CIS 451/551

week 4: More SQL
GROUP BY

• for aggregate functions: COUNT, SUM, AVG, etc
• format:
  SELECT <attribute list, including aggregate functions>
  FROM <table list>
  WHERE <logical condition>
  GROUP BY <attribute list>
  HAVING <logical condition>

• aggregate functions apply to groups of data
• the groups are determined by all rows with same GROUP BY values
• HAVING is optional, like a WHERE clause but applies after the aggregation
• order of “execution”
  FROM → WHERE → GROUP BY → <compute aggregates> → HAVING → SELECT
example 1.1

```
SELECT customer_num, fname, lname, COUNT(*)
FROM customer JOIN orders USING(customer_num)
GROUP BY customer_num

counts the number of orders made by each customer (only if that customer has an order)
```
example 1.2

SELECT customer_num, fname, lname, COUNT(*), SUM(ship_charge), AVG(ship_weight)
FROM customer JOIN orders USING(customer_num)
GROUP BY customer_num
	here are many different aggregate functions – take some time to look them up
example 1.3

SELECT customer_num, fname, lname,
COUNT(order_num)
FROM customer LEFT JOIN orders
USING(customer_num)
GROUP BY customer_num

same as ex 1.1, except this counts customers
with zero orders – note that
• need an outer join to include those
  customers
• and need to count a field value
• count(*) counts the number of rows, even if
  some values are null
HAVING clause

• the HAVING clause applies a logical condition, just like the WHERE clause
• the WHERE clause is applied *before* the aggregate functions are computed

• so the HAVING clause is needed for the values of the aggregate functions
example 1.4

```
SELECT customer_num, fname, lname, state,
       COUNT(order_num) AS onum
FROM customer JOIN orders USING(customer_num)
WHERE state='CA'
GROUP BY customer_num
HAVING onum>=2
```

counts the customers who live in CA and who have made at least two orders
subqueries

• in some cases, a SELECT query can be used inside another query
• can be in either the FROM or WHERE clause
  • behave slightly differently
  • probably could put one on the SELECT line as well

• obviously adds complexity
• come in correlated and uncorrelated forms
  • correlated: fields in the sub-query depend on the main part of the (outer) query
  • uncorrelated more efficient
example 2.1

```
SELECT fname, lname, salary
FROM employee
WHERE salary = (SELECT MAX(salary) FROM employee)
```

- finds the employee with largest salary
- can say = when only one value returned
example 2.2

SELECT fname, lname, salary
FROM employee
WHERE salary >= ALL (SELECT salary FROM employee)

• does the same as previous query
• also ANY, SOME
example 2.3

SELECT fname, lname, salary, dno
FROM employee eout
WHERE salary = (SELECT MAX(salary) FROM employee ein
               WHERE eout.dno=ein.dno)

• finds the person with the maximum salary in their department
• this is a correlated subquery
• less efficient since it has to be run for each row of the outer table
example 2.4

```
SELECT e.ssn, e.fname, e.lname
FROM employee e
WHERE EXISTS (SELECT *
               FROM employee ein
               WHERE ein.superssn=e.ssn)
```
example 2.5

SELECT e.ssn, e.fname, e.lname
FROM employee e
WHERE e.ssn IN (SELECT superssn FROM employee)

- also finds supervisors
- note use of IN
- uncorrelated subquery
example 2.6

```
SELECT e.ssn, e.fname, e.lname
FROM employee e
WHERE e.ssn NOT IN (SELECT superssn FROM employee)
```

- should find non-supervisors
- what goes wrong?
example 2.7

SELECT ssn, fname, lname
FROM employee
WHERE ssn NOT IN (SELECT superssn
                   FROM employee
                   WHERE superssn IS NOT NULL)

NULL fields have special status
watch out for NULL values

• can mean ‘unknown’ or ‘missing’
• x=NULL returns ‘unknown’
• ... so implicitly SQL uses three-valued logic
• “use of NULL considered harmful”, C. J. Date
example 3.1

```sql
SELECT CONCAT(c.fname, ' ', c.lname),
    IFNULL(CONCAT(DAYNAME(@od:=o.order_date), ', ', MONTHNAME(@od), ' ', DAY(@od), ', ', YEAR(@od)), 'No Order') AS orderDate
FROM customer c LEFT JOIN orders o USING(customer_num)
```

- variables within SELECT statement prefixed by @
- note IFNULL operator
example 3.2

```sql
SELECT fname, lname, salary,
    @prev := @curr,
    @curr := salary,
    @rank := IF(@prev = @curr, @rank, @rank+1) AS rank
FROM employee,
    (SELECT @curr:= null, @prev := null, @rank := 0) AS whatever
ORDER BY salary DESC;
```

example 3.3

SELECT manu_code, manu_name
FROM manufact main
WHERE NOT EXISTS
  (SELECT stock_num
   FROM stock s, manufact m
   WHERE s.manu_code=m.manu_code AND m.manu_name='Norge'
   AND s.stock_num NOT IN
     (SELECT stock_num
      FROM stock
      WHERE manu_code=main.manu_code))

• find manufacturers who make everything Norge makes
• double nested subqueries
example 3.4

```
SELECT dno, COUNT(*)
FROM employee e JOIN
department ON dno=dnumber
WHERE salary>=38000
GROUP BY dno
HAVING COUNT(*)>=2
```

- count the number of employees who earn over 38000 in departments with at least two employees
- this is incorrect
example 3.5

SELECT dno, COUNT(*)
FROM employee e JOIN
(SELECT dno FROM employee GROUP BY dno HAVING COUNT(*)>=2)
  ldept
  ON dno=dnumber
WHERE salary>=38000
GROUP BY dno

**correct version of previous**
example 3.6 (UNION)

```sql
SELECT 1 as madeorder, customer_num
FROM customer
WHERE customer_num IN
    (SELECT customer_num FROM orders)
UNION
SELECT 0 as madeorder, customer_num
FROM customer
WHERE customer_num NOT IN
    (SELECT customer_num FROM orders)
ORDER BY madeorder DESC, customer_num
```

find customers who have made an order, followed by those who have not
example 3.7

```
SELECT 1 as madeorder, customer_num
FROM customer c
WHERE EXISTS
  (SELECT *
   FROM orders o
   WHERE o.customer_num=c.customer_num
  )
UNION
SELECT 0 as madeorder, customer_num
FROM customer c
WHERE NOT EXISTS
  (SELECT *
   FROM orders o
   WHERE o.customer_num=c.customer_num
  )
ORDER BY madeorder DESC, customer_num
```
note on UNION

- UNION removes duplicates
- UNION ALL keeps duplicates
other SQL

- *data manipulation language* (DML)
  - SELECT
  - INSERT
  - DELETE
  - UPDATE

- *data definition language* (DDL)
  - CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, etc.