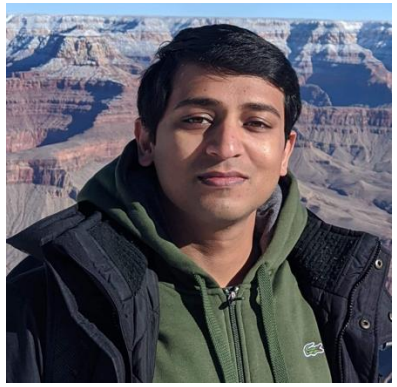
Database Processing
CS 451 / 551

Lecture 2:

Structured Query Language



Suyash Gupta

Assistant Professor

Distopia Labs and ORNG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Assignment Groups

- Assignments to be done in groups of at most 4 members.

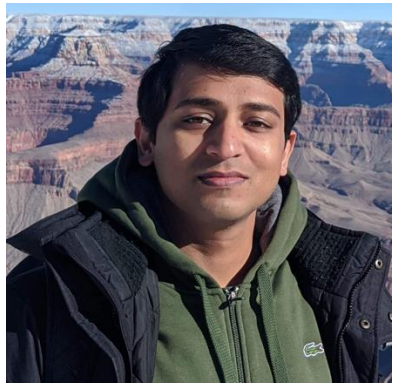
- Start forming your groups.

- **Two options:**
  - Send me your group details **by Oct 3, 2025 by 5pm PST**.
  - Or, If you cannot form a group, then send me a mail **by Oct 3, 2025 by 5pm PST.**

Database Processing
CS 451 / 551

Lecture 2:

Structured Query Language



Suyash Gupta

Assistant Professor

Distopia Labs and ORNG

Dept. of Computer Science

(E) suyash@uoregon.edu

(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Last Class

- What is a Database?
- Relational Algebra.

SQL

- Helps to query, modify, or add constraints to a database.

- Originally designed by IBM and marketed as Sequel.

- SQL has several parts:
  - Data Definition Language
  - Data Manipulation Language
  - Integrity → Helps to add integrity constraints to the database.
  - View Definition → Helps to define views.
  - Transaction control → Helps to define the beginning and end of transactions.

SQL Data Definition

SQL Data Definition

- **SQL DDL helps to specify:**
  - Schema for each relation (table).
  - Types of values for each attribute.
  - Integrity constraints on attributes.
  - Indices on a relation.
  - Security/authorization/physical storage structure of a relation.

Basic Types in SQL

Basic Types in SQL

- **char(n)** – fixed length character string of length **n**.
  - Say **char(10)** and you store a string “Alice”, then **5** spaces are added before storing.

Basic Types in SQL

- **char(n)** – fixed length character string of length **n**.
  - Say **char(10)** and you store a string "Alice", then **5** spaces are added before storing.
- **varchar(n)** – variable length character string with maximum length **n**.

Basic Types in SQL

- **char(n)** – fixed length character string of length **n**.
  - Say **char(10)** and you store a string "Alice", then **5** spaces are added before storing.
- **varchar(n)** – variable length character string with maximum length **n**.
- **int / smallint** (small integer) – machine dependent size.

Basic Types in SQL

- **char(n)** – fixed length character string of length **n**.
  - Say **char(10)** and you store a string "Alice", then **5** spaces are added before storing.
- **varchar(n)** – variable length character string with maximum length **n**.
- **int / smallint** (small integer) – machine dependent size.
- **numeric(p,d)** – fixed point number with total **p** digits (plus a sign) and **d** of the **p** digits after the decimal point.

Basic Types in SQL

- **char(n)** – fixed length character string of length **n**.
  - Say **char(10)** and you store a string "Alice", then **5** spaces are added before storing.
- **varchar(n)** – variable length character string with maximum length **n**.
- **int / smallint** (small integer) – machine dependent size.
- **numeric(p,d)** – fixed point number with total **p** digits (plus a sign) and **d** of the **p** digits after the decimal point.
- **float(n)** – floating-point number with precision of at least **n** digits.

Basic Types in SQL

- **char(n)** – fixed length character string of length **n**.
  - Say **char(10)** and you store a string "Alice", then **5** spaces are added before storing.
- **varchar(n)** – variable length character string with maximum length **n**.
- **int / smallint** (small integer) – machine dependent size.
- **numeric(p,d)** – fixed point number with total **p** digits (plus a sign) and **d** of the **p** digits after the decimal point.
- **float(n)** – floating-point number with precision of at least **n** digits.
- **null** a special value, available to all the types, indicates an absent value.
  - Suppose you don't know what value to fill for an attribute in a tuple → use null.

Create Table

Create Table

```
create table cs_employees
        (       name   varchar(20),
                age    int,
                title  varchar(30)
        );
```

Create Table: Specifying a Primary Key

```
create table cs_employees
        (       name  varchar(20),
                age     int,
                title   varchar(30),
        );
```

Create Table: Specifying a Primary Key

```
create table cs_employees
        (       name  varchar(20),
                age     int,
                title   varchar(30),
                primary key (name)
        );
```

What are the properties of primary key attribute?

Create Table: Specifying a Primary Key

```
create table cs_employees
        (       name  varchar(20),
                age     int,
                title   varchar(30),
                primary key (name)
        );
```

The values of primary key attribute are by default forced to be ***non null and unique***.

How about a Primary Key here?

```
create table courses
        (       course_name     varchar(30),
                quarter         int,
                year            int,
        );
```

Assume no attribute has unique values!

Multiple Attributes as a Primary Key

```
create table courses
        (       course_name         varchar(30),
                quarter             int,
                year                int,
                primary key (course_name, quarter, year)
        );
```

When no single attribute can guarantee a non null and unique value, then multiple attributes can together serve as a primary key.

How to constraint an Attribute to be not Null

```
create table courses
        (       course_name         varchar(30),
                quarter             int,
                year                int,
                primary key (course_name, quarter, year)
        );
```

Not Null Constraint on an Attribute

```
create table courses
        (       course_name         varchar(30),
                quarter             int not null,
                year                int,
                primary key (course_name, quarter, year)
        );
```

Now, the *quarter* attribute can not have a missing value!

Create Table: Specifying a Foreign Key

Create Table: Specifying a Foreign Key

```
create table cs_employees
        (       name    varchar(20),
                age     int,
                title   varchar(30),
                primary key (name)
        );

create table courses
        (       course_name varchar(30),
                quarter     int not null,
                year        int,
                primary key (course_name, quarter, year),
                foreign key (name) references cs-employees
        );
```

This foreign-key states that for each row in the **courses** relation, the value for **name** attribute must exist in the primary key attribute (**name**) of the **cs-employees** relation.

Inserting and Deleting Rows

Inserting and Deleting Rows

insert into cs_employees
values (‘thanos’, 100, ‘prof’);

insert into cs_employees
values (‘voldemort’, 70,); ← **Null value for this attribute is allowed!**

delete from cs_employees ← **Delete all rows from this table.**

drop table cs_employees ← **Delete table.**

SQL Querying: Reading data from Database

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs-employees

SQL Querying: Reading data from Database

```
select title
        from cs_employees;
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs-employees

output

title
Assoc Prof
Assoc Prof
Prof
Asst Prof
Prof

All the titles get displayed, even duplicates.

SQL Querying: Reading data from Database

```
select title
    from cs_employees;
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs-employees

output

title
Assoc Prof
Assoc Prof
Prof
Asst Prof
Prof

All the titles get displayed, even duplicates.

What if I want only distinct titles outputted?

SQL Querying: Reading data from Database

```
select distinct title
        from cs_employees;
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs-employees

output

title
Assoc Prof
Prof
Asst Prof

Only distinct titles get displayed.

What if I want some arithmetic on output?

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

Say I want to print the 2*age?

Select command + arithmetic operators

```
select name, age*2, title
    from cs_employees;
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs-employees

output

name	age	title
Voldemort	140	Assoc Prof
Anakin	40	Assoc Prof
Kang	40	Prof
Gru	90	Asst Prof
Thanos	200	Prof

All attributes are same except the age column, which is multiplied by 2.

Select with Condition

Select with Condition (where clause)

```
select name
        from cs_employees
        where age > 45;
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs-employees

output

name
Voldemort
Thanos

Any row where age is less than or equal to 45 is ignored.

Select with Multiple Relations

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

Select with Multiple Relations

select name, course

from cs_employees, courses;

output

name	course
Voldemort	1100
Anakin	1100
Kang	1100
…	
Voldemort	4510
…	
Voldemort	3311
…	
Voldemort	1200
…	
Voldemort	6100

All combinations of rows like relational algebra's product operation.

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

Select with Multiple Relations – Join Operation

select name, course

from cs_employees **join** courses;

output

name	course
Voldemort	1100
Anakin	1100
Kang	1100
…	
Voldemort	4510
…	
Voldemort	3311
…	
Voldemort	1200
…	
Voldemort	6100

This query and the one in previous slide are equivalent!

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

Select with Multiple Relations – Join Operation

select name, course

from cs_employees **join** courses;

output

name	course
Voldemort	1100
Anakin	1100
Kang	1100
…	
Voldemort	4510
…	
Voldemort	3311
…	
Voldemort	1200
…	
Voldemort	6100

A lot of entries don't make sense here.

Can we have a correct mapping?

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

Select with Multiple Relations

```
select name, course
    from cs_employees, courses
    where cs_employees.title = courses.title;
```

output

name	course
Anakin	1100
Voldemort	1100
Anakin	4510
Voldemort	4510
Gru	1200
Kang	3311
Thanos	3311
Kang	6100
Thanos	6100

Restricted combinations of rows using where clause.

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

Select with Multiple Relations

```
select name, course
    from cs_employees, courses
    where cs_employees.title = courses.title
    and age > 45;
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

output

name	course
Voldemort	1100
Voldemort	4510
Thanos	3311
Thanos	6100

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

Natural Join

This query, which we just saw is also termed as “natural join” in query world.

```
select name, course
        from cs_employees, courses
        where cs_employees.title = courses.title;
```

As it is so common to perform operations on multiple relations with common fields, SQL provides the ”**natural join**” operation, which does the same task.

```
select name, course
        from cs_employees natural join courses;
```

Natural Join

Let's say we have these modified tables.

select name, course

from cs_employees **natural join** courses;

Notice that there are **two common columns**!

output

name	course
Anakin	1100
Voldemort	4510
Thanos	3311
Gru	1200
Kang	6100

name	age	title	salary
Voldemort	70	Assoc Prof	600
Anakin	20	Assoc Prof	500
Kang	20	Prof	200
Gru	45	Asst Prof	400
Thanos	100	Prof	400

cs_employees

course	students	title	salary
1100	200	Assoc Prof	500
4510	35	Assoc Prof	600
3311	55	Prof	400
1200	100	Asst Prof	400
6100	5	Prof	200

courses

Are these queries same or different?

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

name	students	salary
Voldemort	35	500
Anakin	55	600
Kang	5	400
Gru	100	400
Thanos	200	200

salary

select name, course **from** cs_employees **natural join** courses **natural join** salary;

and

select name, course **from** cs_employees **natural join** courses, salary

where salary.students = courses.students.

Are these queries same or different?

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

name	students	salary
Voldemort	35	500
Anakin	55	600
Kang	5	400
Gru	100	400
Thanos	200	200

salary

select name, course **from** cs_employees **natural join** courses **natural join** salary;

and

Different!

select name, course **from** cs_employees **natural join** courses, salary

where salary.students = courses.students.

Natural Join with Three Relations

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

name	students	salary
Voldemort	35	500
Anakin	55	600
Kang	5	400
Gru	100	400
Thanos	200	200

salary

```
select name, course
from cs_employees natural join courses natural join salary;
```

output

name	course
Voldemort	4510
Gru	1200
Kang	6100

First joins these two, which results in a table with 6 columns.

Then joins the result of previous join with salary. Notice 2 common columns.

Natural Join with Three Relations

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

name	students	salary
Voldemort	35	500
Anakin	55	600
Kang	5	400
Gru	100	400
Thanos	200	200

salary

select name, course

from cs_employees **natural join** courses, salary

where salary.students = courses.students.

output

name	course
Voldemort	4510
Anakin	4510
Voldemort	1100
Anakin	1100
Gru	1200
Kang	6100
Thanos	6100

Missing from Join

- What if we want to capture the tuples skipped from the result of join operation?

Outer Join

- Helps to capture missing tuples from the result of join operation.

- Three flavors:
  - Left Outer Join – tuples missing from the left relation.
  - Right Outer Join – tuples missing from the left relation.
  - Full Outer Join – tuples missing from either of the relations.

Natural Right Outer Join

select name, course

from cs_employees **natural right outer join** courses;

output

name	course
Anakin	1100
Voldemort	1100
Anakin	4510
Voldemort	4510
Gru	1200
Kang	3311
Thanos	3311
Kang	6100
Thanos	6100
NULL	2100

Null entry for course 2100 because it has no corresponding match in cs_employees table.

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof
2100	200	Lecturer

courses

Attribute Name Change?

select name, course

from cs_employees **natural join** courses;

output

name	course
Anakin	1100
Voldemort	1100
Anakin	4510
Voldemort	4510
Gru	1200
Kang	3311
Thanos	3311
Kang	6100
Thanos	6100

Say I do not like in my display the attribute "name" and want it to be outputted as "Instructor".

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

As Clause – Renaming

select name **as** Instructor, course
from cs_employees **natural join** courses;

output

Instructor	course
Anakin	1100
Voldemort	1100
Anakin	4510
Voldemort	4510
Gru	1200
Kang	3311
Thanos	3311
Kang	6100
Thanos	6100

Output rendering

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

Any other benefits of As Clause?

As Clause – Renaming

select T.name, S.title
from cs_employees as T **natural join** courses **as** S
where age > 45;

output

name	title
Voldemort	Assoc Prof
Voldemort	Assoc Prof
Thanos	Prof
Thanos	Prof

Shortening names of the tables.

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

course	students	title
1100	200	Assoc Prof
4510	35	Assoc Prof
3311	55	Prof
1200	100	Asst Prof
6100	5	Prof

courses

As Clause – Renaming

```
select distinct T.name
from cs_employees as T, cs_employees as S
where T.age > S.age and S.name = 'Kang';
```

output

name
Voldemort
Gru
Thanos
Joker

Helps in self comparisons in a relations.

For ex: all employees with age greater than Kang.

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

String matching

String matching (%)

Makes use of a **like** operator.

select name **from** cs_employees
where title **like** 'Assoc%';

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

name
Voldemort
Anakin

All names with title starting with Assoc.

String matching (_)

Makes use of a **like** operator.

select name **from** cs_employees

where title **like** 'P_ _ _';

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

name
Kang
Thanos

All names with title starting with P and have at most three more characters.

String matching (%)

Makes use of a **like** operator.

select name **from** cs_employees
where title **like** '**Ad%t%**';

name	**age**	**title**
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

name
Joker

You can include multiple matching operators.

String matching (\)

Makes use of a **like** operator.

select name **from** cs_employees
where title **like** 'Assoc\%Prof';

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

How to print all the columns?

Select *

Asterix (*) when used with **select** clause prints all the attributes.

select * **from** cs_employees
where title **like** 'Assoc%';

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof

Filtering for Null Values

Null Values

As some attributes may have **NULL** values, we can filter our results using **is null** or **not null**.

select * **from** cs_employees
where title **is null**;

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	NULL

cs_employees

select * **from** cs_employees
where title **is not null**;

Aggregate Functions

Take a collection of values and return a single value.

- **Average (avg)**
- **Minimum (min)**
- **Maximum (max)**
- **Total (sum)**
- **Count (count)**

Average Function

select avg(age**) from** cs_employees

group by title;

name	**age**	**title**
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

avg(age)
45

Average – Group By Clause

select title, **avg(**age**) from** cs_employees
group by title;

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

title	avg(age)
Assoc Prof	45
Prof	60
Asst Prof	45
Adjunct Prof	66

We can print averages for each group; grouping sets of tuples.

Average – Group By, Having

```
select title, avg(age) from cs_employees
group by title
having avg(age) > 50;
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

title	avg(age)
Prof	60
Adjunct Prof	66

Having is like where clause but for placing constraints on groups.

Sum Function

select title, **sum(**age**) from** cs_employees

group by title;

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

title	avg(age)
Assoc Prof	90
Prof	120
Asst Prof	45
Adjunct Prof	66

We can print averages for each group; grouping sets of tuples.

Nested Subqueries

Query within a query!

select name **from** cs_employees

where title **in**

(select title **from** courses **where** course > 2000**);**

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

name
Voldemort
Anakin
Kang
Thanos

The result of evaluating the inner query serves as the constraint for the outer query.

Notice the matching field names!

Nested Subqueries

Query within a query!

select name **from** cs_employees

where title **not in**

(select title **from** courses **where** course > 2000**);**

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

name
Gru
Joker

"Not in" works as opposite to "in".

Nested Subqueries

All operation.

```
select name from cs_employees
where age > all
(select age from cs_employees
where title like 'A%');
```

name	age	title
Voldemort	70	Assoc Prof
Anakin	20	Assoc Prof
Kang	20	Prof
Gru	45	Asst Prof
Thanos	100	Prof
Joker	66	Adjunct

cs_employees

output

name
Thanos

Is there some professor who has age greater than all the non-professors.

Functions and Procedures

- SQL allows you to create your **own functions and procedures**.

```
create function dept_count(dept_name varchar(20))
        returns integer
        begin
        declare d_count integer;
                select count(*) into d_count
                from instructor
                where instructor.dept_name = dept_name
                return d_count;
        end
```

A function that, given the name of a department, returns the count of the number of instructors in that department

Self-Reading Task

- Read about how to create procedures and calling functions and procedures.