Discussion 11:
SQL

.....

TA: **Jerry Chen**
Email: **jerry.c@berkeley.edu**
TA Website: **jerryjrchen.com/cs61a**

Agenda

SQL

- SELECT
- JOIN
- Recursion
- Aggregation

Announcements

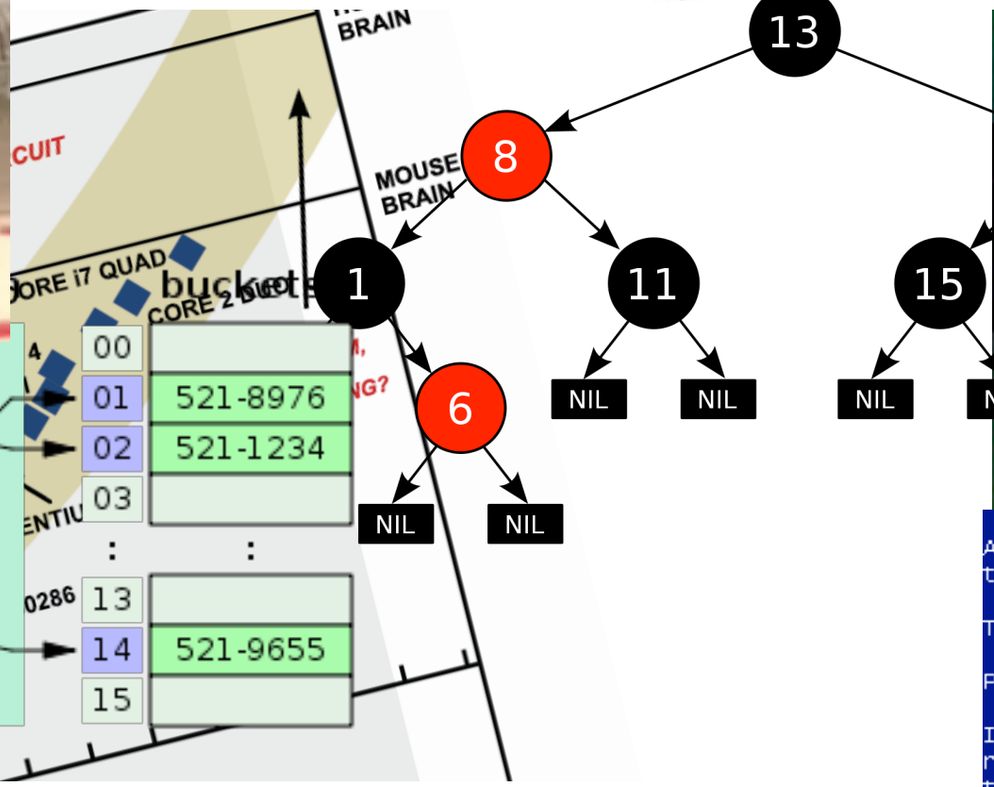
Ants composition due 04/30

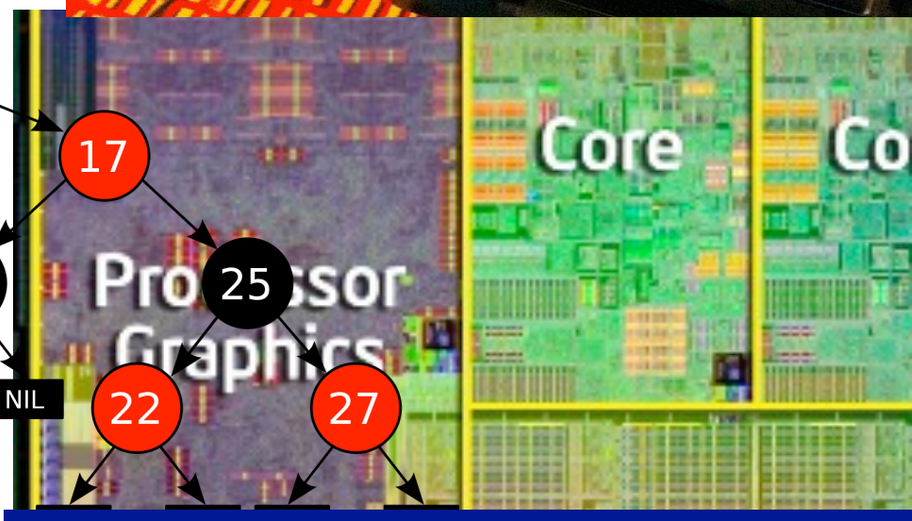
The final is Tues 05/09, submit the conflict form if needed!

Last section might be a little bit different?

Discussion 12 P

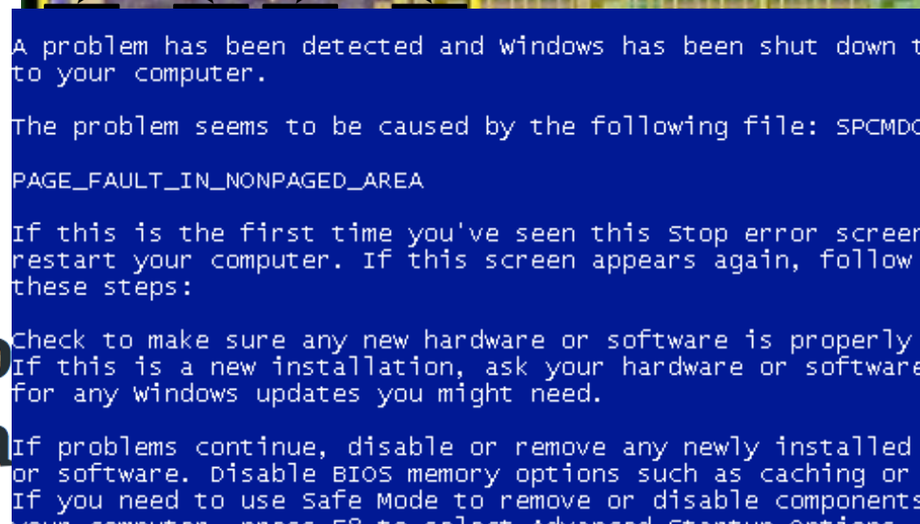
Dropbox





Intel retires “tick-tock” develo
model, extending the life of ea

The new pattern is "Process, Architecture, Optimization."



Databases


ORACLE
DATABASE
Microsoft
SQL Server
PostgreSQL
TERADATA
MariaDB
MySQL
IBM
DB2
SYBASE
H2
Access
Apache Derby
HIVE
SQLite
HyperSQL

Databases

Data — information about pretty much anything

A **database** is an ordered collection of data

Use **tables** to organize data

Databases show up everywhere!



Structured Query Language (SQL)

(Pronounced "Ess Cue El" or "Sequel", not "Squeal")

Used to manage data stored in a database

A **declarative** language — broadly speaking, tell it **what we want**, not how to do it

All "queries" (expressions) end in a semicolon ";"

Misc

Case insensitive — I capitalize keywords and operators for clarity

For example, you *could* do:

```
SeLeCT * FroM reCORDs WherE SALARY > 0;
```

...

Please don't.

SQL

The **SELECT** statement create rows

- Use the **UNION** command to join two select rows

The **CREATE TABLE** expression saves a table for later

Select

SELECT doesn't have to start from scratch

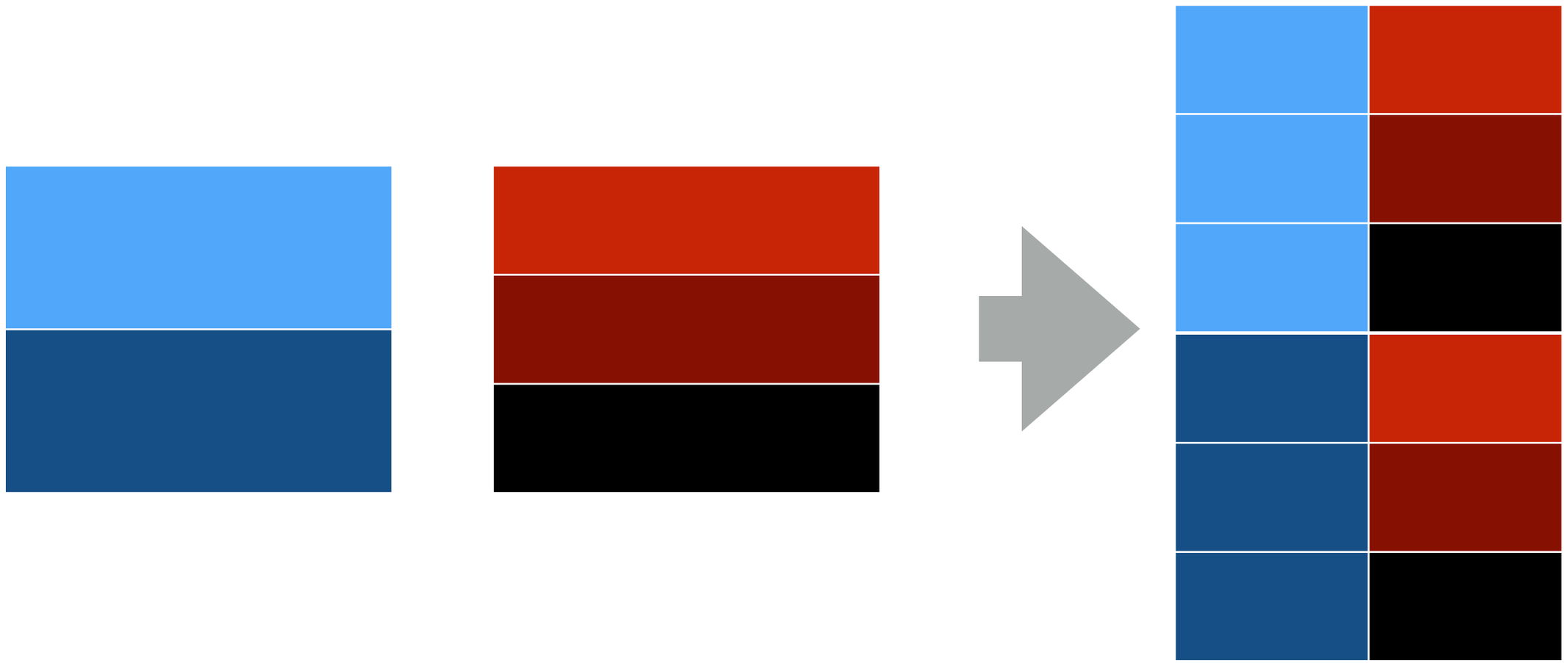
- **SELECT FROM** an existing table to create a new one

Specify what columns to keep in your result

Filter using boolean expressions in the **WHERE** clause

Joins

When we join two tables together, consider all possible pairings:

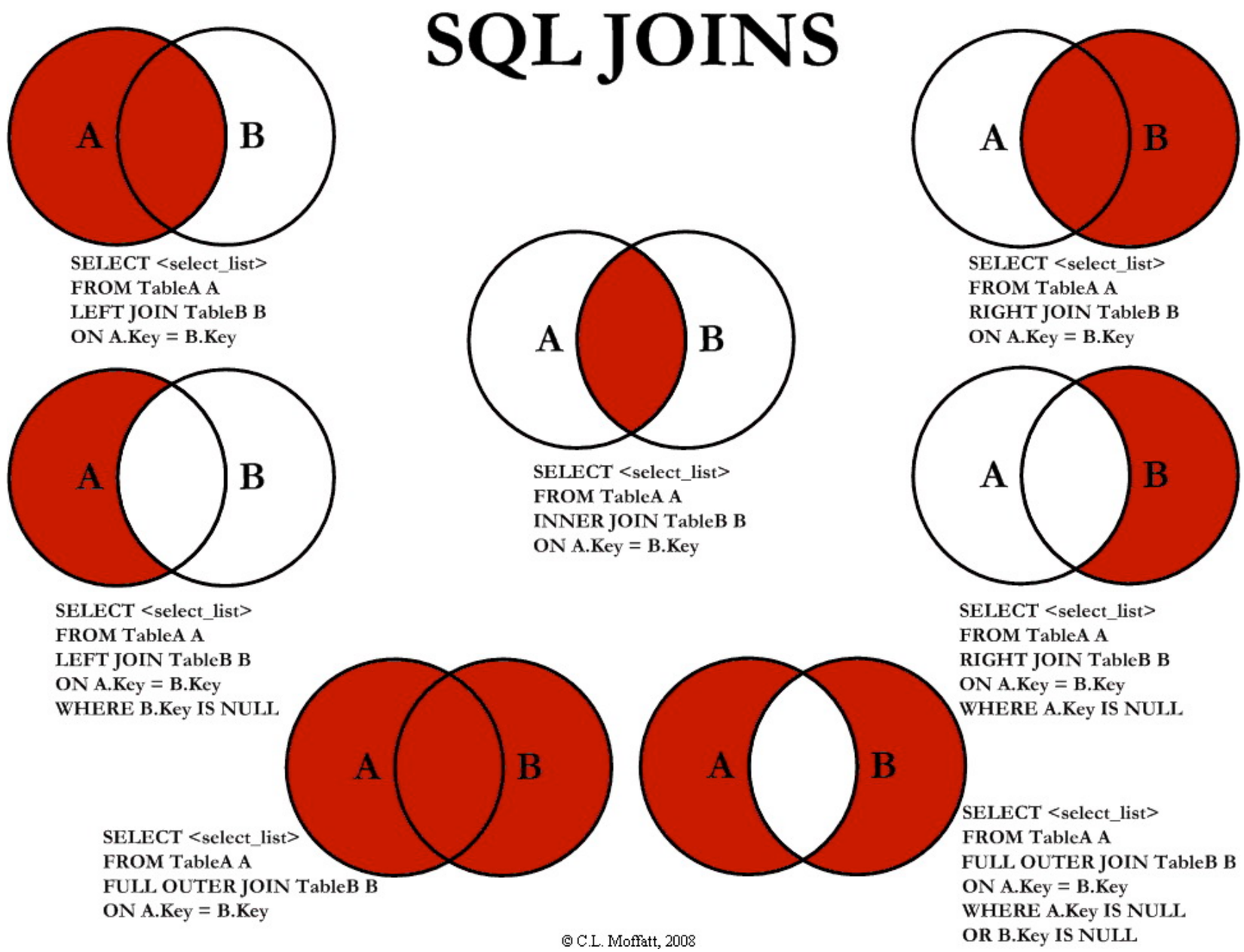


Joins

The tricky part is deciding which pairings to keep (filter in **WHERE**)

Joins

Of course, it gets more complicated (out of scope)



Aliasing

If we're joining with ourself (or a table with the same column names), we may require **aliasing**

Not sure? Then use aliases

Problem Solving

Typical problem solving approach:

1. Figure out what data you need and **join** tables that contain that data

2. Keep only the joins that make sense by filtering using **WHERE**

3. Do any extra filtering and ordering to fit the problem

4. Put the columns you want to include

Problem Solving

SELECT <cols> **FROM** <tables> **WHERE** <conds> …

(1) Where is my data coming from?

(2) What joins make sense?

(3) Any other filtering?

(4) What columns to keep?

Recursive Select

Start with a **base row** (base case)

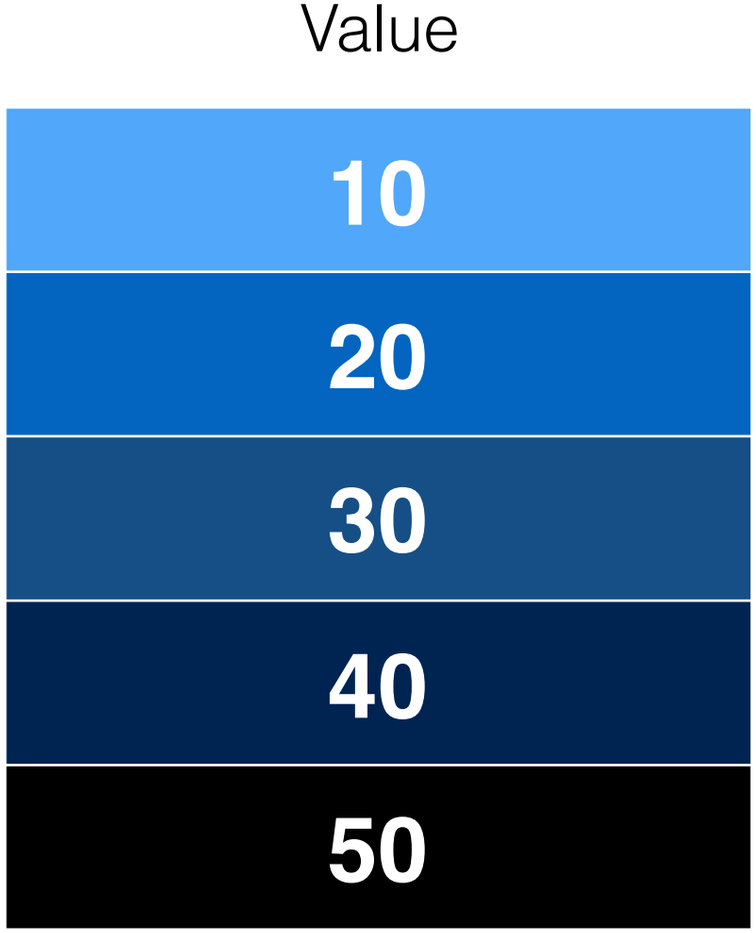
New rows based off previous ones (recursive step)

Use filter (**WHERE**) to determine when to stop

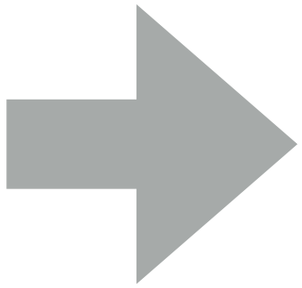
Use local **WITH** table to create recursive tables

Aggregation

Aggregation functions apply to **groupings** of rows (default ALL rows)




```
SELECT MIN(value) FROM data
```

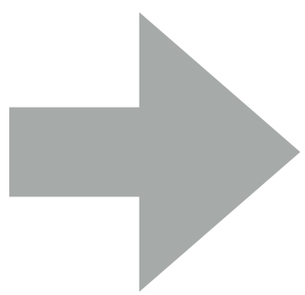


Aggregation

We can specify groups using **GROUP BY**

Value	Letter
10	A
20	B
30	A
40	B
50	A

```
SELECT MIN(value), letter
FROM data GROUP BY letter;
```



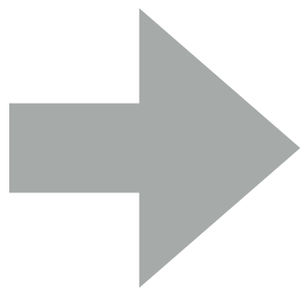
Value	Letter
10	A
20	B

Aggregation

To filter by an aggregate, use **HAVING** instead of WHERE

Value	Letter
10	A
20	B
30	A
40	B
50	A

```
SELECT MIN(value), letter
FROM data GROUP BY letter
 HAVING MIN(value) = 10;
```



Value	Letter
10	A