

# Lec 2 Modern SQL

## # History of SQL

- 1971 SQUARE by IBM
- 1972 SEQUEL for System R
- 1986 ANSI Standard, ISO 1987
- current SQL:2023

→ Minimum support for SQL-92 is common for DB systems

## # Relational Langs

SQL builds upon bags, not sets  
 (allows duplicates, orders could be random)



## # Aggregation, Grouping

```

stu (sid, name, login, age, gpa)
course (cid, name)
enrolled (sid, cid, ...)
  
```

- notation: id means col id is primary key
- DuckDB, SQLite as in-process DBs
- Mem layout can be row / col major
- PostgreSQL, own-process as server, old, gold-standard

```

DuckDB .tables, .schema
select * from artist limit 10;
select COUNT(*) from artist;
select COUNT(*) from artist group by nationality;
    ↑ for each unique nationality count.. but only returns row count
select COUNT(*), nationality from artist group by nationality;
    ↑ can access because group by creates unique record for each nationality
select COUNT(*), nationality from artist
group by nationality
order by nationality desc;
select COUNT(*) as countOfArtists, nationality from artist
group by nationality
order by countOfArtists desc;
select COUNT(*) as countOfArtists, nationality from artist
group by nationality
order by countOfArtists desc, nationality asc;
    ↑ if same count sort alphabetically
select count(distinct login) from student
where login like '%@cs'
    ↑ @ turns bag into set
    ↓ @ gets a bag
  
```

⚠ Non-aggregate values in SELECT output must be in GROUP BY

```

... GROUP BY e.cid, s.name;
    ↓ possible groups are e.cid x s.name
  
```

⚠ filter after GROUP BY — use HAVING  
 WHERE applies to data from FROM

```

... GROUP BY e.cid
HAVING AVG(s.gpa) > 3.9;
  
```

## # String, Date, Time

```

SQL-92 where upper(name) = upper('TuPaC')
MySQL where name = "TuPaC"
    ↑ co is quotation standard
  
```

String matching % — any s ∈ Σ\*  
 - — any c ∈ Σ

String functions - concat...  
 ||  
 +  
 CONCAT()

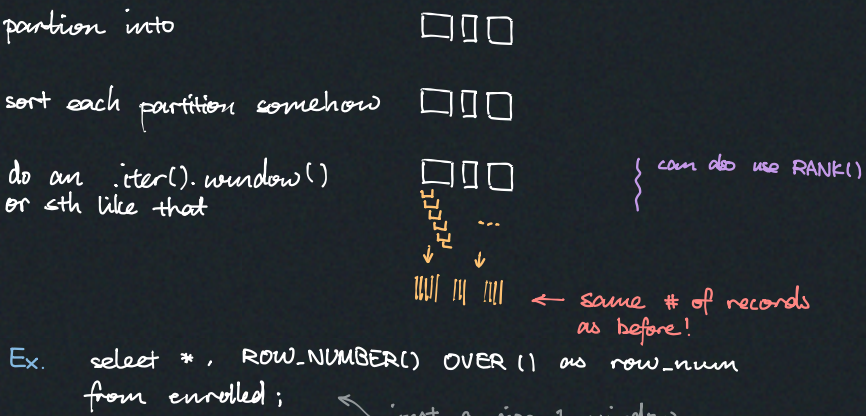
Time  
 ↑ limited standard  
 - SELECT CURRENT\_TIMESTAMP usually portable  
 - time different...  
 SQLite first convert to julianday  
 Postgre use EXTRACT

## # Output Control, Redirection

```

select distinct cid into CourseIds from enrolled; ← SQL-92
create table CourseIds (select distinct cid from enrolled);
select distinct cid into temporary CourseIds from enrolled;
... fetch first 10 rows only;
... offset 10 rows fetch first 10 rows with ties;
  
```

## # Window functions



```

Ex. select *, ROW_NUMBER() OVER () as row_num from enrolled;
    ← just a size 1 window, no partition
select cid, sid, row-number() over (partition by cid) from enrolled order by cid;
    ← numbers each partition independently
select * from (select *, RANK() over (partition by cid order by grades asc) as rank from enrolled) as ranking where ranking.rank = 2;
    ← nesting!
  
```

## # Nesting

```

select name from student where sid in (select sid in enrolled);
  
```

```

select name from student where sid in (select sid in enrolled where cid = '15-445');
  
```

- Some nested queries:
- ALL
  - ANY
  - IN
  - EXISTS
  - NOT EXISTS

```

select * from course where not exists (select * from enrolled where course.cid = enrolled.cid);
    ← courses with no enrollment
    depends on outside, so row for each row in outer block
  
```

## # Lateral Join

Similar: inner correlated to outer

```

select * from (select (as x) as t1, lateral (select t1.x + 1 as y as t2);
  
```

```

select * from course as c, lateral (select count(*) as cnt from enrolled where enrolled.cid = c.cid) as t1 lateral (select avg(gpa) as avg from students as s join enrolled as e on s.sid = e.sid where e.cid = c.cid) as t2;
  
```

## # Common Table Expressions

Like defining & using temporary table

```

with cteSource (maxId) as (select max(sid) from enrolled)
select name from student, cteSource where student.sid = cteSource.maxId
  
```