

COMP 421: Files & Databases

Lecture 2: It's SQL Week!

Last Class

We introduced the Relational Model as the superior data model for databases.

We then showed how the Relational Algebra provides the building blocks that will allow us to query and modify a relational database.

SQL History

In 1971, IBM created its own programming language called SQUAW.

IBM then created "SEC
System R prototype D
→ Structured English Que

IBM releases commercial DBMS
→ System/38 (1979), SQL (1982)

Q2. Find the average salary of employees in the Shoe Department.

```
AVG (EMP' SAL DEPT ('SHOE'))
```

Mappings may be *composed* by applying one mapping to the result of another, as illustrated by Q3.

Q3. Find those items sold by departments on the second floor.

ITEM SALES DEPT ° DEPT LOC FLOOR (2)

The floor '2' is first mapped to the departments located there, and then to the items which they sell. The range of the inner mapping must be compatible with the domain of the outer mapping, but they need not be identical, as illustrated by Q4.

NEWS COMPUTING

The Rise of SQL > It's become the second programming language everyone needs to know

BY RINA DIANE CABALLAR | 23 AUG 2022 | 3 MIN READ |

ISTOCK

SHARE THIS STORY



TAGS

TOP PROGRAMMING LANGUAGES

SQL

SQL dominated the jobs ranking in [IEEE Spectrum's interactive rankings of the top programming languages](#) this year. Normally, the top position is occupied by Python or other mainstays, such as C, C++, Java, and JavaScript, but the sheer number of times employers said they wanted developers with SQL skills, albeit in addition to a more general-purpose language, boosted it to No. 1.

So what's behind SQL's soar to the top? The ever-increasing use of databases, for one. SQL has become the primary query language for accessing and managing data stored in such databases—specifically [relational databases](#), which represent data in table form with rows and columns. Databases serve as the foundation of many enterprise applications and are increasingly found in other places as well, for example taking the place of traditional file systems in smartphones.

“This ubiquity means that every software developer will have to interact with databases no matter the field, and SQL is the de facto standard for interacting with databases,” says [Andy Pavlo](#), a professor specializing in database management at the [Carnegie Mellon University \(CMU\) School of Computer Science](#) and a member of the [CMU database group](#).



Jo Kristian Bergum
@jobergum

Tensor and vector da
decade. A disruption
neural representatio

Natural query langu
(SQL).

2:35 AM · Apr 27, 2021

39 Retweets 32 Qu

The
it s

rious.

r company's data and
chatbot for the data and

marks



Relational Languages

Data Manipulation Language (DML)

Data Definition Language (DDL)

Data Control Language (DCL)

Also includes:

- View definition
- Integrity & Referential Constraints
- Transactions

Important: SQL is based on **bags** (duplicates) not **sets** (no duplicates).

Today's Agenda

Aggregations + Group By
String / Date / Time Operations
Output Control + Redirection
Window Functions
Nested Queries
Lateral Joins
Common Table Expressions

From Last Lecture...

“This is DML, get some rows”

“Some column names (projection, Π)”

```
SELECT b_id-100, a_id  
FROM R WHERE a_id = 'a2';
```

“Some table or result set
to pull from”

“if this predicate is **TRUE**”
(unfortunately named select, σ)

Example Database

student(sid,name,login,gpa)

sid	name	login	age	gpa
53666	RZA	rza@cs	55	4.0
53688	Taylor	swift@cs	27	3.9
53655	Tupac	shakur@cs	25	3.5

enrolled(sid,cid,grade)

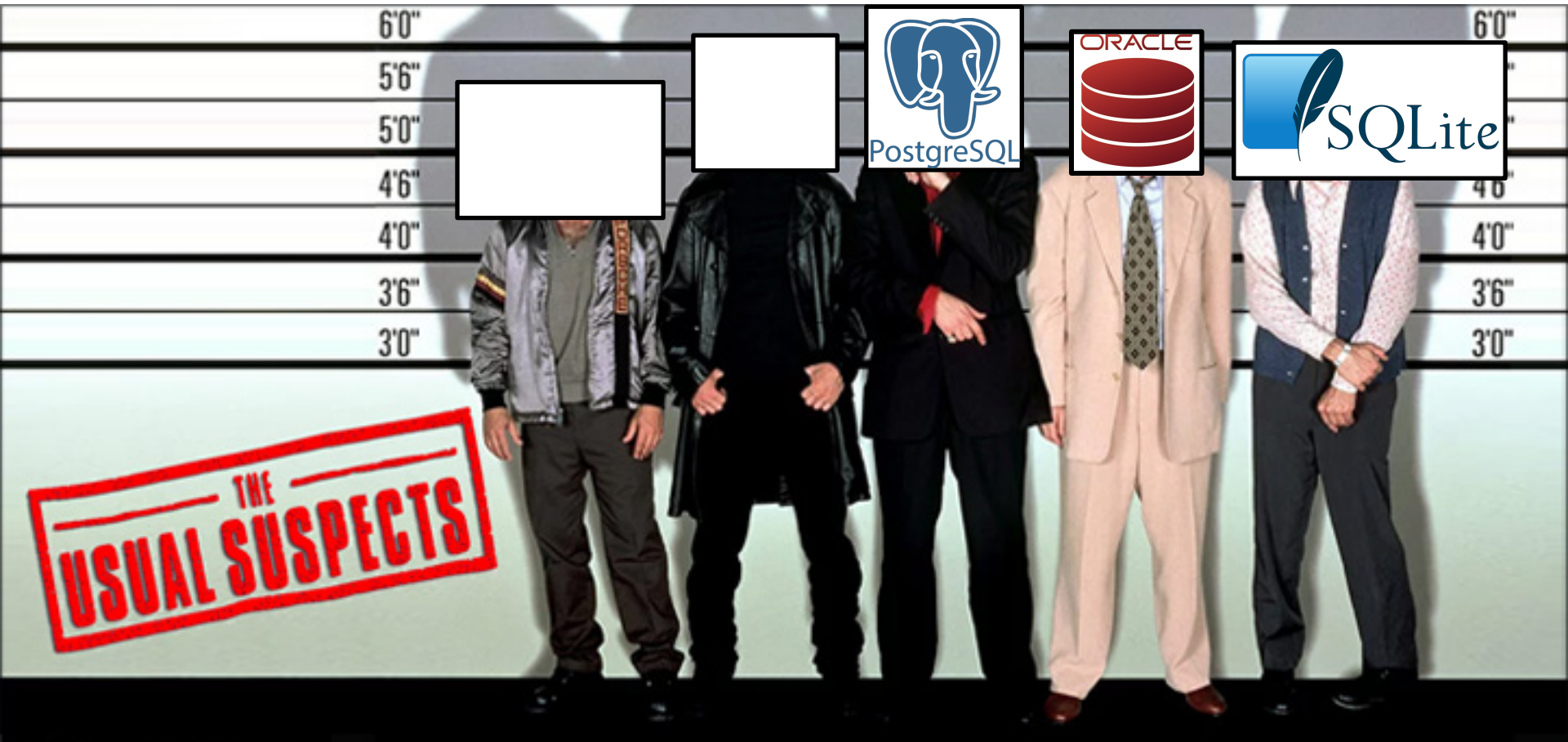
sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53655	15-445	B
53666	15-721	C

course(cid,name)

cid	name
15-445	Database Systems
15-721	Advanced Database Systems
15-826	Data Mining
15-799	Special Topics in Databases



Example Database



Aggregates

Functions that return a single value from a bag of tuples:

- **AVG(col)** → Return the average col value.
- **MIN(col)** → Return minimum col value.
- **MAX(col)** → Return maximum col value.
- **SUM(col)** → Return sum of values in col.
- **COUNT(col)** → Return # of values for col.

Aggregates

Aggregate functions can (almost) only be used in the **SELECT** output list.

Get # of students with a “@cs” login:

```
SELECT COUNT(login) AS cnt
```

```
FROM student WHERE login LIKE '%@cs'
```

```
SELECT COUNT(*) AS cnt
```

```
FROM student WHERE login LIKE '%@cs'
```

```
SELECT COUNT(1) AS cnt
```

```
FROM student WHERE login LIKE '%@cs'
```

```
SELECT COUNT(1+1+1) AS cnt
```

```
FROM student WHERE login LIKE '%@cs'
```

Multiple Aggregates

Get the number of students and their average GPA that have a “@cs” login.

```
SELECT AVG(gpa), COUNT(sid)  
FROM student WHERE login LIKE '%@cs'
```

AVG(gpa)	COUNT(sid)
3.8	3

Aggregates

Output of other columns outside of an aggregate is undefined.

Get the average GPA of students enrolled in each course.

```
SELECT AVG(s.gpa), e.cid  
FROM enrolled AS e JOIN student AS s  
ON e.sid = s.sid
```

AVG(s.gpa)	e.cid
3.86	???

```
SELECT AVG(s.gpa), ANY_VALUE(e.cid)  
FROM enrolled AS e JOIN student AS s  
ON e.sid = s.sid
```

AVG(s.gpa)	e.cid
3.86	15-445

GROUP BY

Project tuples into subsets and calculate aggregates against each subset.

```
SELECT AVG(s.gpa), e.cid
FROM enrolled AS e JOIN student AS s
ON e.sid = s.sid
GROUP BY e.cid
```

e.sid	s.sid	s.gpa	e.cid
53435	53435	2.25	15-721
53439	53439	2.70	15-721
56023	56023	2.75	15-826
59439	59439	3.90	15-826
53961	53961	3.50	15-826
58345	58345	1.89	15-445



AVG(s.gpa)	e.cid
2.46	15-721
3.39	15-826
1.89	15-445

GROUP BY

Non-aggregated values in **SELECT** output clause must appear in **GROUP BY** clause.

```
SELECT AVG(s.gpa), e.cid, s.name  
FROM enrolled AS e JOIN student AS s  
ON e.sid = s.sid  
GROUP BY e.cid, s.name
```

HAVING

Filters results based on aggregation computation.

Like a **WHERE** clause for a **GROUP BY**

```
SELECT AVG(s.gpa) AS avg_gpa, e.cid  
FROM enrolled AS e, student AS s  
WHERE e.sid = s.sid  
GROUP BY e.cid  
HAVING AVG(s.gpa) > 3.9;
```

AVG(s.gpa)	e.cid
3.75	15-415
3.950000	15-721
3.900000	15-826



avg_gpa	e.cid
3.950000	15-721

String Operations

	String Case	String Quotes
SQL-92	Sensitive	Single Only
Postgres	Sensitive	Single Only
MySQL	Insensitive	Single/Double
SQLite	Sensitive	Single/Double
MSSQL	Sensitive	Single Only
Oracle	Sensitive	Single Only

WHERE UPPER(name) = UPPER('TuPaC') **SQL-92**

WHERE name = "TuPaC"

MySQL

String Operations

LIKE is used for string matching.

String-matching operators

→ **'%'** Matches any substring (including empty strings).

→ **'_'** Match any one character

```
SELECT * FROM enrolled AS  
e
```

```
WHERE e.cid LIKE '15-%'
```

```
SELECT * FROM student AS  
s
```

```
WHERE s.login LIKE '%@c_'
```

String Operations

SQL-92 defines string functions.

→ Many DBMSs also have their own unique functions

Can be used in either output and predicates:

```
SELECT SUBSTRING(name,1,5) AS  
abbrev_name
```

```
FROM student WHERE sid = 53688
```

```
SELECT * FROM student AS s  
WHERE UPPER(s.name) LIKE 'KAN%'
```

String Operations

SQL standard defines the **||** operator for concatenating two or more strings together.

```
SELECT name FROM student  
WHERE login = LOWER(name) || '@cs'
```

SQL-92

```
SELECT name FROM student  
WHERE login = LOWER(name) + '@cs'
```

MSSQL

```
SELECT name FROM student  
WHERE login = CONCAT(LOWER(name),  
'@cs')
```

MySQL

DATE/TIME Operations

Operations to manipulate and modify **DATE/**
TIME attributes.

Can be used in both output and predicates.

Support/syntax varies wildly...

Demo: Get the # of days since the beginning of the year.

Output Redirection

Store query results in another table:

- Table must not already be defined.
- Table will have the same # of columns with the same types as the input.

```
SELECT DISTINCT cid INTO CourseIds  
FROM enrolled;
```

SQL-92

```
SELECT DISTINCT cid  
INTO TEMPORARY CourseIds  
FROM enrolled;
```

Postgres

```
CREATE TABLE CourseIds (  
  SELECT DISTINCT cid FROM enrolled);
```

MySQL

Output Redirection

Insert tuples from query into another table:

- Inner **SELECT** must generate the same columns as the target table.
- DBMSs have different options/syntax on what to do with integrity violations (e.g., invalid duplicates).

```
INSERT INTO Courselds SQL-92  
(SELECT DISTINCT cid FROM enrolled);
```

Output Control

ORDER BY <column*> [ASC|DESC]

→ Order the output tuples by the values in one or more of their columns.

SELECT sid, grade FROM enrolled

sid	grade
53123	A

WHERE sid, grade FROM enrolled
 WHERE cid = '15-721'
ORDER BY 2

SELECT sid FROM enrolled
 WHERE cid = '15-721'

ORDER BY grade DESC, sid ASC

sid
53666
53650
53123
53334



Output Control

FETCH {FIRST|NEXT} <count> ROWS OFFSET <count> ROWS

- Limit the # of tuples returned in output.
- Can set an offset to return a “range”

```
SELECT sid, name FROM student  
WHERE login LIKE '%@cs'  
FETCH FIRST 10 ROWS ONLY;
```

```
SELECT sid, name FROM student  
WHERE login LIKE '%@cs'  
ORDER BY gpa  
OFFSET 10 ROWS  
FETCH FIRST 10 ROWS WITH TIES;
```

Window Functions

Performs a calculation across a set of tuples that are related to the current tuple, without collapsing them into a single output tuple, to support running totals, ranks, and moving averages.

→ Like an aggregation but tuples are not grouped into a single output tuples.

```
SELECT FUNC-NAME(...) OVER (...)  
FROM tableName
```

*How to “slice” up data
Can also sort tuples*

*Aggregation Functions
Special Functions*

Window Functions

Aggregation functions:

→ Anything that we discussed earlier

Special window functions:

→ **ROW_NUMBER()** → # of the current row

→ **RANK()** → Order position of the current row.

sid	cid	grade	row_num
53666	15-445	C	1
53688	15-721	A	2
53688	15-826	B	3
53655	15-445	B	4
53666	15-721	C	5

```
SELECT *, ROW_NUMBER() OVER () AS  
row_num  
FROM enrolled
```

Window Functions

The **OVER** keyword specifies how to group together tuples when computing the window function.

Use **PARTITION BY** to specify group.

cid	sid	row_number
15-445	53666	1
15-445	53655	2
15-721	53688	1
15-721	53666	2
15-826	53688	1

```
SELECT cid, sid,  
       ROW_NUMBER() OVER (PARTITION BY  
cid)  
FROM enrolled  
ORDER BY cid
```



Window Functions

You can also include an **ORDER BY** in the window grouping to sort entries in each group.

```
SELECT *,  
    ROW_NUMBER() OVER (ORDER BY cid)  
FROM enrolled  
ORDER BY cid
```

Window Functions

Find the student with the second highest grade for each course.

*Group tuples by cid
Then sort by grade*

```
SELECT * FROM (  
  SELECT *, RANK() OVER (PARTITION BY  
    cid  
    ORDER BY grade ASC) AS rank  
  FROM enrolled) AS ranking  
WHERE ranking.rank = 2
```

Nested Queries

Invoke a query inside of another query to compose more complex computations.

→ Inner queries can appear (almost) anywhere in query.

Outer Query →

```
SELECT name FROM student  
WHERE  
sid IN (SELECT sid FROM enrolled)
```

← *Inner Query*

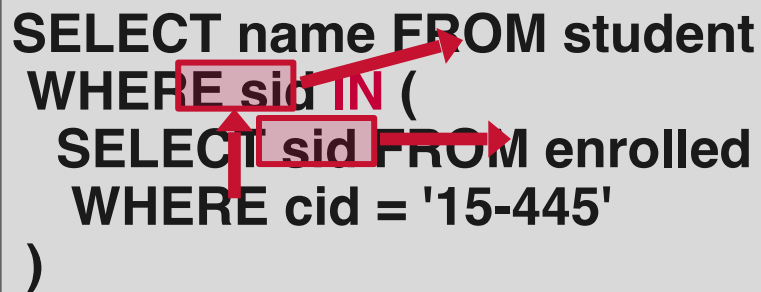
```
SELECT sid,  
       (SELECT name FROM student AS s  
        WHERE s.sid = e.sid) AS name  
FROM enrolled AS e;
```

```
SELECT * FROM student  
ORDER BY (SELECT MAX(sid) FROM  
student):
```

Nested Queries

Get the names of students in '15-445'

```
SELECT name FROM student
WHERE sid IN (
  SELECT sid FROM enrolled
  WHERE cid = '15-445'
)
```



15-445

Nested Queries

ALL → Must satisfy expression for all rows in the sub-query.

ANY → Must satisfy expression for at least one row in the sub-query.

IN → Equivalent to '**=ANY()**'.

EXISTS → At least one row is returned without comparing it to an attribute in outer query.

Nested Queries

Get the names of students in '15-445'

```
SELECT name FROM student
WHERE sid = ANY(
  SELECT sid FROM enrolled
  WHERE cid = '15-445'
)
```

Nested Queries

Find student record with the highest id that is enrolled in at least one course.

```
SELECT MAX(e.sid), s.name  
FROM enrolled AS e, student AS s  
WHERE e.sid = s.sid;
```

This won't work in SQL-92. It runs in SQLite, but not Postgres or MySQL (v8 with strict mode).

Nested Queries

Find student record with the highest id that is enrolled in at least one course.

```
SELECT sid, name FROM student
WHERE sid is the
      SELECT MAX(sid) FROM enrolled
```

sid	name
53688	Taylor

is the highest enrolled sid

Nested Queries

Find all courses that have no students enrolled in it.

```
SELECT * FROM course
WHERE NOT EXISTS(
  SELECT * FROM enrolled
  WHERE course.cid = enrolled.cid
)
```

cid	name	sid	cid	grade
15-445	Database Systems	53666	15-445	C
15-721	Advanced Database Systems	53688	15-721	A
15-826	Data Management Systems			B
15-799	Special Topics in Databases			B
15-799	Special Topics in Databases	53666	15-721	C



Lateral Joins

The **LATERAL** operator allows a nested query to reference attributes in other nested queries that precede it.

→ You can think of it like a **for** loop that allows you to invoke another query for each tuple in a table.

```
SELECT * FROM  
(SELECT 1 AS x) AS t1,  
LATERAL (SELECT t1.x+1 AS y) AS t2;
```

t1.x	t2.y
1	2

Lateral Join

Calculate the number of students enrolled in each course and the average GPA.

cid	name	cnt	avg
15-445	Database Systems	2	3.75
15-721	Advanced Database Systems	2	3.95
15-826	Data Mining	1	3.9
15-799	Special Topics in Databases	0	null

```

SELECT * FROM course AS c
  LATERAL (SELECT COUNT(*) AS cnt
           WHERE enrolled.cid = c.cid) AS t1,
  LATERAL (SELECT AVG(gpa) AS avg FROM student AS s
           JOIN enrolled AS e ON s.sid = e.sid
           WHERE e.cid = c.cid) AS t2;
  
```

Common Table Expressions

Specify a temporary result set that can then be referenced by another part of that query.

→ Think of it like a temp table just for one query.

Alternative to nested queries, views, and explicit temp tables.

```
WITH cteName AS (  
    SELECT 1  
)  
SELECT * FROM cteName
```


Common Table Expressions

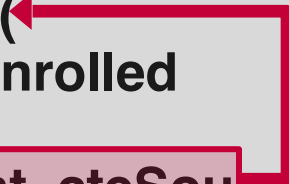
You can bind/alias output columns to names before the **AS** keyword.

```
WITH cteName (col1, col2) AS (  
    SELECT 1, 2  
)  
SELECT col1 + col2 FROM cteName
```

```
WITH cteName (colXXX, colXXX) AS (Postgres  
    SELECT 1, 2  
)  
SELECT * FROM cteName
```

Common Table Expressions

Find student record with the highest id that is enrolled in at least one course.

```
WITH cteSource (maxId) AS (  
    SELECT MAX(sid) FROM enrolled  
)  
SELECT name FROM student, cteSource  
WHERE student.sid = cteSource.maxId
```

Conclusion

SQL is a hot language.

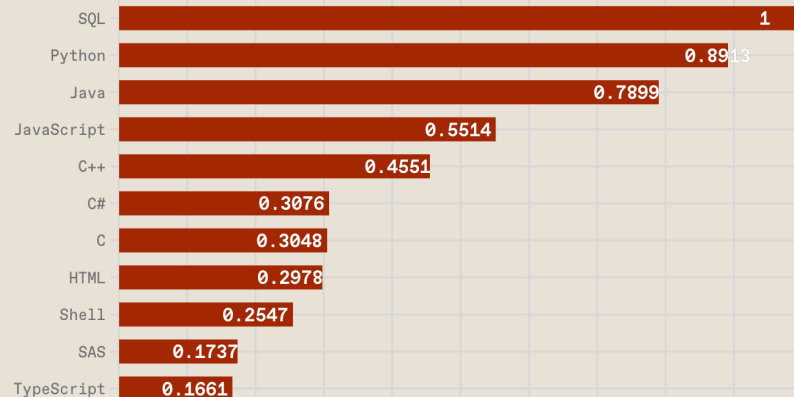
→ Lots of NL2SQL tools, but writing SQL is not going away.

You should (almost) always strive to compute your answer as a single SQL statement.

Top Programming Languages 2023

Click a button to see a differently weighted ranking

Spectrum **Jobs** Trending



Now, DIY

Write SQL queries to perform basic data analysis.

- Subset of IMDb data
- Open devcontainer for Bootcamp 1, go to **./sql/**
- To check: **./sql/check.py PATH_TO_FILE**
- Where the files are like **./sql/q1_sample.sql**

Next Class

We go from history / applications to present day systems. Starting with storage.