

# Backend Development– W2S2 SQL

Cyrille Jegourel – Singapore University of Technology and Design



SINGAPORE UNIVERSITY OF  
TECHNOLOGY AND DESIGN

# Outline (Day 2, Sessions 1 and 2)

- SQL: intro
- Data creation and manipulation
- Query
- Output control
- Nested queries

# Structure Query Language

- SQL

- Pronounced SEQUEL...

- A set of standards for:

- Data definition:
    - create/delete tables
  - Data manipulation:
    - update
  - Query language

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin  
Raymond F. Boyce

IBM Research Laboratory  
San Jose, California

# Structure Query Language

- History:

- First by Chamberlin & Boyce
- SQL '92
- SQL 3 ('99)
- SQL/XML ('06)
- SQL: 2008
- SQL: 2011

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin  
Raymond F. Boyce

IBM Research Laboratory  
San Jose, California

# SQL

- MySQL is not another SQL
  - It's a DBMS!
- Implemented in all major DBMS
  - Each may slightly deviate from the standard



# SQL

- It's a programming language
- vs. Python/C++/Java:

**Payroll**

<b>UserID</b>	<b>Name</b>	<b>Job</b>	<b>Salary</b>
<b>123</b>	<b>Jack</b>	<b>TA</b>	<b>50000</b>
<b>345</b>	<b>Allison</b>	<b>TA</b>	<b>60000</b>
<b>567</b>	<b>Magda</b>	<b>Prof</b>	<b>90000</b>
<b>789</b>	<b>Dan</b>	<b>Prof</b>	<b>100000</b>

**Regist**

<b>UserID</b>	<b>Car</b>
<b>123</b>	<b>Charger</b>
<b>567</b>	<b>Civic</b>
<b>567</b>	<b>Pinto</b>
<b>345</b>	<b>Tesla</b>

Question: Find the job of any person who drives a Tesla?

# SQL

Question: Find the job of any person who drives a Tesla?

```
for r1 in Payroll:
    for r2 in Regist:
        if (r1.UserID == r2.UserID) and (r2.Car=="Tesla"):
            print(r1.Job)
```

**Imperative**  
(How you do it)

- + C/C++, Java, Go, etc.
- + Better control
- Do your own optimization

```
Select Job from Payroll p, Regist r
Where p.UserID = r.UserID and r.Car = "Tesla"
```

**Declarative**  
(What you want)

- + SQL, HTML
- + Easy to use
- + Someone else optimizes for you

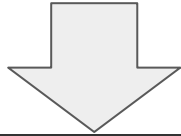
# Data Creation & Manipulation



# SQL Statement

- Create database and tables

**Payroll (UserID, Name, Job, Salary)**



```
create table Payroll (  
  UserID integer,  
  Name varchar(100),  
  Job varchar(100),  
  Salary integer  
);
```

```
create database if not exists university;
```

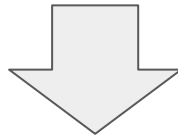
Case insensitive (except for Table name).

But *please* don't capitalize everything.

# SQL Statement

- Create a new table with primary key

**Payroll (UserID, Name, Job, Salary)**

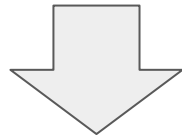


```
create table Payroll (  
    UserID integer primary key,  
    Name varchar(100),  
    Job varchar(100),  
    Salary integer  
);
```

# SQL Statement

- Create a new table with primary key

Regist (UserID, Car)



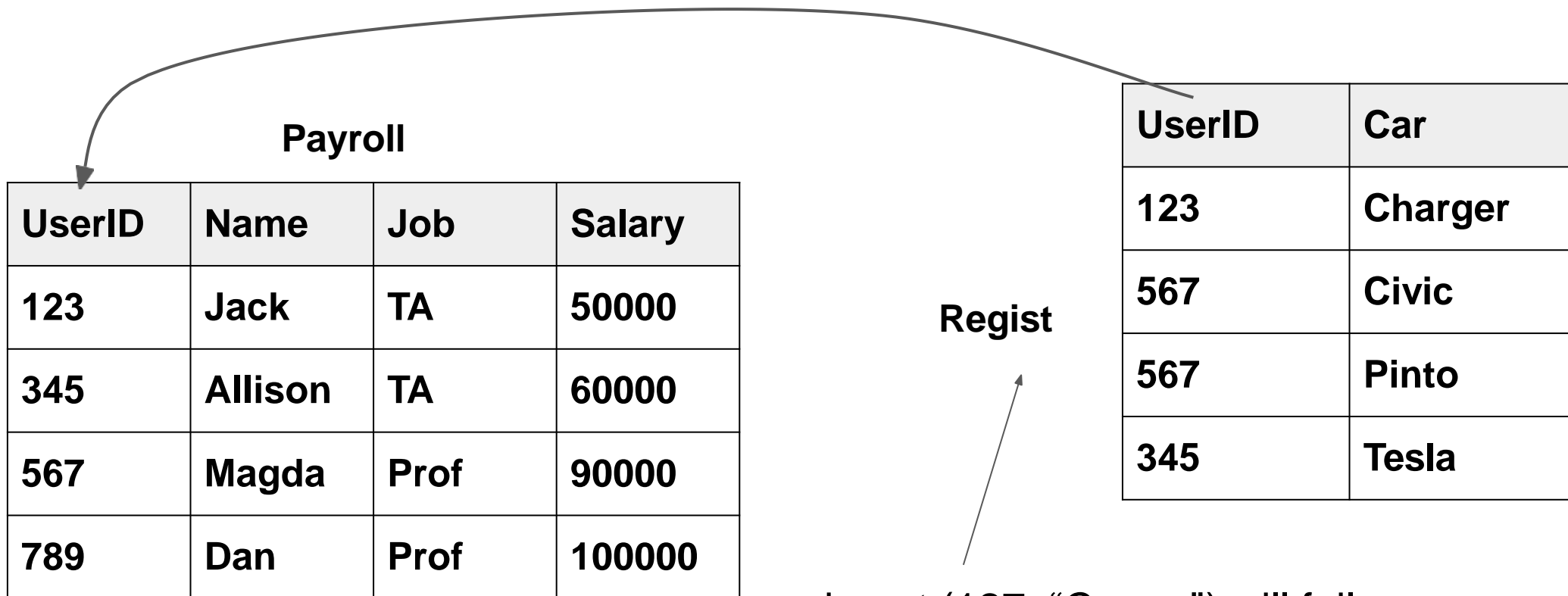
```
create table Regist (  
  UserID integer,  
  Car varchar(100),  
  primary key (UserID, Car)  
);
```

Why not making ***all fields*** as a ***default key***?



# SQL Statement

- Create a new table with foreign key



insert (127, "Camry") will fail

# SQL Statement

- Create a new table with foreign key

**Regist (UserID, Car)**

```
create table Regist (  
    UserID integer,  
    Car varchar(100),  
    primary key (UserID, Car),  
    foreign key (UserID) references Payroll(UserID)  
);
```

# SQL Statement

- You guess what these do.

```
drop table Payroll;
```

```
drop table if exists Payroll;
```



# SQL Statement

- Add new data into table

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000
001	Anh	Prof	10000
002	Cyrille	TA	10000

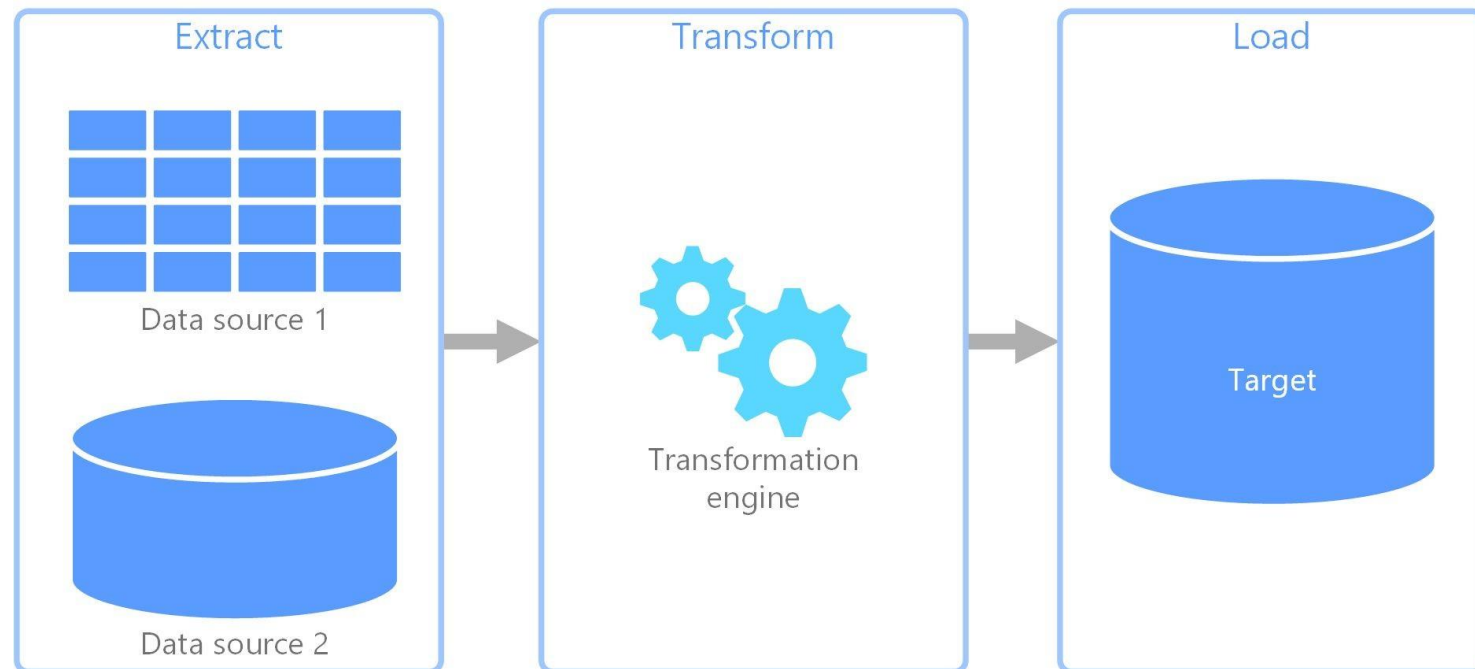
```
insert into Payroll (UserID, Name)
      values (003, "Bob");
```

```
insert into Payroll values (001, "Bob", Prof, 20000);
```



# SQL Statement

- We don't normally insert one by one
  - There may be millions of tuples
- We use tools!
  - They even have a name
- ETL



# SQL Statement

- Load data in bulk:

- From csv
- Or other formats.



```
load data infile "payroll.csv" into table Payroll
fields terminated by ',' Enclosed by '"'
lines terminated by '\n'
ignore 1 rows;
```

# SQL Statement

- Now try this!
  - **Regist** table without specifying primary key (userID, Car)

```
insert into Regist values  
  (123, "Charger"), (567, "Civic"),  
  (567, "Pinto"), (345, "Tesla");
```

```
create table Regist (  
  UserID integer,  
  Car varchar(100),  
);
```

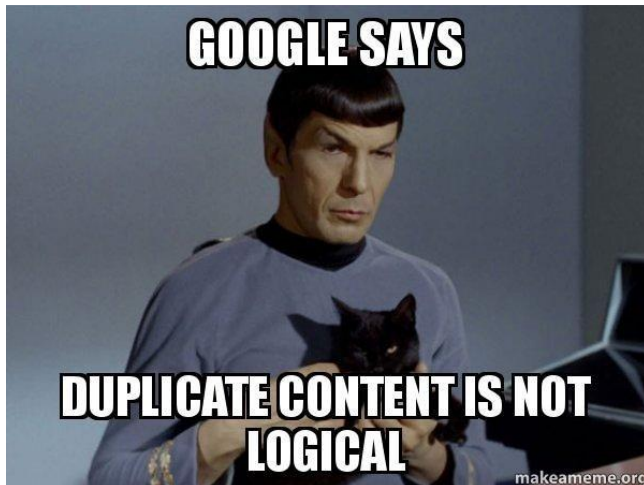
UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

# SQL Statement

- Now try this!
  - **Regist** table without specifying primary key (userID, Car)

```
insert into Regist values (123, "Charger");
```

```
create table Regist (  
    UserID integer,  
    Car varchar(100),  
);
```



DUPLICATES!



UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla
123	Charger

Query

# SQL Queries



# SQL Queries



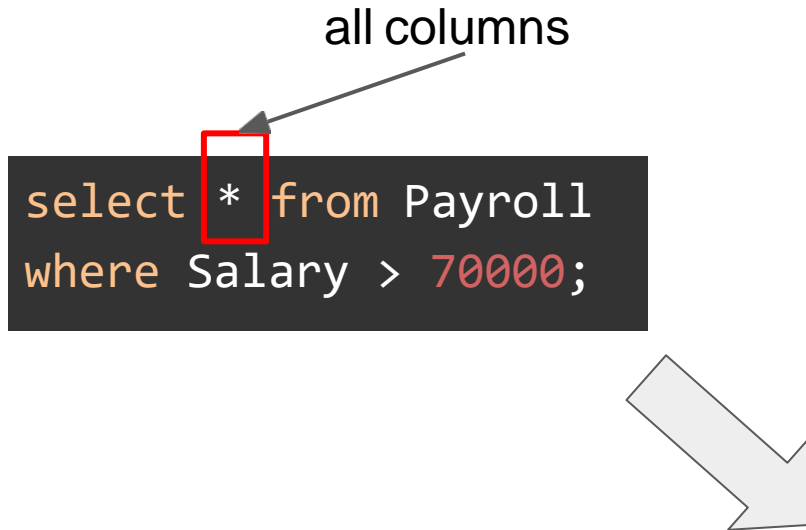
1. Take product of input relation R1, R2,...
2. Apply selection condition
3. Take specific column Col1, Col2, ...

# SQL Queries

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

all columns

```
select * from Payroll  
where Salary > 70000;
```



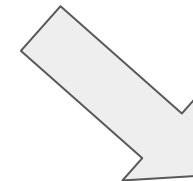
UserID	Name	Job	Salary
567	Magda	Prof	90000
789	Dan	Prof	100000



# SQL Queries

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
select UserID, Salary from Payroll  
where Salary > 50000;
```



UserID	Salary
345	60000
567	90000
789	100000

# SQL Query

Join: bread and butter!

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

Alias: R <alias>

Join condition (equi-join)

```
select p.Name, r.Car
from Payroll p, Regist r
where p.UserID = r.UserID
```

Name	Car
Jack	Charger
Magda	Civic
Magda	Pinto
Allison	Tesla

$$R1 \bowtie_{\text{condition}} R2 = \sigma_{\text{condition}} (R1 \times R2)$$

# SQL Queries

## Left Outer Join

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

```
select p.Name, r.Car
from Payroll p left outer join Regist r
on p.UserID = r.UserID
```

Name	Car
Jack	Charger
Allison	Tesla
Magda	Civic
Magda	Pinto
Dan	NULL

# SQL Queries

- Find person who drives a Civic AND a Pinto.

```
select p.Name, r.Car  
from Payroll p, Regist r  
where p.UserID = r.UserID  
and r.Car = "Civic" and r.Car = "Pinto"
```

**Won't work!**

# Self Join

- Join a relation with itself
  - Very common pattern in graph-like queries

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

```
select p.Name, r1.Car, r2.Car
from Payroll p, Regist r1, Regist r2
where p.UserID = r1.UserID
and r1.UserID = r2.UserID
and r1.Car = "Civic" and r2.Car = "Pinto"
```

Name	r1.Car	r2.Car
Magda	Civic	Pinto

# Output Control

# Output Control

- So far, output is a relation
  - A bag of tuples
- But we sometimes don't want the whole bag!
  - Summaries often better

# Output Control

- **DISTINCT(.)**: eliminate duplicates
  - Enforce set semantics

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
select Job from Payroll;
```

Job
TA
TA
Prof
Prof

```
select distinct(Job) from Payroll;
```

Job
TA
Prof



# Output Control

- Aggregates:
  - Return a summary from a bag of tuples
  - *Apply only to columns in SELECT list*

AVG
MIN
MAX
SUM
COUNT

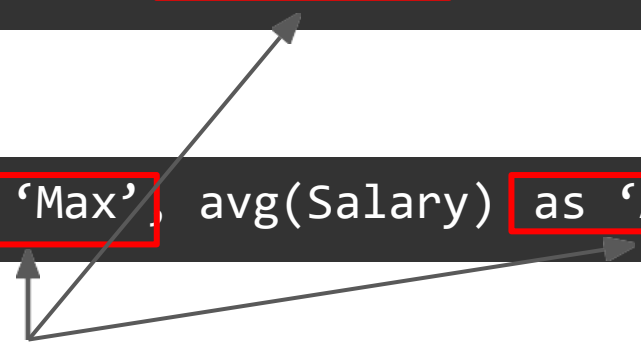
```
select count(distinct(Car)) as 'NoCar' from Regist;
```

NoCar
4

```
select max(Salary) as 'Max', avg(Salary) as 'Avg' from Payroll;
```

Max	Avg
100000	75000

Names of the output columns



# Output

- Aggregate Semantics:
  - Always applied LAST!

```
select avg(Salary) as 'Avg'  
from Payroll p, Regist r  
where p.UserID = r.UserID;
```

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

**Avg**

**72500**

# Output Control

- Group By:
  - **Extremely useful**
  - Project tuples into *distinct groups*, then compute aggregate

Attribute to group

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Group 1 → Aggregate on the remaining fields

Group 2 → Aggregate on the remaining fields

```
select Job, avg(Salary) as 'AvgPerJob'
from Payroll group by Job;
```

Job	AvgPerJob
TA	55000
Prof	95000

# Output Control

- Group By:
  - Non-aggregated attributes must appear in Group By

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
select Job, avg(Salary) as 'AvgPerJob', Name  
from Payroll group by Job;
```



# Output Control

- HAVING

- Selection on output of Group By

```
select Job, avg(Salary) as 'AvgPerJob'  
from Payroll group by Job  
HAVING AvgPerJob > 60000;
```

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Job	AvgPerJob
Prof	95000

# Output Control

- LIMIT
  - Restrict the number of output tuples

```
select * from Payroll limit 2;
```

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000

# Output Control

- ORDER BY

- Sort the tuples by values of one or more columns
- ASC | DESC: ascending or descending

```
select Name, Salary from Payroll  
order by Salary desc;
```

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Name	Salary
Dan	100000
Magda	90000
Allison	60000
Jack	50000

# Nested Queries



# Nested Queries

- Usually used as the last resorts:
  - They are often difficult to optimize
  - There could be non-nested ways to compute the same thing

**Student** (sid, name, login, pga)

**Enrolled** (sid, cid, grade)

**Course** (cid, name)

# Nested Queries

- Nested query:
  - Like a function returning a bag of tuples
  - ALL: all tuples in that bag must meet a condition
  - ANY: any tuple meeting that bag is OK
  - IN: if a value is in the bag
  - EXISTS: if there are values in the bag

# Nested Queries

```
select name from Student
where sid = ANY (
    select sid from Enrolled
    Where cid = '50-043');
```

Find names of students who  
enrolled in 50043

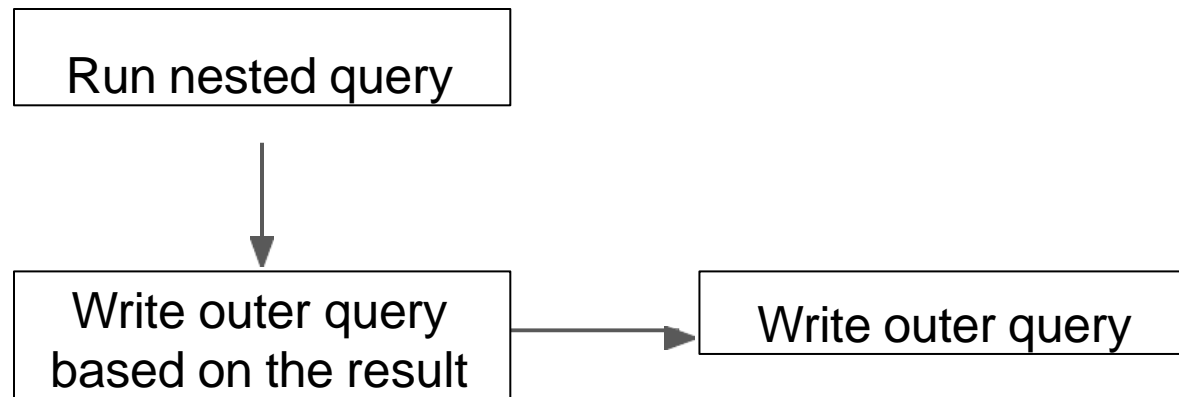
```
select name from Student
where sid IN (select max(sid) from Enrolled);
```

These two are the same

```
select name from Student
where sid >= ALL (select sid from Enrolled);
```

# How To Avoid Nested Queries

- If you can rewrite to get rid of them. DO SO!
- Another cleaner way:



See any problem?

Going further

# String Data Type

- Exact match:

```
select Name from Payroll where Name = "Anh";
```

- Wildcard: %

```
select Name from Payroll where Name like "A%";
```

- Functions:

- SUBSTR, LENGTH, etc.

```
select Name from Payroll where Length(Name) > 3;
```

# Summary

