**Notes on SQL**

07/30/2018

[andyo@olsensoft.com](mailto:andyo@olsensoft.com) [schaua@augustschau.com](mailto:schaua@augustschau.com)

download notepad++; download mysql

MYSQL commands:

Run mysql.exe

mysql> show databases;

mysql> use test;

mysql> show tables;

mysql> select \* from person;

cd C:\mysql-5.6.14-winx64\bin

mysql -u –root

source C:\Users\DELL 2010\Desktop\SQL Overview for Citi\Labs\SqlDev\Scripts\Create.sql

source C:\Users\DELL 2010\Desktop\SQL Overview for Citi\Labs\SqlDev\Scripts\Populate.sql

show tables;

describe EMPLOYEES;

SELECT NAME, REGION FROM EMPLOYEES;

SELECT NAME, REGION FROM EMPLOYEES WHERE SALARY > 30000;

SELECT NAME, STARTDATE, STARTSALARY

FROM EMPLOYEES, CONTRACTS

WHERE EMPLOYEES.EMPLOYEEID = CONTRACTS.EMPLOYEEID;

SELECT e.employeeID, startsalary, startdate

FROM EMPLOYEES e, CONTRACTS c

WHERE e.employeeID = c.employeeID; [two tables have column named “employeeID”]

(note that the primary key “EmployeeID” auto-incremental)

INSERT INTO MYSCHEMA.EMPLOYEES (Name, Salary, Region) VALUES

('Andy', 25000.00, 'South Wales'),

('Claire', 37000.00, 'Kent'),

('Mary', 42000.00, 'London'),

('Mungo', 47000.00, 'Cumbria'),

('Midge', 72000.00, 'Scotland'),

('Hayley', 69000.00, 'Northern Ireland'),

('Nicki', 22000.00, 'Kent'),

('Sara', 11000.00, 'Kent'),

('Fiona', 88000.00, 'Kent');

UPDATE MYSCHEMA.EMPLOYEES

SET Salary=50000

WHERE EmployeeID=1;

DELETE FROM MYSCHEMA.CONTRACTS

WHERE EmployeeID=1;

SELECT NAME, SALARY, REGION

FROM EMPLOYEES

WHERE (REGION = 'Kent' OR REGION = 'London')

AND SALARY <= 30000;

SELECT NAME, SALARY, REGION

FROM EMPLOYEES

WHERE REGION IN ('Kent', 'London')

AND SALARY <= 30000;

SELECT NAME, SALARY, REGION

FROM EMPLOYEES

WHERE REGION NOT IN ('Kent', 'London', 'Cumbria')

AND SALARY <= 30000;

1. HR

Run mysqld.exe first, and then run mysql.exe

Run create and populate; exit and then re-enter mysql server to do the following commands.

# inner join: join…on Useful if tables don't have common named join columns

join…using Useful if tables have common named join columns

## inner join: To get every department names and manager names

Select d.department\_name, e.first\_name, e.last\_name

From department d join employees e

On d.manager\_id = e.employee\_id

## inner join: To get every location street\_address, and the corresponding postal\_code, country\_name, region\_name

select street\_address, postal\_code, city, country\_name, region\_name

from locations

join countries using (country\_id)

join regions using (region\_id);

#OR when the foreign key has inconsistent names across tables

select street\_address, postal\_code, city, country\_name, region\_name

from locations l join countries c join regions r

on l.country\_id=c.country\_id and c.region\_id=r.region\_id;

# outer join: Always returns rows from the first table Even if there is no corresponding row in the second table

## LEFT OUTER JOIN: Useful where there may be no corresponding rows on RHS of join

SELECT location\_id, city, department\_name

FROM LOCATIONS LEFT JOIN DEPARTMENTS

USING (location\_id)

ORDER BY location\_id;

## RIGHT OUTER JOIN: Useful where there may be no corresponding rows on LHS of join

SELECT department\_name, city, location\_id

FROM DEPARTMENTS RIGHT JOIN LOCATIONS

USING (location\_id)

ORDER BY location\_id;

## FULL OUTER JOIN: Useful where there may be no corresponding rows on either side

SELECT last\_name, first\_name, department\_name

FROM EMPLOYEES FULL JOIN DEPARTMENTS

USING (manager\_id); [not supported by MYSQL]

# COALESCE(): return the first non-null value

SELECT COALESCE(

(SELECT phone\_int FROM EMPLOYEES WHERE employee\_id = 119),

(SELECT email FROM EMPLOYEES WHERE employee\_id = 119)

) contact\_info

FROM dual;

# Use the LIKE operator

-Use % to represent one or more characters

-Use \_ to represent one character

SELECT employee\_id, last\_name, first\_name, hire\_date

FROM EMPLOYEES

WHERE last\_name LIKE 'B%'

ORDER BY last\_name;

SUBQUERIES

# to get the name of the employee with the highest salary

SELECT last\_name, first\_name

FROM EMPLOYEES

WHERE salary = **(SELECT MAX(salary) FROM EMPLOYEES)**;

# display employee salaries compared against the company average

SELECT last\_name, first\_name, salary,

ROUND((SELECT AVG(salary) FROM EMPLOYEES)) avg\_sal

FROM EMPLOYEES

WHERE last\_name LIKE 'K%';

# find DEPARTMENTS with higher avg salary than company avg

SELECT department\_id, ROUND(avg(salary)) dept\_avg\_sal

FROM EMPLOYEES

GROUP BY department\_id

HAVING avg(salary) > **(SELECT AVG(salary) FROM EMPLOYEES)**

ORDER BY dept\_avg\_sal DESC;

# in bracket nested query: find the max salary for each department; then find the corresponding name of the employee with highest salary within each department

SELECT last\_name, first\_name, department\_id, salary

FROM EMPLOYEES

WHERE (department\_id, salary) IN

(SELECT department\_id, max(salary)

FROM EMPLOYEES

GROUP BY department\_id)

ORDER BY department\_id;

# See which EMPLOYEES are also managers

SELECT employee\_id, last\_name, first\_name

FROM EMPLOYEES e

WHERE EXISTS

(SELECT 'ANY LITERAL WILL DO HERE'

FROM EMPLOYEES

WHERE e.employee\_id = manager\_id);

# determine the number of non-manager EMPLOYEES in a company

SELECT count(\*)

FROM EMPLOYEES e

WHERE NOT EXISTS

(SELECT 'X'

FROM EMPLOYEES

WHERE e.employee\_id = manager\_id);

**Notes on NoSQL**

07/31/2018

Run mongod.exe then run mongod.exe (do not close the previous prompt)

Data will be stored in data/db

BSON has many more standard data types than JSON

See <https://docs.mongodb.com/manual/reference/bson-types/>

# create a document

var emp1 = {

\_id: ObjectId("21aa914e0405a59ce30a94a2"), // Unique ID for this object.

name: { first: "Ola", last: "Nordmann" }, // Embedded document.

dob: new Date('Jul 2, 1997'), // Date object.

langs: [ "Norwegian", "Swedish", "English" ], // Array of strings.

views: NumberLong(1250000) // 64-bit long integer.

}

* To access a field in a document:
  + Use dot notation (e.g. emp1.name)
  + To access an element in an array:
  + Use [] notation and specify a zero-based index
* By default, documents in a collection don't have to have the same schema
* To create documents in a collection, call:
  + insertOne() - insert a single document into a collection
  + insertMany() - insert an array of documents into a collection

# insert one object to “people”, if the collection “people” does not exist, automatically create one;

# MongoDB generates unique \_id fields if not specified

db.people.insertOne(

{ name: "Jayne", age: 52, gender: "F" }

)

# documents can have different sets of data

db.people.insertMany([

{ name: "Thomas", age: 20, gender: "M" },

{ name: "Emily", age: 20, gender: "F", favTeam: "Swans" }

])

# call find() without any parameters, it returns all the documents in the collection: Analogous to SELECT \* in SQL

db.people.find()

* You can pass a query filter document into find(), and specify the conditions in {}
* Here are some of the query operators you can use:
  + $eq, $ne, $gt, $gte, $lt, $lte, $in, $nin
  + $and, $or, $nor, $not
  + $exists, $type
  + $mod, $regex, $text, $where
  + For full details about these query operators and more, see <https://docs.mongodb.com/manual/reference/operator/query/>

db.people.find({

name: 'Jayne'

})

db.people.find({

age: { $gte: 20 }

})

db.people.find({

age: { $gte: 20 },

age: { $lte: 30 }

})

db.people.find({

$or: [

{ age: { $lt: 20 } },

{ age: { $gt: 30 } }

]

})

# regular exppressions: /^J/ start with J; /J$/ end with J

db.people.find({

name: /^J/,

gender: 'F',

$or: [

{ age: { $lt: 20 } },

{ age: { $gt: 30 } }

]

})

* You can pass a projection document into find()
  + Specify the fields to include/exclude in the result documents
  + To specify fields to include, set fields to 1
  + To specify fields to exclude, set fields to 0
  + Apart from \_id, you can't combine 1s and 0s in your projection

db.people.find(

{ name: 'Jayne' },

{ age: 1, gender: 1 }

)

db.people.find(

{ name: 'Jayne' },

{ name: 0, \_id: 0 }

)

db.people.find(

{ name: 'Jayne' },

{ age: 1, gender: 1, \_id: 0 }

)

* To update existing documents in a collection, call:
  + updateOne() - update a single document in a collection
  + updateMany() - update an array of documents in a collection
  + replaceOne() - replace a single document in a collection
* For updateOne() and updateMany(), pass 3 params:
  + Filter, same as for find()
  + Update to perform (e.g. $set, $unset, etc.)
  + Options object:
    - upsert - If true, will cause an insert if no matching document found
    - writeConcern - Details about how to perform the "write" operation
    - collation - Language-specific rules for string comparison (locale etc.)
* replaceOne() is the same, except the 2nd param is the replacement object

db.people.updateOne(

{ name: 'Jayne' },

{ $set: { name: 'JAYNE', favTeam: 'Swans' } }

)

db.people.updateMany(

{},

{ $inc: { age: 1 } }

)

db.people.updateMany(

{},

{ $rename: { favTeam: 'favouriteTeam' } }

)

db.people.updateMany(

{},

{ $currentDate: {

datestamp: { $type: 'date' },

timestamp: { $type: 'timestamp' }

}

}

)

db.people.replaceOne(

{ name: 'JAYNE' },

{ name: 'Jayne', age: 52, gender: 'F' }

)

* To delete documents in a collection, call:
  + deleteOne() - delete a single document in a collection
  + deleteMany() - delete an array of documents in a collection
    - For both these methods, pass 2 params:
  + Filter, same as for find()
  + Options object:
    - writeConcern - Details about how to perform the "write" operation
    - collation - Language-specific rules for string comparison (locale etc.)

db.people.deleteOne(

{ name: 'Wilfried' }

)

db.people.deleteMany(

{ favouriteTeam: 'Cardiff' }

)

db.people.deleteMany(

{ overdraft: { $exists: true } }

)

* MongoDB has various other useful collection operations available, including:
  + aggregate()
  + bulkWrite()
  + count(), totalSize(), explain(), distinct(),
  + createIndex(), dropIndex(), reIndex(),
  + findAndModify(), findAndReplace(), findAndDelete()
  + mapReduce()
  + remove()
  + For full details, see: http://docs.mongodb.com/manual/reference/method/js-collection/

Cd\

Cd “Program Files\MongoDB”

Cd server

Cd 3.6

Cd bin

Mongod

Open another command prompt

cd “Program Files\MongoDB\Server\3.6\bin”

Mongo

Open mongodb compass community, connect to localhost

Host name: 172.31.22.78

Port: 27017