匠人学院
jiangren.com.au

Java Spring Boot ——SQL

数据库

- Data Persistence
- Why?
- Compared with file

非关系型数据库(NoSQL)：MongoDB，LevelDB……

关系型数据库(SQL)：Sqlite，MySQL，PostgreSQL，SQL Server……

数据库表

一个数据库通常包含一个或多个表。每个表由一个名字标识（例如“客户”或者“订单”）。

表包含带有数据的记录（行）。 下面的例子是一个存个人信息的表：

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

SQL

结构化查询语言 (SQL) 是用于访问SQL数据库的标准语言

SQL 是一门 ANSI 的标准计算机语言，用来访问和操作数据库系统。

SQL 语句用于取回和更新数据库中的数据。

SQL 可与数据库程序协同工作

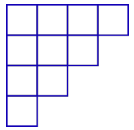
存在着很多不同版本的 SQL 语言，但是为了与 ANSI 标准相兼容，它们必须以相似的方式共同地来支持一些主要的关键词（比如 SELECT、UPDATE、DELETE、INSERT、WHERE 等等）
除了 SQL 标准之外，大部分 SQL 数据库程序都拥有它们自己的私有扩展

SQL语法

* SQL 对大小写不敏感！
* 某些数据库系统要求在每条 SQL 命令的末端使用分号。

分号是在数据库系统中分隔每条 SQL 语句的标准方法，这样就可以在对服务器的相同请求中执行一条以上的语句。

* 可以分行



安装数据库

* Postgres docker version
* pgAdmin Client

SQL 数据定义语言 (DDL)

- CREATE DATABASE 创建数据库
- CREATE TABLE - 创建新表
- ALTER TABLE - 变更（改变）数据库表
- DROP TABLE - 删除表
- CREATE INDEX - 创建索引（搜索键）
- DROP INDEX - 删除索引

CREATE DATABASE 语句

CREATE DATABASE 用于创建数据库。

SQL CREATE DATABASE 语法

```
CREATE DATABASE database_name
```

CREATE TABLE 语句

CREATE TABLE 语句用于创建数据库中的表

SQL CREATE TABLE 语法

```
CREATE TABLE 表名称
(
列名称1 数据类型,
列名称2 数据类型,
列名称3 数据类型,
....
)
```

CREATE TABLE 语句

SQL 数据类型

数据类型（data_type）规定了列可容纳何种数据类型。

下面的表格包含了SQL中最常用的数据类型：

数据类型	描述
integer (size) **int** (size) **smallint** (size) **tinyint**(size)	仅容纳整数。在括号内规定数字的最大位数。
decimal(size,d) **numeric**(size,d)	容纳带有小数的数字。"size" 规定数字的最大位数。"d" 规定小数点右侧的最大位数。
char(size)	容纳固定长度的字符串（可容纳字母、数字以及特殊字符）。在括号中规定字符串的长度。
varchar(size)	容纳可变长度的字符串（可容纳字母、数字以及特殊的字符）。在括号中规定字符串的最大长度。
date(yyyymmdd)	容纳日期。

CREATE TABLE 语句

SQL 约束

可以在创建表时规定约束（通过 CREATE TABLE 语句），或者在表创建之后也可以（通过 ALTER TABLE 语句）。

常见以下几种约束：

- NOT NULL： 不可为空
- UNIQUE：不可重复
- PRIMARY KEY：主键
- FOREIGN KEY：外键，一个表中的 FOREIGN KEY 指向另一个表中的 PRIMARY KEY（语法取决于数据版本）
- CHECK：值约束（语法取决于数据版本）
- DEFAULT：默认值

CREATE TABLE 语句

创建一个价格历史数据表

```
CREATE TABLE "price_history" (
    `ID`    INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
    `datetime`  INTEGER NOT NULL,
    `exchange`  TEXT,
    `symb`  TEXT,
    `price` REAL
)
```

CREATE INDEX 语句

创建索引

```sql
CREATE INDEX `datetime` ON `price_history` (`datetime` )
```

SQL 数据操作语言 (DML)

- SELECT – 从数据库表中获取数据
- UPDATE – 更新数据库表中的数据
- DELETE – 从数据库表中删除数据
- INSERT INTO – 向数据库表中插入数据

SELECT 语句

SELECT 语句用于从表中选取数据。

结果被存储在一个结果表中（称为结果集）。

语法

```
SELECT 列名称 FROM 表名称
```

以及：

```
SELECT * FROM 表名称
```

注释：SQL 语句对大小写不敏感。SELECT 等效于 select。

SELECT 语句

SQL SELECT 实例

```
SELECT datetime, price FROM price_history
```

SQL SELECT * 实例

请使用符号 * 取代列的名称，就像这样：

```
SELECT * FROM price_history
```

Limit 语句

```
SELECT * FROM price_history LIMIT 10
```

SELECT DISTINCT 语句

在表中，可能会包含重复值。
关键词 DISTINCT 用于返回唯一不同的值。

语法：

```
SELECT DISTINCT 列名称 FROM 表名称
```

使用 DISTINCT 关键词

```
SELECT DISTINCT exchange FROM price_history
```

WHERE 子句

如需有条件地从表中选取数据，可将 WHERE 子句添加到 SELECT 语句。

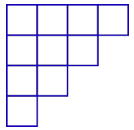
语法

```
SELECT 列名称 FROM 表名称 WHERE 列 运算符 值
```

下面的运算符可在 WHERE 子句中使用：

注释：在某些版本的 SQL 中，操作符 <> 可以写为 !=。

操作符	描述
=	等于
<>	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于
BETWEEN	在某个范围内
LIKE	搜索某种模式



WHERE 子句

```
SELECT * FROM price_history where exchange = 'OKCoin'
```

WHERE 子句

引号的使用

* SQL 使用单引号来环绕文本值（大部分数据库系统也接受双引号）
* 如果是数值，不要使用引号。

这是正确的：

```
SELECT * FROM Persons WHERE
FirstName='Bush'
```

这是错误的：

```
SELECT * FROM Persons WHERE
FirstName=Bush
```

这是正确的：

```
SELECT * FROM Persons WHERE Year>1965
```

这是错误的：

```
SELECT * FROM Persons WHERE Year>'1965'
```

WHERE 子句

匠人学院
jiangren.com.au

AND & OR 运算符

AND 和 OR 可在 WHERE 子语句中把两个或多个条件结合起来。

如果第一个条件和第二个条件都成立，则 AND 运算符显示一条记录。

如果第一个条件和第二个条件中只要有一个成立，则 OR 运算符显示一条记录。

WHERE 子句

```
SELECT * FROM price_history where exchange = 'OKCoin'
and datetime > 1510933571
```

```
SELECT * FROM price_history where exchange = 'OKCoin'  or
exchange = 'Bitfinex'
```

结合 AND 和 OR 运算符

我们也可以把 AND 和 OR 结合起来（使用圆括号来组成复杂的表达式）：

```
SELECT * FROM price_history
where
(exchange = 'OKCoin'  or exchange = 'Bitfinex')
and datetime > 1510933571
```

WHERE 子句

LIKE 操作符

LIKE 操作符用于在 WHERE 子句中搜索列中的指定模式。

SQL LIKE 操作符语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

WHERE 子句

```
SELECT * FROM price_history
where
symb like '%AUD'

SELECT * FROM price_history
where
symb not like  '%AUD'
```

WHERE 子句

BETWEEN 操作符

操作符 BETWEEN ... AND 会选取介于两个值之间的数据范围。这些值可以是数值、文本或者日期。

SQL BETWEEN 语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

WHERE 子句

```
SELECT * FROM price_history
where datetime
BETWEEN 1510933500 AND 1510933550
```

ORDER BY 子句

匠人学院
jiangren.com.au

ORDER BY 语句用于根据指定的列对结果集进行排序。

ORDER BY 语句默认按照升序对记录进行排序。

如果希望按照降序对记录进行排序，可以使用 DESC 关键字。

ORDER BY 子句

```
SELECT * FROM price_history
order by datetime
```

```
SELECT * FROM price_history
order by datetime, exchange
```

```
SELECT * FROM price_history
order by datetime DESC
```

```
SELECT * FROM price_history
order by datetime, exchange desc
```

INSERT INTO 语句

INSERT INTO 语句用于向表格中插入新的行。

语法

```
INSERT INTO 表名称 VALUES (值1, 值2,….)
```

我们也可以指定所要插入数据的列：

```
INSERT INTO table_name (列1, 列2,...) VALUES (值1, 值2,....)
```

INSERT INTO 语句

匠人学院
jiangren.com.au

插入新的行

SQL 语句：

```
INSERT INTO exchange VALUES (1, 'Bitfinex', 'https://
www.bitfinex.com/', '')
```

在指定的列中插入数据

SQL 语句：

```
INSERT INTO exchange (name,url) VALUES
 ('Bitstamp', 'https://www.bitstamp.net')
```

UPDATE 语句

Update 语句用于修改表中的数据。

语法：

```
UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值
```

UPDATE 语句

匠人学院
jiangren.com.au

更新某一行中的一个列

```
UPDATE exchange
SET url = 'https://
www.bitstamp.net/'
WHERE name = 'Bitstamp'
```

更新某一行中的若干列

```
UPDATE exchange
SET url = 'https://
www.bitstamp.net/', comments = 'OK'
WHERE name = 'Bitstamp'
```

DELETE 语句

DELETE 语句用于删除表中的行。

语法

```
DELETE FROM 表名称 WHERE 列名称 = 值
```

DELETE 语句

匠人学院
jiangren.com.au

删除某行

```
DELETE FROM
exchange
WHERE
ID = 3
```

删除所有行

可以在不删除表的情况下删除所有的行。这意味着表的结构、属性和索引都是完整的：

```
DELETE FROM table_name
```

或者：

```
DELETE * FROM table_name
```

SQL JOIN

匠人学院
jiangren.com.au

引用两个表

```
SELECT *
FROM exchange, price_history
WHERE exchange.name = price_history.exchange
```

使用 Join

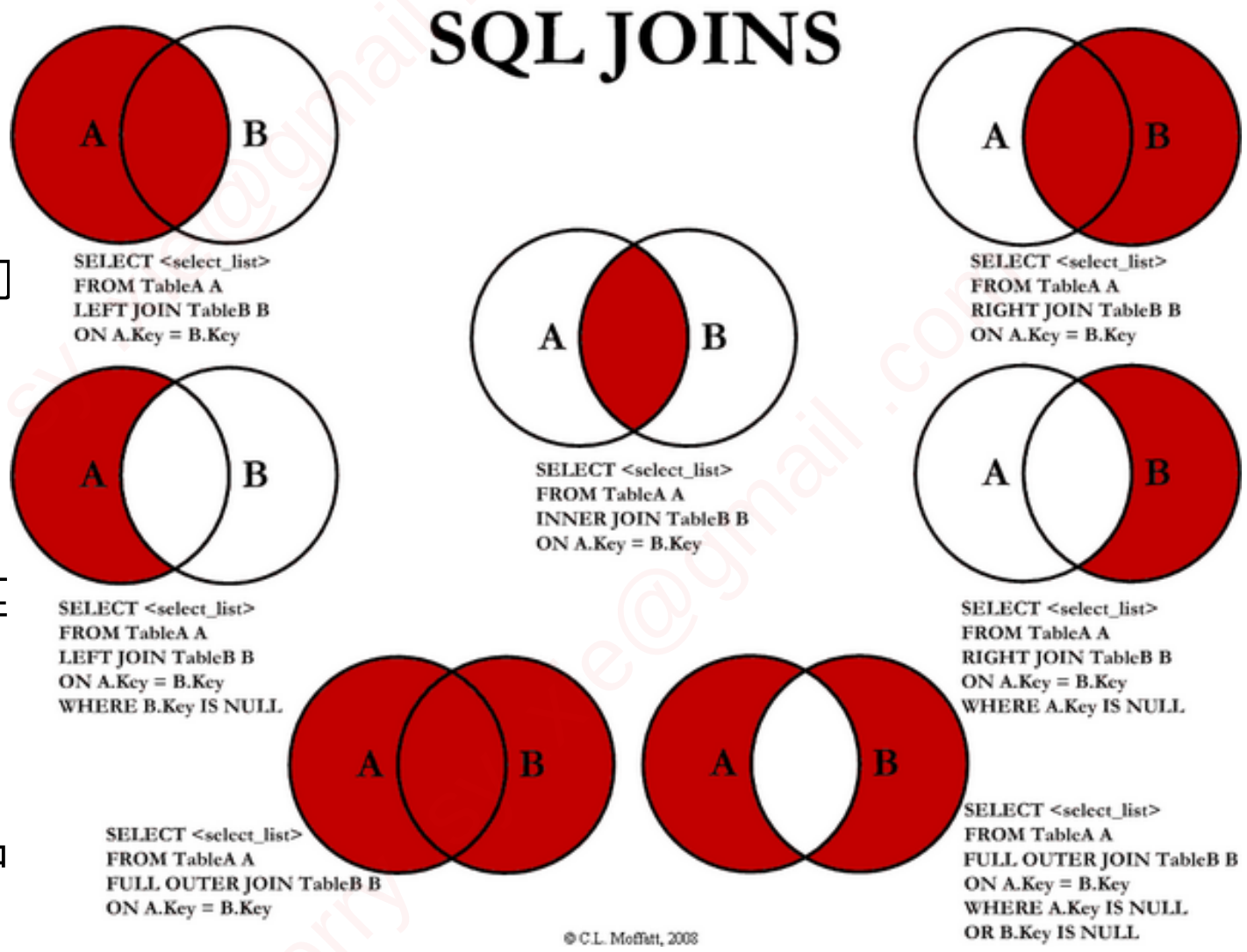
```
SELECT *
FROM price_history
INNER JOIN exchange
ON exchange.name = price_history.exchange
```

SQL JOIN

不同的 SQL JOIN

除了我们在上面的例子中使用的 INNER JOIN（内连接），我们还可以使用其他几种连接。

- JOIN: 如果表中有至少一个匹配，则返回行
- LEFT JOIN: 即使右表中没有匹配，也从左表返回所有的行
- RIGHT JOIN: 即使左表中没有匹配，也从右表返回所有的行
- FULL JOIN: 只要其中一个表中存在匹配，就返回行



UNION 操作符

SQL UNION 操作符

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。

请注意，UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。

SQL UNION 语法

```sql
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

GROUP BY 语句

语法

用于结合合计函数，根据一个或多个列对结果集进行分组。

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

GROUP BY 语句

示例

```sql
SELECT *,COUNT(price) FROM price_history
GROUP BY exchange
```

不用GROUP BY会对所有行进行计算

```sql
SELECT *,COUNT(price) FROM price_history
```

GROUP BY 语句

匠人学院
jiangren.com.au

计算平均

```sql
SELECT *,avg(price) FROM price_history
GROUP BY exchange, symb
```

HAVING 子句

在 SQL 中增加 HAVING 子句原因是，WHERE 关键字无法与合计函数结果一起使用。

记录超过30条的交易所

错误示例：

```sql
SELECT *,COUNT(price) FROM
price_history
GROUP BY exchange
WHERE COUNT(price) > 30
```

正确示范

```sql
SELECT *,COUNT(price) FROM
price_history
GROUP BY exchange
HAVING COUNT(price) > 30
```

HAVING 子句

匠人学院
jiangren.com.au

HAVING 语法

```sql
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

事务（transaction）

应用：银行账户转账问题

1. 提出A-B转账需求：取100
2. 检查A账户余额：101
3. 余额充足
4. A账户余额设为101-100=1
5. <停电了>
6. B账户余额设为0+100=100

事务（transaction）

- 一组SQL命令
  - 要么全都执行，要么全都不执行（原子性/Actomicity）
  - 在事务开始和完成时，数据都必须保持一致状态（一致性/Consistent）
  - 事务在不受外部并发操作影响的“独立”环境执行（隔离性/Isolation）
  - 事务完成之后，它对于数据的修改是永久性的（持久性/Durable）

语法（注意参考具体数据库软件的手册）

```
BEGIN TRANSACTION;
<SQL1>;
<SQL2>;
<……>;
COMMIT;
```

数据库设计

降低冗余，消除依赖

Project Number	Project Name	Employee Number	Employee Name	Salary Category	Salary Package
100001	TPMS	200001	Johnson	A	2000
100001	TPMS	200002	Christine	B	3000
100001	TPMS	200003	Kevin	C	4000
100002	TCT	200001	Johnson	A	2000
100002	TCT	200004	Apple	B	3000

- <Project Name>部分依赖于主键中的<Project Number>
- <Employee Name>，<Salary Category>和<Salary package>都部分依赖于主键中的<Employee Number>

数据库设计

降低冗余，消除依赖

Salary Category	Salary Package
A	2000
B	3000
C	4000

Project Number	Project Name
100001	TPMS
100002	TCT

Project Number	Employee Number
100001	200001
100001	200002
100001	200003
100002	200001
100002	200004

Employee Number	Employee Name	Salary Category
200001	Johnson	A
200002	Christine	B
200003	Kevin	C
200004	Apple	B

工作中真的需要掌握SQL吗？

- 需要！
- ORM的性能不够好，做report的时候需要用sql
- Data migration （增加修改column，增加修改table，数据处理等）
- Trouble shooting
- 性能调优

匠人学院
jiangren.com.au

Q & A