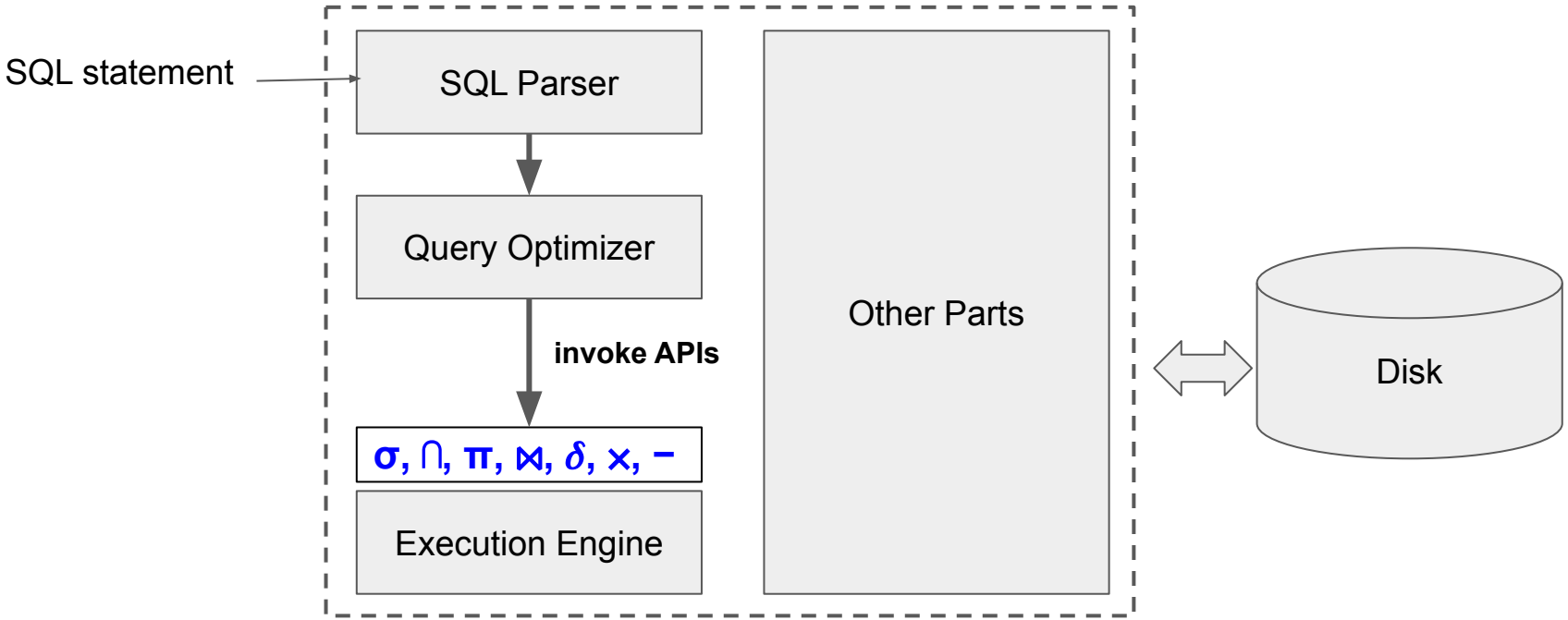
Databases and Big Data

SQL

Glimpse Into Database Internal



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 <u>programming language</u>
- vs. Python/C++/Java:

Payroll

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

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

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)
```

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

Imperative
(How you do it)

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

Declarative
(What you want)

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

Data Creation & Manipulation

SQL Statement

if strings (varchar), you can restrict the size of string

- Create database and tables

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

Payroll (UserID, Name, Job, Salary)

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

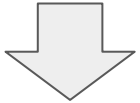
Case insentive (except for Table name).

But *please* don't capitalize everything.

SQL Statement

- Create a new table with primary key

Payroll (<u>UserID</u>, 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 (<u>UserID, Car</u>)

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

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

SQL Statement

127 does not exist in Payroll table, so insertion operation fails

- Create a new table with foreign key

Payroll

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

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

insert (127, “Camry”) will fail

SQL Statement

you can specify a tuple too
foreign key (UserID, Car) references Payroll(UserID, Car)

- Create a new table with foreign key

Regist (<u>UserID, Car</u>)

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

SQL Statement

delete table

- 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 values (001, "Anh", Prof, 10000);
insert into Payroll values (002, "Cyrille", TA, 10000);
```

=

```
insert into Payroll values (001, "Anh", Prof, 10000),
                           (002, "Cyrille", TA, 10000);
```

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

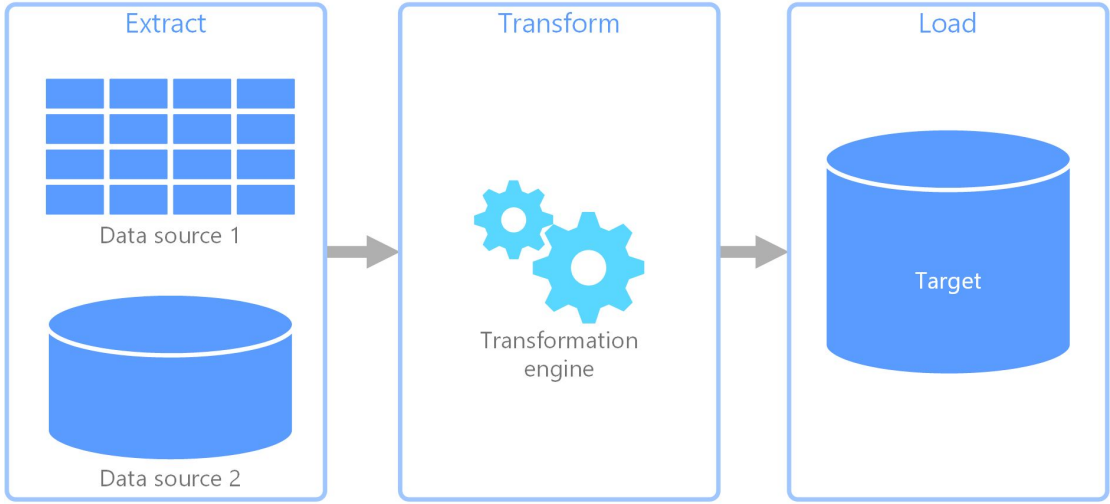
```
insert into Payroll (UserID, Name)
                    values (003, "Bob");
```

```
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

Extract, Transform, Load



SQL Statement

- Load data in bulk:
  - From csv
  - Or other formats.

have to create Payroll table first, and make sure that the .csv is in the same format as the Payroll schema

ignore 1 rows ----> ignore the header

```
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)

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

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

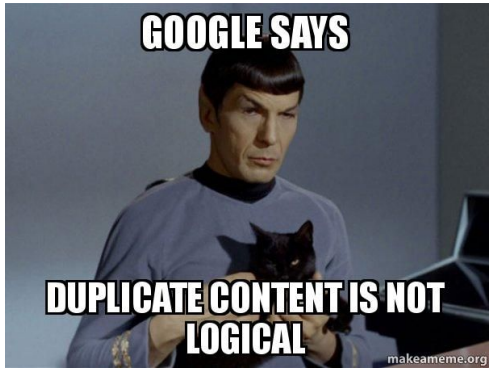
UserID	Car
123	Charger
567	Civic
567	Pinto
345	Tesla

SQL Statement

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

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

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



DUPLICATES!

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

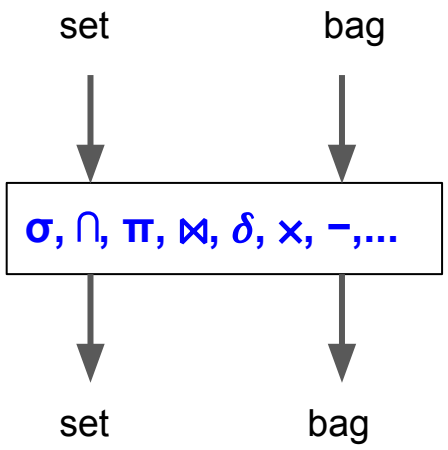
A digression

- Relational model uses set semantics
  - No duplicates
- SQL uses **bag** semantics

set ones are changed
e.g. intersect, union

Set: {1,2,3}

Bag: {1,2,1,3,1,2}



Most operators have the same meaning as before, **except** that input and output has duplicate

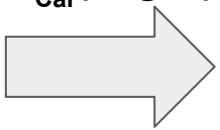
Can you guess which operators have new meaning?

A digression

- Example
- Why bags?

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

Π_{Car}(Regist)



Set

Car
Charger
Civic
Pinto
Tesla

Bag

Car
Charger
Civic
Pinto
Tesla
Charger

A digression

bag is faster
don't need to check theres no duplicates

- Why bag?
  - Which one is faster?
- When no primary keys are specified
  - = no key
- You can enforce set semantics:
  - When creating table, using UNIQUE keyword
  - When query: next slides ….

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

SQL Queries

```
SELECT Col1, Col2,..
FROM R1, R2,...
WHERE Conditions
```

Projection / Rename

Product

Selection

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

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

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!

Definitely gonna be tested

Self Join

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

```
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”
```

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

Name	r1.Car	r2.Car
Magda	Civic	Pinto

and r1.Car <> r2.Car
--> if r1.Car is not equal to r2.Car
but will have duplicates

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
shere 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

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

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*

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;
```

but you can do aggregate functions like max(length(Name))

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

- 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

Not Tested

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

Find names of students who enrolled in 50043

instead of a join

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

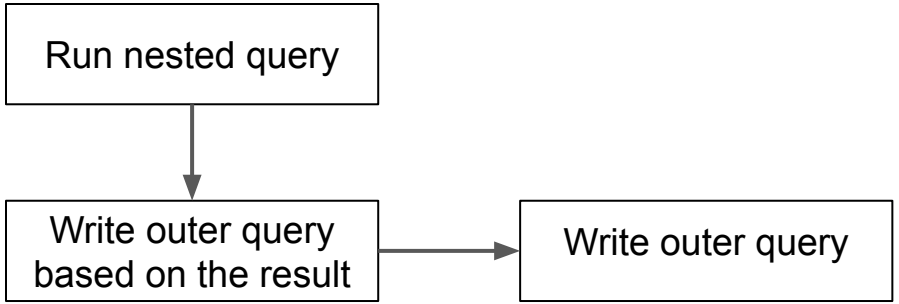
```
select name from Student
where sid > ALL (select sid from Enrolled);
```

These two are the same

Student who is enrolled and the highest student ID
can just use max

How To Avoid Nested Queries

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



See any problem?

String Data Type

- Exact match:

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

- Wildcard: %

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

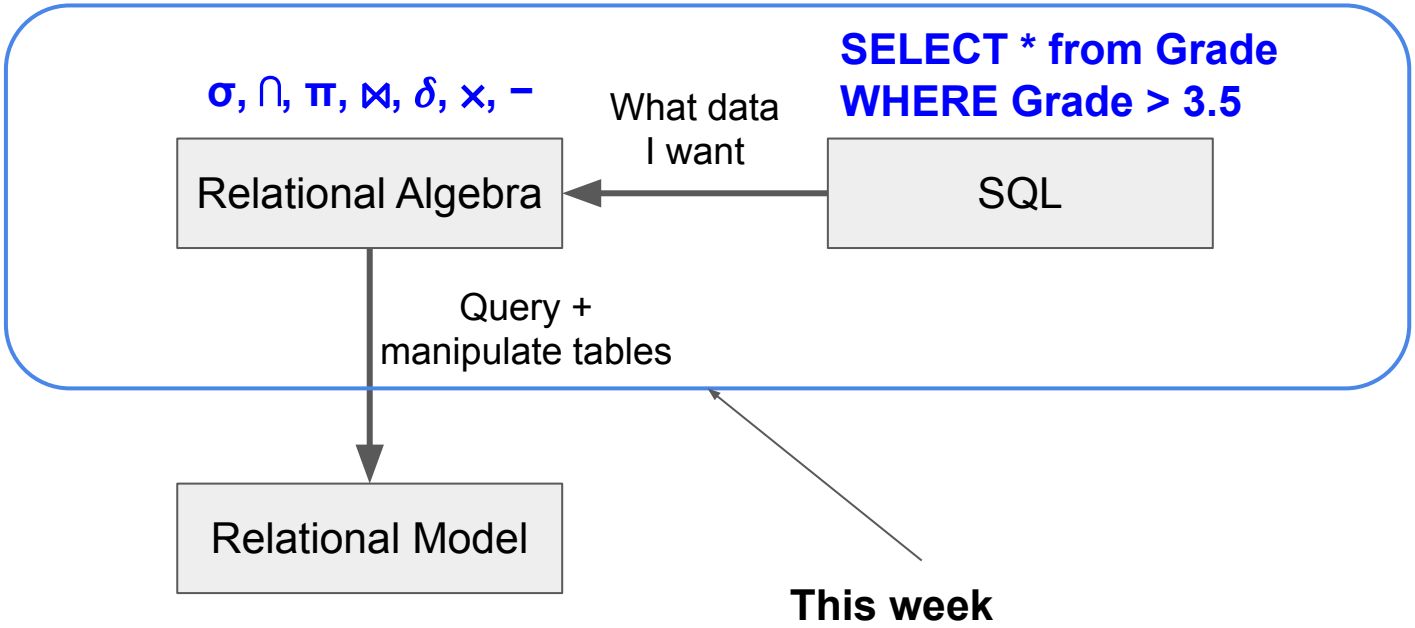
- Functions:
  - SUBSTR, LENGTH, etc.

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

Date

- Date
- DateTime
- Timestamp

Summary


σ, ∩, π, ⋈, δ, ×, −
Relational Algebra
What data
I want
SELECT * from Grade
WHERE Grade > 3.5
SQL
Query +
manipulate tables
Relational Model
This week