Usual GROUP BY issue

The following query runs in MySQL but not Access, Informix:

```
SELECT c.accountId, lastName, firstName, count(*)
FROM Customer c INNER JOIN Rental r
   ON c.accountId=r.accountId
GROUP BY lastName, firstName;
```

Why might there be a problem?

Some Hard Problems

+ Find Customers who are currently renting a movie they haven’t rented before. Include accountId, firstName, lastName, and the number of those videos each of those customers is currently renting.

+ List all videos that are being rented for the first time.

+ List all movies that are being rented for the first time.

+ Which customer (account) has (previously) rented the most movies?

+ Do the previous query but count previous and current rentals.

+ For each customer (name), give the number of accounts they have and the total they have spent on their (previous) rentals.

Some Solutions

```
SELECT c.accountId, c.firstName, c.lastName, count(r.videoId) AS 'Video Count'
FROM ((Customer c INNER JOIN Rental r
   ON c.accountId=r.accountId) INNER JOIN video v
   ON r.videoId=v.videoId) INNER JOIN movie m
   ON v.movieId=m.movieId
WHERE NOT EXISTS
   (SELECT p.accountId
    FROM PreviousRental p INNER JOIN Video v2
    ON p.videoId=v2.videoId
    WHERE p.accountId=c.accountId
    AND m.movieId=v2.movieId)
GROUP BY c.accountId, c.firstName, c.lastName;
```
SELECT v.videoid, m.title
FROM (movie m JOIN video v USING(movieid))
JOIN rental r USING(videoid)
WHERE v.videoid NOT IN
  (SELECT videoid
   FROM video JOIN previousrental USING(videoid)
  )

SELECT m.title
FROM (movie m JOIN video v USING(movieid))
JOIN rental r USING(videoid)
WHERE m.movieid NOT IN
  (SELECT movieid
   FROM (movie JOIN video USING(movieid))
   JOIN previousrental USING(videoid)
  )

SELECT c.accountid, c.firstname, c.lastname, count(p.videoid)
FROM customer c JOIN previousrental p USING(accountid)
GROUP BY c.accountid
HAVING count(p.videoid) >= ALL
  (SELECT count(videoid)
   FROM customer JOIN previousrental USING(accountid)
   GROUP BY accountid)

SELECT c.accountid, c.firstname, c.lastname, count(a.videoid)
FROM customer c JOIN
  (SELECT r.accountid, r.videoid FROM rental r
   UNION ALL
   SELECT p.accountid, p.videoid FROM previousrental p) AS a
USING(accountid)
GROUP BY c.accountid
HAVING count(a.videoid) >= ALL
  (SELECT count(a1.videoid)
   FROM customer c1 JOIN
   (SELECT r1.accountid, r1.videoid FROM rental r1
    UNION ALL
    SELECT p1.accountid, p1.videoid FROM previousrental p1) AS a1
   USING(accountid)
  GROUP BY c1.accountid)

SELECT c.firstname, c.lastname, COUNT(accountid), IFNULL(SUM(t.ac),0) as totspent
FROM customer c LEFT JOIN
  (SELECT accountid, SUM(cost) ac FROM previousrental GROUP BY 1) AS t
USING(accountid)
GROUP BY 1,2
Transactions

Goal is to enforce ACID properties for concurrently running transactions accessing the same database.

Atomicity
Each transaction is done completely or not at all.

Consistency
A transaction is assumed to leave the database in a consistent state.

Isolation
Concurrent transactions should not interfere with one another.

Durability
When a transaction finishes (commits), its actions are permanent.

Handled by locking and logging. Locks ensure isolation, while logging allows for rollbacks and crash recovery.

SQL transaction statements

BEGIN
START TRANSACTION -- same as BEGIN
COMMIT
ROLLBACK