | Jack Ma | 2 | 2 | 2018-08-08 |
| 1 | Sara Brown | 3 | 2 | 2019-09-09 |
| 2 | John Smith | 3 | 2 | 2019-09-09 |
| 3 | Jack Ma | 3 | 2 | 2019-09-09 |



```

    ✓ Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)
    SELECT * FROM patient, visit WHERE patient.patientID = visit.patientID
  
```

Filter rows: Search

| Patient | | Visit | | |
|------------|--------------|----------|------------|--------------|
| Patient_ID | Patient name | Visit_ID | Patient_ID | Clinic Visit |
| 1 | Sara Brown | 1 | 1 | 2002-01-01 |
| 2 | John Smith | 2 | 2 | 2018-08-08 |
| 2 | John Smith | 3 | 2 | 2019-09-09 |

Schema vs data

- Schema versus data
 - Schema: data model, data types of columns,
Note: all data entered in rows of the same column must all have same data type
 - data: values stored in columns
 -
- ```
CREATE TABLE `patient` (
 `patientid` int(11) NOT NULL,
 `name` varchar(100) DEFAULT NULL,
 PRIMARY KEY(patientid)
);
...
INSERT INTO patient VALUES
(1, 'Sara Brown'),
(2, 'John Smith'),
(3, 'Jack Ma');
```

## 模式与数据

- 模式与数据的对比
    - 模式: 数据模型、各列的数据类型，  
注意: 同一列各行中输入的所有数据，其数据类型必须完全相同。
    - 数据: 各列中存储的具体值
- ```
创建表 `p` 患者 (
  遵者ID 整数(11)非空,
  可变长度字符串(100)默认值为空,
  主键(患者ID)
);
...
插入数据到 患者表中
(1, “莎拉·布朗”),
(2, ‘约翰·史密斯’),
(3, ‘马云’);
```

Popular Datatypes (MySQL)

| | |
|------------------|--|
| VARCHAR(size) | A VARIABLE length string (can contain letters, numbers, and special characters). The <i>size</i> parameter specifies the maximum column length in characters - can be from 0 to 65535 |
| LONGTEXT | Holds a string with a maximum length of 4,294,967,295 characters |
| INT(size) | Integer number. The <i>size</i> parameter specifies the maximum display width (which is 255) |
| DECIMAL(size, d) | An exact fixed-point number. The total number of digits is specified in <i>size</i> . The number of digits after the decimal point is specified in the <i>d</i> parameter. The maximum number for <i>size</i> is 65. |
| DATE | A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' |

常用数据类型 (MySQL)

| | |
|-------------------|--|
| VARCHAR(长度) | 一种可变长度的字符串（可包含字母、数字和特殊字符）。参数 长度 指定列的最大字符数，取值范围为 0 至 65535。 |
| LONGTEXT | 用于存储最大长度为 4,294,967,295 个字符的字符串。 |
| INT (size) | 整数。参数 size 指定最大显示宽度（为 255）。 |
| DECIMAL(精度, 小数位数) | 精确的定点数。总位数由 size 指定。小数点后的位数由参数 d 指定。size 的最大值为 65. |
| DATE | 日期类型，格式为 YYYY-MM-DD。支持的取值范围为 '1000-01-01' 至 '9999-12-31' |

- There are many more data types. Please refer to the source:
https://www.w3schools.com/sql/sql_datatypes.asp

- 还有更多数据类型，请参阅以下资料：
https://www.w3schools.com/sql/sql_datatypes.asp

Referential integrity
constraint

5

参照完整性
约束

5

Referential integrity constraint

- Q: in our patient visit data model, which table has to be created before the other one?
- (Which table can exist by itself? Patient or visit?)
- A: patient table has to exist before the visit, since the primary key of the patient table is used in the visit table

Q: What if we delete the second row of patient table?

```
DELETE FROM patient  
WHERE patient_ID=2;
```

A: the 2nd and 3rd rows of Visit table would not make sense!

Lets try it at phpMyAdmin!

- Referential integrity means for every value of a foreign key there is a primary key with that value

- A primary key must exist before the foreign key can be defined
- Must create the patient table before the visit table
- E.g. For every value of patientID in the visit table there is a value of patientID in the patient table

| Patient | | Visit | | |
|------------|--------------|----------|------------|--------------|
| Patient_ID | Patient name | Visit_ID | Patient_ID | Clinic Visit |
| 1 | Sarah Brown | 1 | 1 | 01/01/2001 |
| 2 | John Smith | 2 | 2 | 08/08/2018 |
| 3 | Jack Ma | 3 | 2 | 09/09/2019 |

参照完整性约束

- 问：在我们的患者就诊数据模型中，哪个表必须先于另一个表创建？
- (哪个表可以独立存在？是患者表还是就诊表？)
- 答：患者表必须先于就诊表存在，因为就诊表会用到患者表的主键。

问：如果我们删除患者表中的第二行数据，会发生什么？

从患者表中删除其中患者_ID=2;

A: Visit 表中的 2nd 行和 3rd 行将毫无意义！

让我们在 phpMyAdmin 中尝试一下！

| Patient | | Visit | | |
|------------|--------------|----------|------------|--------------|
| Patient_ID | Patient name | Visit_ID | Patient_ID | Clinic Visit |
| 1 | Sarah Brown | 1 | 1 | 01/01/2001 |
| 2 | John Smith | 2 | 2 | 08/08/2018 |
| 3 | Jack Ma | 3 | 2 | 09/09/2019 |

- 参照完整性是指对于每一个外键值均存在一个具有该值的主键。

- 必须先定义主键，才能定义外键。
- 例如，必须先创建患者表，再创建就诊表。
- 例如：就诊表中的每个 patientID 值，在患者表中均存在对应的 patientID 值。

Learn SQL
by Examples!

6

学习SQL
通过示例！

6

SQL Statements(queries)

- open:https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- **SELECT * FROM Customers;**
- This statement selects all the records(all rows, all tuples) in the customers table
SQL keywords(such as **SELECT**) are NOT case sensitive: **selEct** is the same as **SELECT**.
; is not needed by good practice
- **SELECT * FROM Customers
WHERE Country='Mexico';**
- Example 3:
**SELECT * FROM Customers
WHERE CustomerID=1;**
- Example:
SELECT CustomerName FROM Customers;
- Example:
SELECT CustomerName FrOM Customers WHERE City="London"
- Example:
SELECT Country FROM Customers;
- Example:
SELECT DISTINCT Country FROM Customers; (keep unique rows)

SQL Statements(queries)

- open:https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- **选择所有列来自客户表；**
- 该语句从 **customers** 表中选取所有记录（即所有行、所有元组）。
SQL 关键字（例如 **SELECT**）不区分大小写：NOT **selEct** 与 **SELECT** 含义相同。
根据良好实践，分号（;）并非必需。
- **选择*来自客户
WHERE Country='墨西哥';**
- **示例 3：**
**选择所有列来自客户表
WHERE 客户ID=1;**
- **示例：**
SELECT CustomerName FROM Customers;
- **示例：**
SELECT CustomerName FROM Customers WHERE City="London"
- **示例：**
SELECT国家FROM Customers;
- **示例：**
SELECT DISTINCT国家FROM Customers; (仅保留唯一行)

WHERE clause operators

BETWEEN
LIKE
IN

WHERE 子句运算符

BETWEEN、
LIKE、IN

WHERE Clause: Operators in The WHERE Clause

- Q: what does this one do?
- `SELECT * FROM OrderDetails WHERE Quantity > 80`
- Q: list the customers who live in the city of Berlin in Germany
- `SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin';`

| Operator | Description |
|----------|--|
| = | Equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |
| <> | Not equal. Note: In some versions of SQL this operator may be written as != |
| BETWEEN | Between a certain range |
| LIKE | Search for a pattern |
| IN | To specify multiple possible values for a column |

WHERE 子句: WHERE 子句中的运算符

- 问: 这个语句的作用是什么?
- `选择*`
- `来自订单详情`
- `WHERE Quantity > 80`
- 问: 列出居住在以下地区的客户:
德国柏林市
- `SELECT * 来自 客户
WHERE Country='德国'
AND City='柏林';`

| 操作员 | 说明 |
|---------|----------------------------------|
| = | 等于 |
| > | 大于 |
| < | 小于 |
| >= | 大于或等于 |
| <= | 小于或等于 |
| <> | 不等于。注意: 在某些 SQL 版本中, 该运算符可能写作 != |
| BETWEEN | 介于某一范围内 |
| LIKE | 搜索匹配的模式 |
| IN | 用于为列指定多个可能的值 |

WHERE Clause: AND, OR and NOT Operators

- Q: What products were ordered for quantity of more than 80?

```
SELECT * FROM Products,OrderDetails  
WHERE Products.ProductID = OrderDetails.ProductID  
AND  
OrderDetails.Quantity>80
```

Equal signs makes sure we talk about same product in both tables. (remember how PatientID relates two tables in week 1)

WHERE 子句: AND、OR 和 NOT 运算符

- Q: What products were ordered for quantity of more than 80?

```
SELECT * FROM Products,OrderDetails  
在 Products 表中。产品 ID = OrderDetails 表中。产品 ID  
AND  
OrderDetails.Quantity>80
```

等号可确保我们讨论的是两个表中均包含同一产品
(请注意 PatientID 的关联方式
第一周中的两张表)

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character
- Examples:
 - SELECT * FROM Customers WHERE CustomerName LIKE 'a%'
 - Finds any CustomerName that start with "a"
 - SELECT * FROM Customers WHERE CustomerName LIKE '%st%'
 - Finds any values that has "st" in any position
 - Q: selects all customers with a CustomerName ending with "a"

- LIKE 运算符用于 WHERE 子句中，在某一列中搜索指定的模式。
- 在 LIKE 运算符中，常与之配合使用的通配符有两个：
- % – 表示百分号 (%) 表示零个、一个或多个字符
- _ – 下划线表示单个字符
- 示例：
 - 从 Customers 表中选择所有列，其中 CustomerName 以 'a' 开头
 - 查找以 “a” 开头的任意 CustomerName
 - 从 Customers 表中选择所有列，其中 CustomerName 包含 'st'。
 - 查找在任意位置包含 “st” 的所有值。
 - Q: selects all customers with a CustomerName ending with "a"

SQL Wildcard Characters

31

- A wildcard character is used to **substitute one or more characters in a string.**
- Wildcard characters are used with the SQL LIKE operator.
- The LIKE operator itself is used in a WHERE clause to search for a specified pattern in a column.

| Symbol | Description | Example |
|--------|------------------------------------|-------------------------------------|
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| _ | Represents a single character | h_t finds hot, hat, and hit |

SQL 通配符字符

31

- 通配符用于**替代字符串中的一个或多个字符。**
- 通配符与 SQL LIKE 运算符配合使用。
- LIKE 运算符本身用于 WHERE 子句中，在某一列中搜索指定的模式。

| Symbol | Description | Example |
|--------|------------------------------------|-------------------------------------|
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| _ | Represents a single character | h_t finds hot, hat, and hit |

SQL IN (NOT IN) Operator

SQL IN (NOT IN) 运算符

- allows you to specify multiple values in a WHERE clause.
- is a shorthand for multiple OR conditions.
- SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');

• 允许您在 WHERE 子句中指定多个值。

• 是多个 OR 条件的简写形式。

• SELECT * FROM Customers WHERE Country IN ('德国', '法国', '英国');

Q: what does this query return?
SELECT * FROM customers
where country
in
(select country from customers
where country like '__')

问：该查询返回什么结果？
SELECT*
FROM customers WHERE country
IN
(SELECT country FROM customers
位于国家如 '__')

SQL BETWEEN Operator and sorting alphabetically (ASCII code)

39

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

- Example:**

```
SELECT * FROM Products  
WHERE Price BETWEEN 10 AND 20;
```

- Example**

```
SELECT * FROM customers  
where country BETWEEN 'UA' AND 'UZ'
```

'AAA'
'AB'
'B'
..
'ZA'
'Z...Z'
'a'
'b'

Note: if we sort the strings above alphabetically in ascending order , the uppercase ones appear before the lower case ones.

The reason is their ASCII codes.
E.g. ASCII code of 'A' is 65
But ASCII code of 'a' is 97

ASCII stands for American Standard Code for Information Interchange.

SQL BETWEEN Operator and sorting alphabetically (ASCII code)

39

- BETWEEN 运算符用于选取指定范围内的值。这些值可以是数字、文本或日期。
- BETWEEN 运算符是包含边界的：起始值和结束值均被包含在内。

- 示例:**

选择所有列来自产品表
其中 价格 介于 10 与20之间；

- 示例**

SELECT * FROM 客户
其中 国家 介于 'UA' AND 'UZ'

'AAA'
'AB'
'B'
..
'ZA'
'Z...Z'
'a'
'b'

注意：如果将上述字符串按字母顺序升序排列，则大写字母会排在小写字母之前。

原因在于它们的 ASCII 码值。
例如， 'A' 的 ASCII 码为 65。
而 'a' 的 ASCII 码为 97。

ASCII 是美国标准信息交换码的缩写。

Order by [column name]

按 [列名]排序

- Q: What does this query return?
- SELECT * FROM [Customers] ORDER BY City
-
- Q: What products were ordered for quantity of more than 50? (sort the result in descending order so that the highest ordered product appears at top of result)
- A: SELECT * FROM Products,OrderDetails where Products.ProductID = OrderDetails.ProductID AND OrderDetails.Quantity>50 ORDER BY OrderDetails.Quantity DESC

- Q: 该查询返回什么结果?
 - 从 [Customers] 表中选择所有列，并按 City 字段排序
 -
- Q: 哪些产品的订购数量超过 50? (按降序排列结果，使订购数量最高的产品排在结果最上方)
 - A: SELECT * FROM Products ,OrderDetails 表，其中 Products.ProductID = OrderDetails.ProductID 且 OrderDetails.Quantity >50 ，按 OrderDetails.Quantity 降序排列

Arithmetic within SQL clauses

- We can use arithmetic within SQL clauses.
- Example:
SELECT Quantity, Quantity*10 FROM OrderDetails;
- Example:
SELECT *, Quantity*Price FROM Products, OrderDetails WHERE Products.ProductID = OrderDetails.ProductID;
- Try this one too:
SELECT *, Quantity*Price AS Total FROM Products, OrderDetails WHERE Products.ProductID = OrderDetails.ProductID;
- In AS Total, Total is an alias that can be used to name the new column our set result generate

SQL 子句中的算术运算

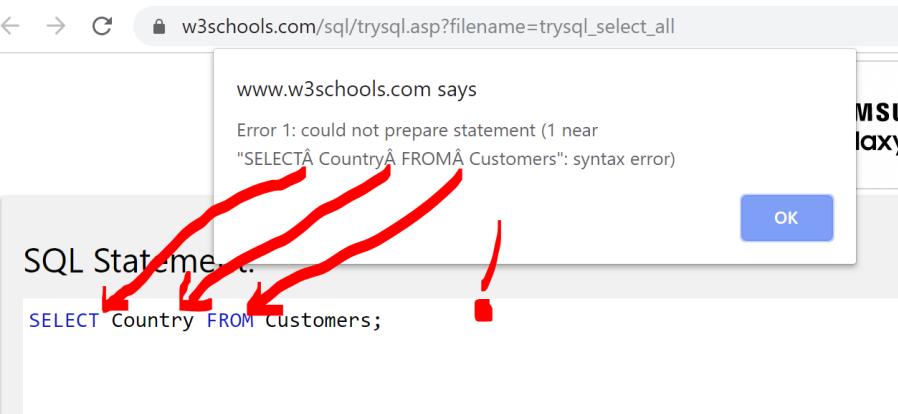
- 我们可以在 SQL 子句中使用算术运算。
- 示例：
选择数量，数量*10 来自 OrderDetails；
- 示例：
选择所有列，数量×单价 来自 Products 表和 OrderDetails 表，其中 Products.ProductID = 等于 OrderDetails.ProductID；
- 也请尝试这个示例：
选择所有列，数量 × 单价 作为总计 来自 Products 表和 OrderDetails 表，其中 Products.ProductID = 等于 OrderDetails.ProductID；
- 在 AS Total 中，Total 是一个别名，可用于为查询结果集中生成的新列命名。

Tips

Tips

Tip: beware of whitespaces

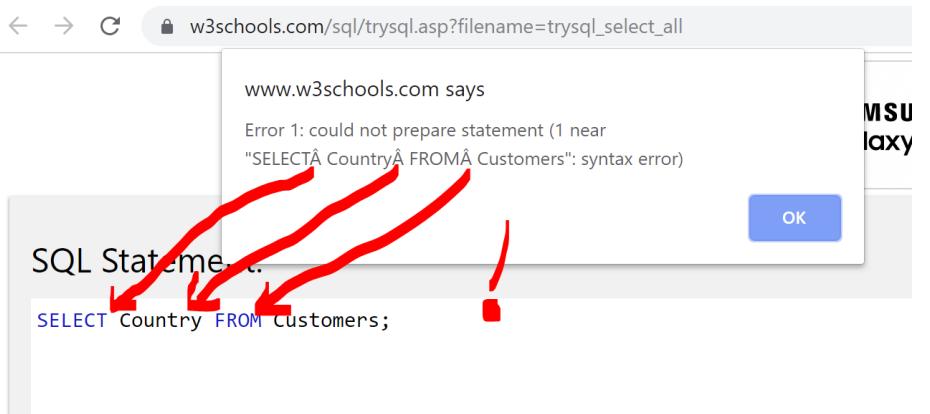
- Try this statement (copy it as is and paste it at https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in)
- and run it
- SELECT Country FROM Customers;**
- What seems to be wrong here?



- The problem is that the above text contains invisible white spaces which are not valid blank spaces. You need to replace them with blank space
- You can use this link to remove white spaces: <https://www.textfixer.com/tools/remove-white-spaces.php>

提示：注意空格

- 请尝试执行以下语句（原样复制并粘贴到 https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in）并运行该语句
- 选择 国家 来自 客户；**
- 此处似乎存在什么问题？



- 问题在于上述文本中包含不可见的空白字符，而这些字符并非有效的空格。空格。您需要用空格替换它们。
- 您可以使用以下链接来删除空白字符：
<https://www.textfixer.com/tools/remove-white-spaces.php>

Tip: confusing column names!

提示：列名容易混淆！

- Q1: Try to run this query. What is the problem? How can you fix it?

```
SELECT ProductID FROM Products,OrderDetails  
WHERE Products.ProductID = OrderDetails.ProductID  
AND  
OrderDetails.Quantity>80
```

- Q2: Now remove the WHERE clause and again run the query. What do you get? Why? How can you fix it?
- Q3: what will this query return?
- SELECT * FROM Shippers, employees;

- 问题1：尝试运行此查询。存在什么问题？如何修复？
SELECT ProductID FROM Products, OrderDetails WHERE Products.ProductID = OrderDetails.ProductID
AND
订单详情.数量>80

- 问题2：现在移除 WHERE 子句，再次运行该查询。你得到了什么结果？为什么会出现这种情况？如何修复？
 - Q3: what will this query return?
 - SELECT * FROM Shippers, employees;

SQL functions

AVG, SUM, MIN, COUNT, ...

SQL 函数

AVG、SUM、MIN、COUNT……

SQL MIN(), MAX(), SUM, COUNT() Functions

- Q: How much is the most expensive product ?
A: `SELECT MAX(price) FROM products`
- `SUM()`: returns the total sum of a numeric column
- **Q:** what is the total quantity of all products ordered combined?
- **A:** `SELECT SUM(Quantity) FROM OrderDetails;`
- Q: What is the total quantity of all goods shipped in all orders (use OrderDetails table)
- A: `SELECT SUM(Quantity) FROM [OrderDetails]`
- `COUNT()`, counts the number of rows in a table
- Try `SELECT country FROM [Customers]`
- Observe `SELECT count(country) FROM [Customers]`
- **Q:** How many different countries are there in the table Customers ?
- A: use `count` on `SELECT distinct country FROM [Customers]`
- `SELECT count(distinct country) FROM [Customers]`

SQL MIN()、MAX()、SUM 和 COUNT() 函数

- Q: 最贵的产品价格是多少?
A: `SELECT MAX(price) FROM products`
- `SUM()`: 返回数值列的总和
- **Q:** 所有已订购产品的总数量是多少?
- **A:** `SELECT SUM(Quantity) FROM OrderDetails;`
- Q: 所有订单中发货商品的总数量是多少? (使用 OrderDetails 表)
- A: `SELECT SUM(Quantity) FROM [OrderDetails]`
- `COUNT()` 函数用于统计表中的行数。
- 尝试执行 `SELECT country FROM [Customers]`
- 观察执行 `SELECT COUNT(country) FROM [Customers]` 的结果
- **Q:** Customers 表中包含多少个不同的国家?
- A: 在 `SELECT DISTINCT country FROM [Customers]` 中使用 `COUNT`
- `SELECT COUNT(DISTINCT country) FROM [Customers]`

Grouping on the result set ...

对结果集进行分组……

GROUP BY Statement

- The GROUP BY statement **groups rows** that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

- Example:

```
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;
```

We count number of customerIDs that are in the group of same country

- Q: What does the above query return?
- Q: What if you remove " GROUP BY Country "?

GROUP BY 语句

- GROUP BY 语句将具有相同值的行分组为汇总行，例如“统计每个国家/地区的客户数量”。
- GROUP BY 语句通常与聚合函数（COUNT、MAX、MIN、SUM、AVG）配合使用，以按一个或多个列对结果集进行分组。

- 示例：SELECT

```
COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;
```

我们统计属于同一国家的客户ID数量。

- Q: What does the above query return?
- Q: What if you remove " GROUP BY Country "?

GROUP BY examples 1

- Q: Write a query to return number of orders by each city
- A: `SELECT City,COUNT(orderID) FROM [Orders], customers where Orders.CustomerID=customers.CustomerID GROUP BY city`
- Q: How can we verify the answer?

GROUP BY 示例 1

- Q: 编写一个查询语句，按城市返回订单数量。
- A: `SELECT 城市, COUNT(订单ID) FROM [订单], 客户 WHERE 订单.客户ID=客户.客户ID GROUP BY 城市`
- Q: 我们如何验证该答案？

GROUP BY examples 2

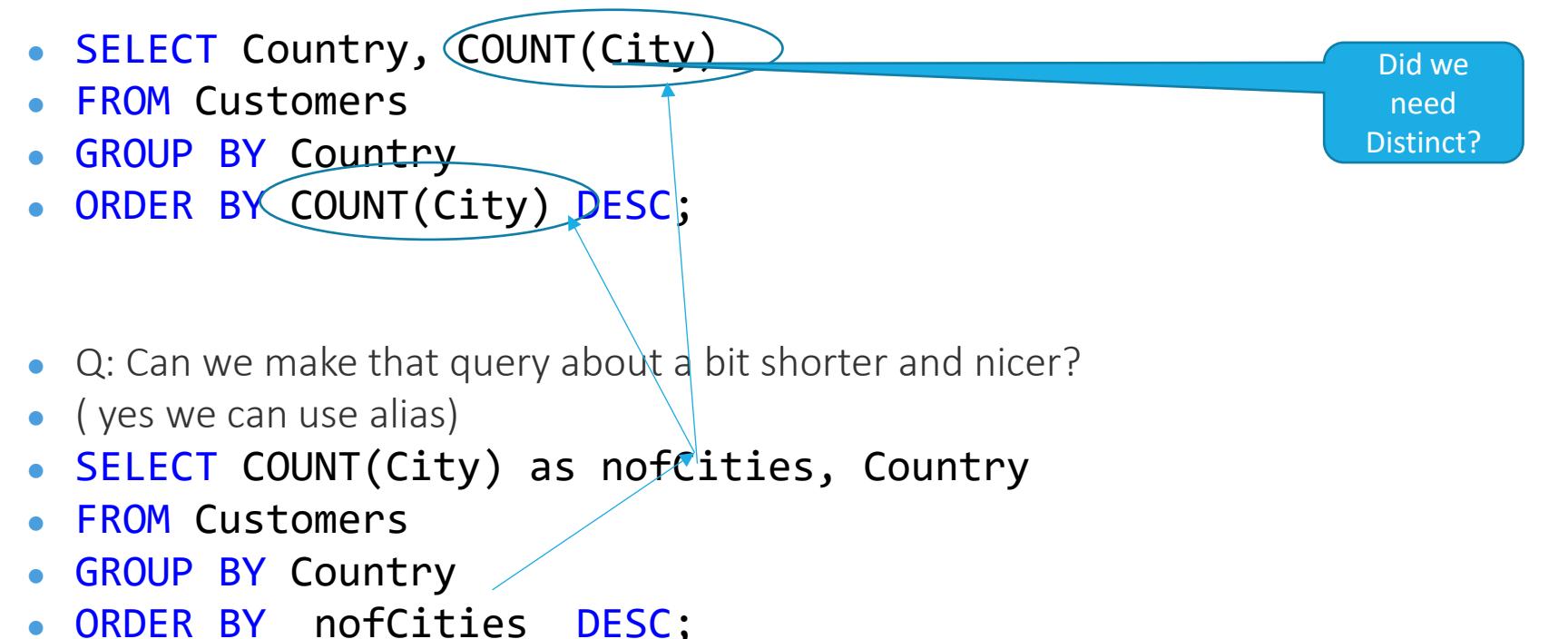
- Q: Write a query statement to list the number of customers in each country, sorted high to low
- SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
- Q: Write the same query about for only the countries whose name starts with A:
 - SELECT Country,COUNT(CustomerID)
 - FROM Customers
 - GROUP BY Country
 - where country like "A%" Wait! "Where" does not work here!

GROUP BY 示例 2

- Q: 编写一条查询语句，按从高到低的顺序列出每个国家的客户数量。
- SELECT COUNT(CustomerID), 国家
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
- Q: Write the same query about for only the countries whose name starts with A:
 - SELECT 来自Country ,COUNT(CustomerID)
Customers 表
 - 按国家分组
 - where country like "A%" Wait! "Where" does not work here!

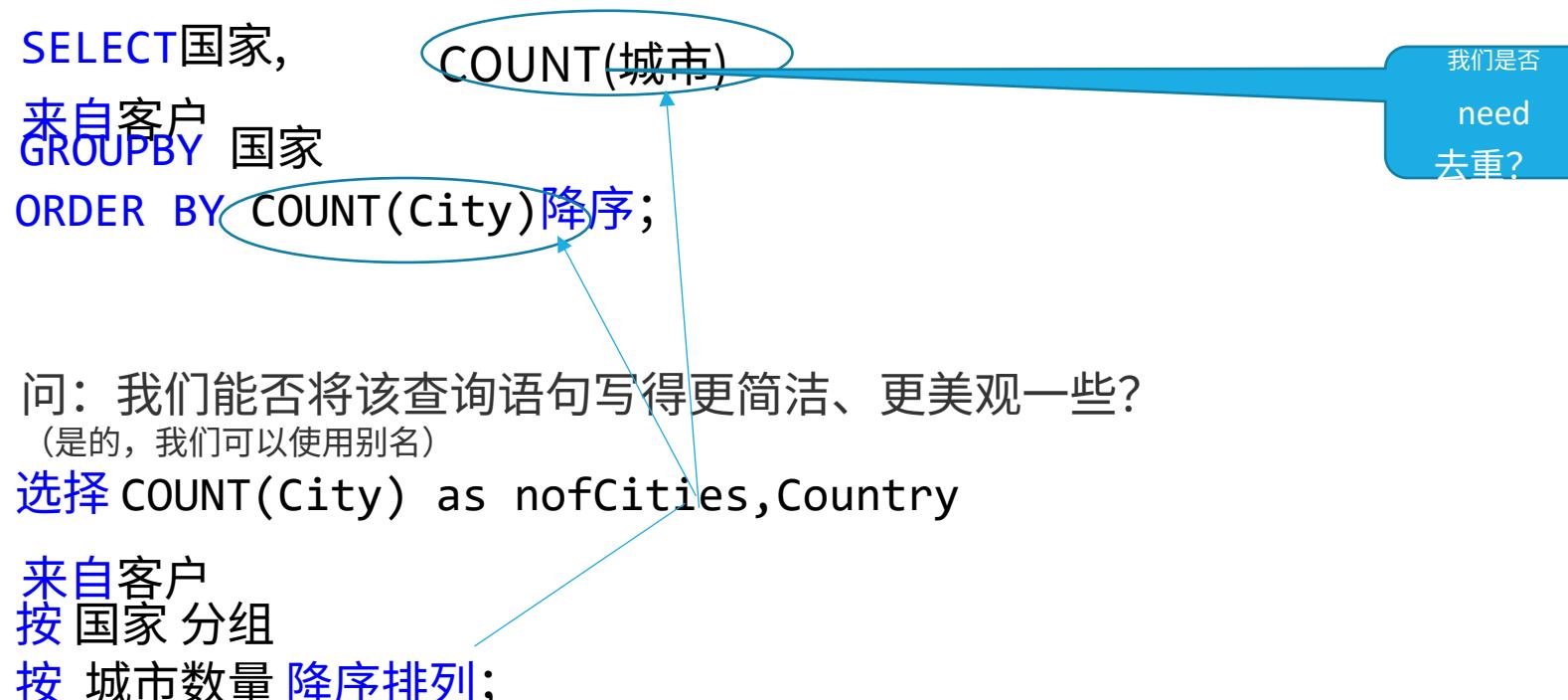
GROUP BY examples 3

- Q: Write a query statement to count the cities of each country in the northWind DB at
https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all



GROUP BY 示例 3

- Q: 编写一条查询语句，统计 NorthWind 数据库中各国的城市数量，数据库地址为
https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all



The **Having** clause

Made for use with aggregate functions and after "Group by"

“HAVING” 子句用于与聚合函数配合使用，且位于
“GROUP BY” 之后

WHERE does not work everyWHERE!

- Why this query does not run (throws errors!)
- `SELECT Country,COUNT(CustomerID)`
- `FROM Customers`
- `GROUP BY Country`
- `where country like "A%"`
- Wait! "Where' does not work here!!!"

Where
does

Not work with aggregate
functions or aggregate
parameters!

WHERE 子句并非在所有位置都适用!

- Why this query does not run (throws errors!)
- `SELECT 国家, COUNT(客户ID)`
- 来自客户
- 按 国家 分组
- `WHERE country LIKE "A%"`
- 等等! “WHERE” 在此处不适用！！！

WHERE
子句位于

不支持聚合函数或聚合
参数!

The SQL HAVING Clause

```
Select A1,A2,A3  
From T1,T2, T3  
Where condition  
Group bY columns  
Having conditions
```

Aggregation functions
over values in multiple
rows: min, max, sum, avg,
count

SQL HAVING 子句

选择 A1、A2、
A3来自 T1、T2、
T3其中 条件按
列分组 且满足

聚合函数：对多行中的值进
行计算，包括最小值 (min) 、
最大值 (max) 、求和 (sum) 、
平均值 (avg) 和计数 (count)

Note: The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one "or more columns".

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

So HAVING is like "WHERE" when we use GROUP BY(aggregated functions)!

注意：GROUP BY 语句通常与聚合函数 (COUNT、MAX、MIN、SUM、AVG) 配合使用，以按一个或多个列对结果集进行分组。

SQL 中引入 HAVING 子句的原因在于，WHERE 关键字无法与聚合函数一起使用。

因此，HAVING 子句在使用 GROUP BY (聚合函数) 时，作用类似于 "WHERE" 子句！

The SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE clause could not be used with aggregate functions or after group by was performed.
- Example
- Write a query that lists the number of customers in each country.
- ```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
```
- Now Only include countries with more than 10 customers:  
(add this to last line)
- HAVING COUNT(CustomerID) > 10;**
- ```
SELECT Country,COUNT(CustomerID)
FROM Customers
where count(CustomerID )
GROUP BY Country
```
- ```
SELECT Country, COUNT(CustomerID)
FROM Customers
WHERE country LIKE "A%"
GROUP BY Country
Its ok! "Where" is used before "Group by"
and is not used on aggregate functions
```

The screenshot shows a SQL editor interface. At the top, a red box highlights an error message: "www.w3schools.com says Error 1: could not prepare statement (1 misuse of aggregate: count())". Below the error is an "OK" button. The main area is titled "SQL Statement:" and contains the following code:  

```
SELECT Country,COUNT(CustomerID)
FROM Customers
where count(CustomerID)
GROUP BY Country
```

A red arrow points to the word "count" in the "where" clause. Below the code, it says "Edit the SQL Statement, and click "Run SQL" to see the result." At the bottom is a green "Run SQL »" button.

## SQL HAVING 子句

- HAVING 子句被添加到 SQL 中，是因为 WHERE 子句无法与聚合函数配合使用，也无法在执行 GROUP BY 后使用。
- 示例
- 编写一个查询，列出每个国家的客户数量。
- 选择计数(客户ID)，国家  
来自 客户  
按 国家 分组**
- 现在仅包含客户数量超过 10 个的国家：  
(将此内容添加到最后一行)
- HAVING COUNT(客户ID) > 10;**
- ```
SELECT Country, COUNT(CustomerID)
FROM Customers
where count(CustomerID )
GROUP BY Country
```
- ```
SELECT Country, COUNT(CustomerID)
FROM Customers
WHERE country LIKE "A%"
GROUP BY Country
按国家分组
没问题! "Where" 应置于 "Grou" 之前
且不适用于聚合函数
```

The screenshot shows a SQL editor interface, identical to the one on the left. It displays the same error message: "www.w3schools.com says Error 1: could not prepare statement (1 misuse of aggregate: count())". The "OK" button is visible. The "SQL Statement:" section contains the same erroneous code:  

```
SELECT Country,COUNT(CustomerID)
FROM Customers
where count(CustomerID)
GROUP BY Country
```

A red arrow points to the word "count" in the "where" clause. Below the code, it says "Edit the SQL Statement, and click "Run SQL" to see the result." At the bottom is a green "Run SQL »" button.

## "WHERE" does not work everyWHERE! (part 2)

```
SELECT Country,COUNT(CustomerID)
FROM Customers
GROUP BY Country
where country like "A%"
HAVING
```

Or place  
Where before  
Group by

```
SELECT Country,COUNT(CustomerID)
FROM Customers
where country like "A%"
GROUP BY Country
```

Q:List countries with more than 2 customers

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
GROUP BY Country
where c>2
HAVING
```

Note!

- 1) The "where" has to go before the "having" and the group by
- 2) "where" cannot be used on aggregate functions

## "WHERE" 并非在所有地方都适用! (第二部分)

```
SELECT 国家,COUNT(客户ID)
来自 客户
GROUPBY 国家
WHERE 国家LIKE "A%"
HAVING
```

或在  
GROUP BY 之前  
放置 WHERE

```
选择 国家,客户ID数量来自 客户表
其中 国家类似 "A%"
按以下分组: 国家
```

问题: 列出客户数量超过 2 个的国家

```
SELECT 国家,COUNT(客户ID) AS c
来源 客户表按
国家筛选条件 c>2
HAVING
```

注意! 1) The "where" 子句  
必须位于 "having" 子句和 "  
GROUP BY" 子句之前; 2)  
"where" 不能对聚合函数使用 "  
where" 子句

## "WHERE" does not work everyWHERE!

```
SELECT Country, COUNT(CustomerID)
FROM Customers
GROUP BY Country
where CustomerName like "A%"
Not ok! Where cannot be used after we grouped rows
```

## “WHERE” 并非在所有位置都适用！

```
SELECT 国家, COUNT(客户ID)
来自 客户
按国家 分组
WHERE CustomerName LIKE "A%"
不正确！WHERE 子句不能用在对行进行分组之后。
```

```
SELECT Country, COUNT(CustomerID)
FROM Customers
where CustomerName like "A%"
GROUP BY Country
Is ok! Where is used before we group rows
```

选择 国家, 客户ID数量  
来自 客户  
其中 客户姓名类似于 "A%"  
按国家/地区分组 国家/地区  
没问题！WHERE 子句在对行进行分组之前使用

## "WHERE" does not work everyWHERE!

```
SELECT Country, COUNT(CustomerID)
FROM Customers
where COUNT(CustomerID) > 2
GROUP BY Country
```

Not ok! Where cannot be used on aggerate functions

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
where c > 2
GROUP BY Country
```

Not ok! Where cannot be used on aggerate functions (place having c>2 after "group by")

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
having c > 2
GROUP BY Country
```

Not ok! Having cannot be used before we have grouped the rows!

```
SELECT Country,COUNT(CustomerID) as c
FROM Customers
GROUP BY Country
having c > 2
```

Its ok! Having is used after we have grouped the rows!

1

2

3

4

## “WHERE” 并非在所有地方都适用！

```
SELECT 国家, 客户ID数量
来自客户表
WHERE COUNT(CustomerID) > 2
GROUP BY 国家
```

不正确! WHERE 子句不能用于聚合函数。

```
选择 国家, COUNT(客户ID)作为 c
来自 客户表
其中 c > 2
按国家分组 国家
```

不正确! WHERE 子句不能用于聚合函数 (请将 HAVING c>2 置于“GROUP BY”之后)

```
选择 国家, COUNT(客户ID)作为 c
来自 Customers
```

筛选条件为 c > 2

按 Country 分组

操作无效! 在对行进行分组之前, 不能使用 HAVING 子句!

```
选择 Country,COUNT(CustomerID) 作为 c
来自 Customers
```

按 国家 分组

筛选条件为 c > 2

没问题! HAVING 子句用于对行进行分组之后!

1

2

3

4

*Note!*

- 1) The "where" has to go before the "having" and the group by
- 2) "where" cannot be used on aggregate functions
- 3) "having" has to be used only after we used "group by" (having does not make sense if we have not grouped rows yet.)

注意! 1) The “WHERE” 子句必须置于 “HAVING” 子句及 “GROUP BY” 子句之前

2) “WHERE” 子句不能用于聚合函数

3) “HAVING” 子句仅可在使用 “GROUP BY” 子句之后使用 (若尚未对行进行分组，则 “HAVING” 子句无实际意义)

7

Sub-queries

7

子查询

## Subquery

- Remember what we said about the **closure of SQL** where the result of each SQL query is again another table.
- That means we can run a query on table A and B which results in generating table T and run another query on Table T and C
- Example
- `SELECT * FROM (SELECT * FROM Products)`
- Or
- `SELECT * FROM (SELECT * FROM Products WHERE price > 10)`

## 子查询

- 还记得我们之前提到的 SQL 的闭包性 吗？即每个 SQL 查询的结果本身又是一张表另一张表。
  - 这意味着，我们可以对表 A 和表 B 执行一个查询，从而生成新表 T，然后对表 T 和表 C 执行另一个查询。
  - **示例**
  - **选择\* 来自 (选择\* 来自 Products)**
- Or
- **选择\* 来自 (选择\* 来自 Products, 其中 price > 10)**

## Subqueries, Example 1

- Q: Write a query to return products whose price is more than the average price of all products
- Step 1: get the average price
- `Select avg(price) from products`
- The result will be a number, 28.86, ( which is actually table with only one cell)
- Step 2: Now we want to see which one of those products has price greater than 28.86:
- `SELECT * FROM [Products] where price > 28.86`
- Step 3: putting all together
- Replace 28.86 with the query from Step 1, but put it in brackets:
- `SELECT * FROM [Products] where price > (Select avg(price) from products)`

## 子查询, 示例 1

- 问题: 编写一条查询语句, 返回价格高于所有产品平均价格的商品。
- 步骤 1: 获取平均价格
- `选择 av g(price) from p产品`
- 结果将是一个数值 (28.86), 实际上是一个仅含单行单列的表。仅一个单元格)
- 步骤 2: 现在我们要查看这些产品中哪一个的价格大于 28.86:
- `SELECT* FROM [Products] WHERE price > 28.86`
- 步骤 3: 将所有内容整合起来
- 将 28.86 替换为步骤 1 中的查询, 但需将其置于括号内:
- `选择* 来自[产品] 其中价格>`  
`(选择价格平均值来自产品)`

### Subqueries Example 3:

- Q: List the most expensive product(s).
- A: We need the products' names and the prices. The table Products has both we need.
- Note: it is very important to know that we cannot assume there is only one product with the highest price. The products may be like this

| Product name |     | Price |
|--------------|-----|-------|
| Milk         | ... | 2     |
| Jeans        | ... | 10    |
| Shirt        | ... | 10    |

- Therefore the answer is
- `SELECT * FROM [Products]`
- `where price = ( select max (price) from products)`
- 

### 子查询示例 3:

- 问：列出最昂贵的 p 产品 p (或产品)
- 答：我们需要产品的名称及其价格。Products 表中已包含这两项所需信息。
- 注意：必须明确一点，我们不能假定仅存在一种价格最高的产品。  
最高价格。相关产品可能如下所示

| 产品名称 |     | 价格 |
|------|-----|----|
| Milk | ... | 2  |
| 牛仔裤  | ... | 10 |
| 衬衫   | ... | 10 |

- 因此，答案为
- 选择\* 来自 [产品]
- `WHERE price = ( SELECT MAX(price) FROM products)`
-

# Creating M:M

by example

8

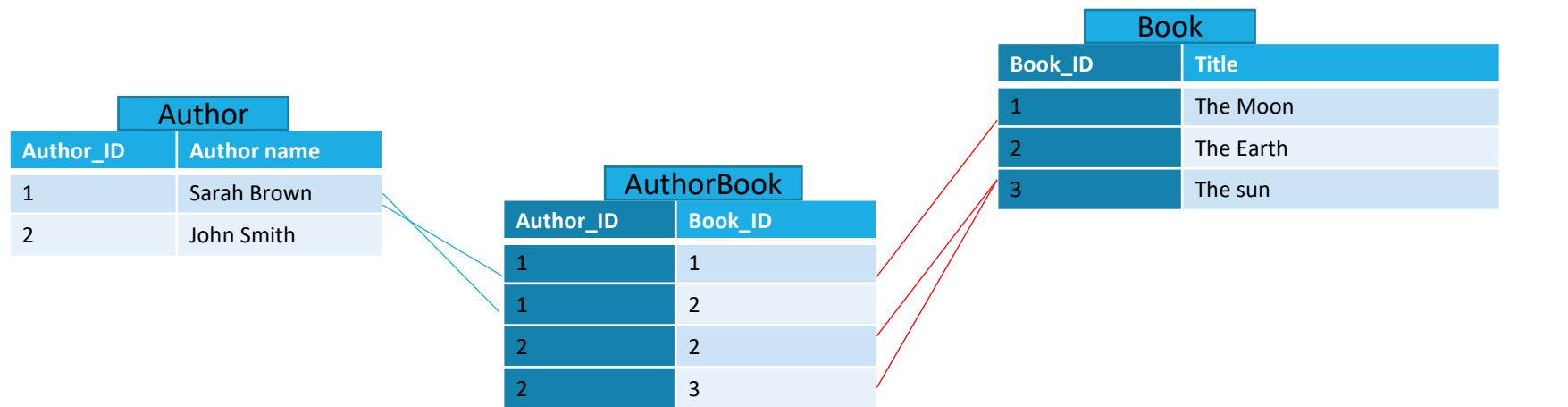
# 创建多对多关系

以示例说明

8

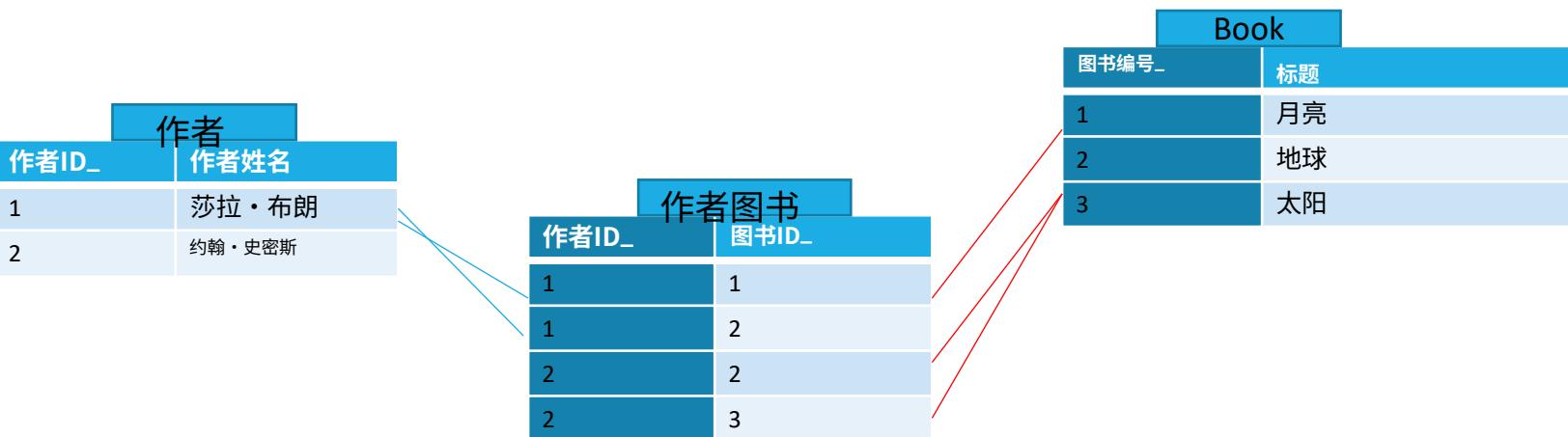
Q: Design an RDBM to model the relationship between books and their authors:

- Sarah Brown authored "The Moon"
- Sarah Brown and John Smith co-authored "The Earth"
- John Smith authored "The Sun"



Q: 设计一个关系型数据库 (RDBMS) 来建模图书与其作者之间的关系:

- 莎拉·布朗著有“月亮”
- 莎拉·布朗与约翰·史密斯合著了“地球”
- 约翰·史密斯著有“太阳”



- Q: What type of relationship between Autor and AuthorBook?
- Q: What type of relationship between Book and AuthorBook?
- A: 1:M and 1:M

- 问：作者与作者图书之间是什么类型的关系？问：图书与作者图书之间是什么类型的关系？
- A: 1:M and 1:M

## M:M sample in SQL

```
CREATE TABLE author (
 authorID INTEGER,
 name VARCHAR(50),
 PRIMARY KEY(authorID)
);
CREATE TABLE book (
 bookID INTEGER,
 title VARCHAR(30) NOT NULL,
 PRIMARY KEY(bookID)
);
CREATE TABLE authorBook (
 authorID INTEGER,
 bookID INTEGER,
 PRIMARY KEY(authorID, bookID),
 CONSTRAINT fk_has_author FOREIGN KEY(authorID) REFERENCES author(authorID),
 CONSTRAINT fk_has_book FOREIGN KEY(bookID) REFERENCES book(bookID)
);
```

What kind of primary key  
is this?! \$!!#%

Its **composite** primary  
key  
Q: why there are two  
foreign keys here?

## SQL 中的多对多 (M:M) 示 例

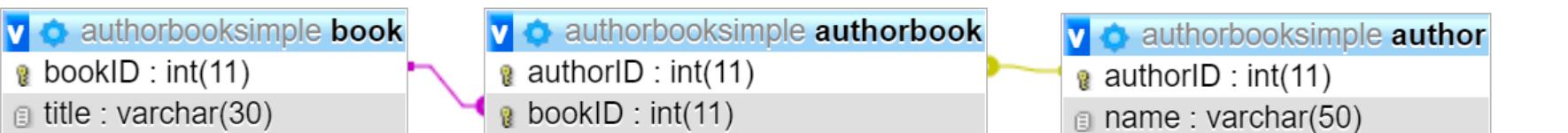
```
CREATE TABLE author (
 authorID INTEGER,
 name VARCHAR(50),
 PRIMARY KEY(authorID)
);
CREATE TABLE book (
 bookID INTEGER,
 title VARCHAR(30) NOT NULL,
 PRIMARY KEY(bookID)
);
CREATE TABLE authorBook (
 authorID INTEGER,
 bookID INTEGER,
 PRIMARY KEY(authorID, bookID),
 CONSTRAINT fk_has_author FOREIGN KEY(authorID) REFERENCES author(authorID),
 CONSTRAINT fk_has_book FOREIGN KEY(bookID) REFERENCES book(bookID)
);
```

这是哪种主键？！\$!!#%

其复合主键Q：此  
处为何存在两个外键？

## *associative entity*

- The table authorBook which links the other two tables is referred as *associative entity* or *associative table*



- Q: Now write SQL queries to insert these data into our database
- Sarah Brown authored "The Moon"
- Sarah Brown and John Smith co-authored "The Earth"
- John Smith authored "The Sun"

## 关联实体

- 将另外两张表关联起来的表 authorBook 被称为关联实体或关联表



- Q: 现在请编写 SQL 查询语句，将这些数据插入到我们的数据库中
- 莎拉·布朗撰写了“月亮”
- 莎拉·布朗与约翰·史密斯合著了“地球”
- 约翰·史密斯撰写了“太阳”

Same M:M relation, a bit different implementation of associative entity

```
CREATE TABLE author (
 authorID INTEGER,
 name VARCHAR(50),
 PRIMARY KEY(authorID)
);
CREATE TABLE book (
 bookID INTEGER,
 title VARCHAR(30) NOT NULL,
 PRIMARY KEY(bookID)
);
CREATE TABLE authorBook (
 authorBookID INTEGER,
 authorID INTEGER,
 bookID INTEGER,
 PRIMARY KEY(authorBookID, authorID),
 CONSTRAINT fk_has_author FOREIGN KEY(authorID) REFERENCES author(authorID),
 CONSTRAINT fk_has_book FOREIGN KEY(bookID) REFERENCES book(bookID)
);
```

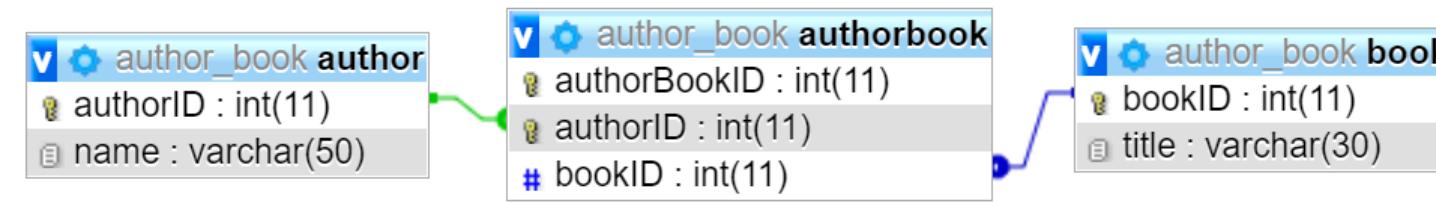
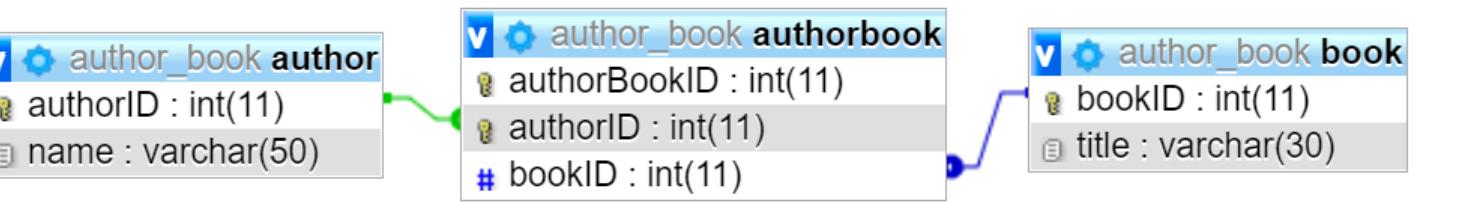
Do we need this?

相同的多对多（M:M）关系，但关联实体的实现方式略有不同

```
CREATE TABLE author (
 authorID INTEGER,
 name VARCHAR(50),
 PRIMARY KEY(authorID)
);
CREATE TABLE book (
 bookID INTEGER,
 title VARCHAR(30) NOT NULL,
 PRIMARY KEY(bookID)
);
CREATE TABLE authorBook (
 authorBookID INTEGER,
 authorID INTEGER,
 bookID INTEGER,
 PRIMARY KEY(authorBookID, authorID),
 CONSTRAINT fk_has_author FOREIGN KEY(authorID) REFERENCES author(authorID),
 CONSTRAINT fk_has_book FOREIGN KEY(bookID) REFERENCES book(bookID)
);
```

我们需要这个吗？

- The table authorBook which links the other two tables is referred as *associative entity* or *associative table*
- 用于关联另外两张表的表 authorBook 被称为关联实体或关联表



## AUTO INCREMENT Field

- allows a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.
- MySQL
- ```
CREATE TABLE Persons (
    Personid int AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    PRIMARY KEY (Personid)
);
```
- Q: Try it in your db using workbench or phpMyAdmin or in console
- Insert into Persons (LastName) values ("McDonnal")
- Note that we did not have to insert personID, why?
- Q: can we AUTO_INCREMENT non integer fields?

自动递增字段

- 在向数据表插入新记录时，可自动生成一个唯一编号。
- 通常，该字段即为主键字段，我们希望每次插入新记录时均能自动创建。
- MySQL
- ```
CREATE TABLE Persons (Personid 整数 自增_,
 LastName 可变长度字符串(255) 非空, 主键
 (Personid)
);
```
- Q: 请在数据库中使用 Workbench、phpMyAdmin 或命令行工具尝试执行该语句。
- 向 Persons 表的 LastName 字段插入值 "McDonnal"
- 请注意，我们无需插入 personID 字段，原因何在？
- 问：能否对非整数类型字段使用 AUTO\_INCREMENT？

## CHECK - Ensures that all values in a column satisfies a specific condition

- SQL Server
- `CREATE TABLE Persons (`  
    `ID int ,`  
    `LastName varchar(255) NOT NULL,`  
    `Age int CHECK (Age>=18) );`

- MySQL
- `CREATE TABLE Persons (`  
    `ID int ,`  
    `LastName varchar(255) NOT NULL,`  
    `Age int,`  
    `CONSTRAINT age_18 CHECK (Age>=18) );`

- Q: Try it in your DB with Workbench or phpMyAdmin
- Q1: try entering a person with age 5 and see what happens
- `INSERT INTO Persons (ID, Age) values( 1,5);`

- Q2: do you see any potential issue with the column Age (aside from the fact that it should be dateOfBirth, what else do you see?)
- Tip: has something to do with null
- Q3: What if we wanted to also ensure the Age<150?

What's this?  
Do we need it?

## 检查约束 – 确保列中的所有值均满足特定条件

- SQL Server
- `CREATE TABLE Persons (`  
    `ID int ,`  
    `LastName varchar(255) NOT NULL,`  
    `Age int CHECK (Age>=18) );`

- MySQL
- `CREATE TABLE Persons (`  
    `ID int ,`  
    `LastName varchar(255) NOT NULL,`  
    `年龄 (整数类型) ,`  
    `约束 a    ge_18 检查约束 (Age>=18) );`

- Q: 在 Workbench 或 phpMyAdmin 中的数据库中尝试执行
- Q1: 尝试插入一位年龄为 5 岁的人员，观察会发生什么
- 插入数据到 Persons 表 (ID, Age) 的值为 (1, 5) ;

- 问题2：除列名 Age 应为 dateOfBirth 外，您是否还发现 Age 列存在其他潜在问题？  
除此之外，您还发现了什么问题？
- 提示：该问题与空值 (NULL) 有关。
- 问题3：如果我们还想确保 Age<150，应如何处理？

这是什么?  
我们需要它吗?

## CHECK example error

- `INSERT INTO Persons (ID, Age) values( 1,5);`
- Will result in the error below:

The screenshot shows a MySQL Workbench interface. In the top query editor, the following SQL code is entered:

```
1 INSERT INTO Persons (ID, Age) values(1,5);
2
```

Below the code, there are several buttons: Clear, Format, Get auto-saved query, and a checkbox for Bind parameters. Under the Delimiter section, a semicolon is selected. A checked checkbox labeled "Show this query here again" is also present.

A red box highlights the "Error" section at the bottom, which contains the following text:

**Error**

SQL query:

```
INSERT INTO Persons (ID, Age) values(1,5)
```

MySQL said:  #4025 - CONSTRAINT `age\_18` failed for `amir1`.`persons`

## 检查示例错误

- 插入到 Persons (ID, 年龄) 取值 (1, 5) ;
- 将导致以下错误：

The screenshot shows a MySQL Workbench interface. In the top query editor, the following SQL code is entered:

```
1 INSERT INTO Persons (ID, Age) values(1,5);
2
```

Below the code, there are several buttons: Clear, Format, Get auto-saved query, and a checkbox for Bind parameters. Under the Delimiter section, a semicolon is selected. A checked checkbox labeled "Show this query here again" is also present.

A red box highlights the "Error" section at the bottom, which contains the following text:

**Error**

SQL query:

```
INSERT INTO Persons (ID, Age) values(1,5)
```

MySQL said:  #4025 - CONSTRAINT `age\_18` failed for `amir1`.`persons`

## Remarks

14

- SQL statements or clauses are NOT case sensitive
- Table names are NOT case sensitive
- `SELECT City FROM Customers;`
- `SeLeCT ciTy frOm cuStoMers;`
- **Keywords** e.g. "ORDER BY"
- **Operators** e.g. =, <>, OR
- **Clauses** e.g. WHERE
- **Statements** e.g. `select * from ...`
- **Functions** like: `min()`, `max()`
- Operators are used in clauses
- **Keyword:** SQL reserved words
- **Identifiers:** Names we choose
- Constants: Fixed values
- **Clauses:** Portion of SQL statement

## 备注

- SQL 语句或子句不区分大小写
- 表名不区分大小写
- 从 Customers 表中选择城市；
- 从客户中选择城市；
- 关键字（例如 "ORDER BY"）
- 运算符，例如 =、<>、OR
- 子句，例如 WHERE
- 语句，例如 `select * from ...`
- 函数，例如：`min()`、`max()`
- 运算符用于子句中
- 关键字：SQL 保留字
- 标识符：我们自行定义的名称
- 常量：固定不变的值
- 子句：SQL 语句的一部分

14