Database Management Systems - I, CS 157A

SQL Group-by, Sub-query Clauses and Security

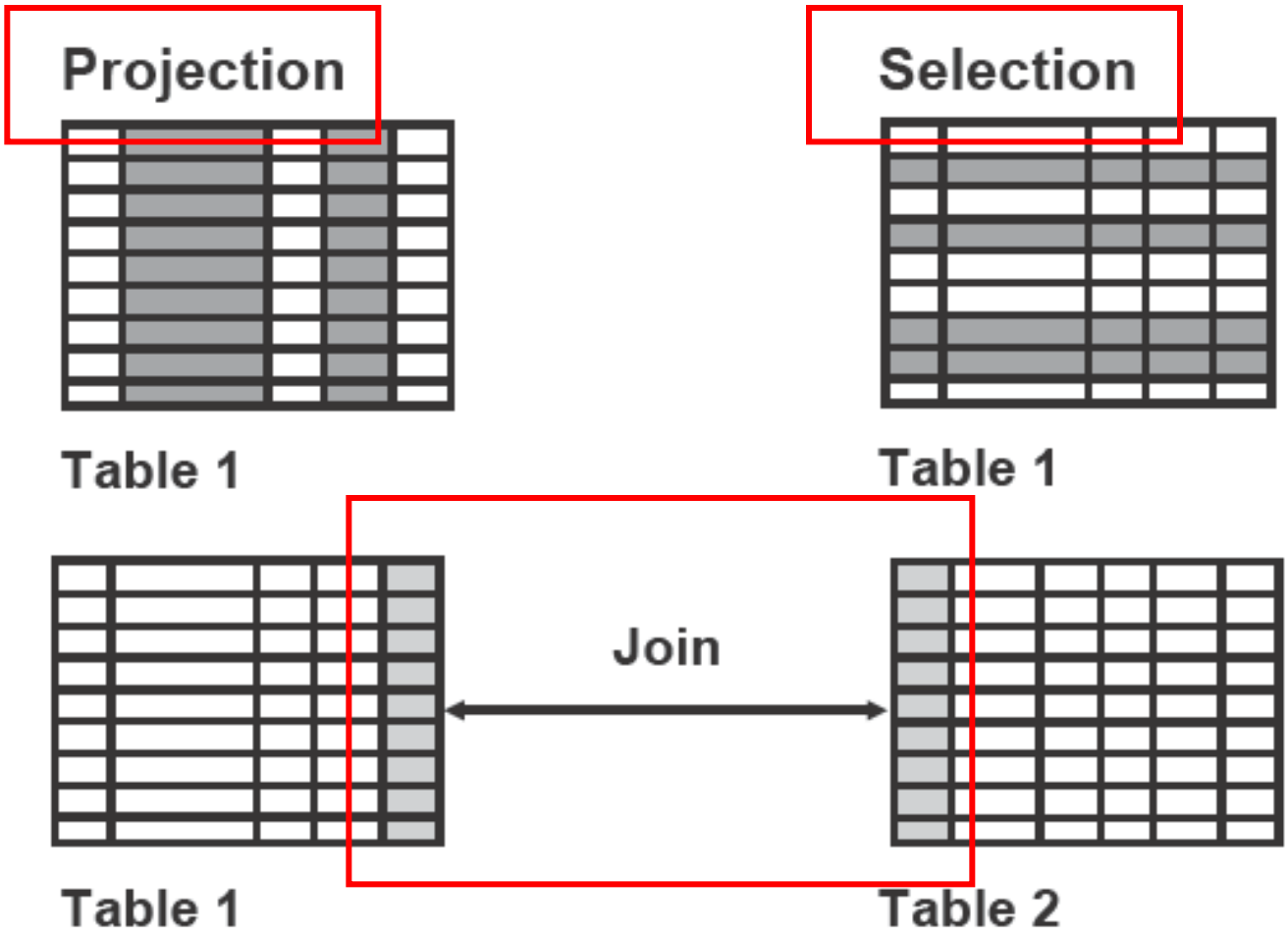
Agenda

- Functions
  - Group functions
- Outer Join
- Sub-queries
- Security

SQL Statements

DML (Data Manipulation Language)	SELECT INSERT UPDATE DELETE
DDL (Data Definition Language)	CREATE ALTER DROP
DCL and Transaction Control	GRANT REVOKE COMMIT ROLLBACK

REVIEW: SQL SELECT



BETWEEN Operator

- Use the BETWEEN operator to display rows based on a range of values
- **SELECT** last_name, salary
  FROM employees
  WHERE salary **BETWEEN** 2500 **AND** 3500 ;

Membership Condition Using IN

- Use the **IN** operator to test for values in a list
- **SELECT** last_name, salary, manager_id
  FROM employees
  WHERE manager_id **IN** (100, 101, 201);

Using NULL Conditions

- Test for nulls with **IS NULL** operator
-
```
SELECT     last_name, manager_id
FROM       employees
WHERE      manager_id  IS  NULL ;
```

- Note: you cannot test with = (you need to use **IS** instead**)**
  - A null is not equal, or unequal to any value

ORDER-BY Clause

- Sort retrieved rows with ORDER BY clause
  - **ASC**: Ascending order, default
  - **DESC**: Descending order
- The ORDER BY clause comes last in the SELECT statement

```
SELECT      last_name, department_id, hire_date
FROM        employees
ORDER BY    hire_date ;
```

Sorting

- Sorting in descending order

```
SELECT      last_name, department_id, hire_date
FROM        employees
ORDER BY    hire_date DESC ;
```

- Sorting by column alias

```
SELECT      last_name, salary*12   annsal
FROM        employees
ORDER BY    annsal;
```

Sorting (cont.)

- Sorting using column's numeric position

```
SELECT      last_name, job_id, hire_date, salary
FROM        employees
ORDER BY    3;
```

- Sorting by multiple columns

```
SELECT      last_name, job_id, salary
FROM        employees
ORDER BY    job_id,  salary  DESC ;
```

Functions

Case-Conversion Functions

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

Example: Case-Conversion

```
SELECT      last_name, job_id, salary
FROM        employees
WHERE       last_name = 'peng';
```

0 rows returned.

```
SELECT      last_name, job_id, salary
FROM        employees
WHERE       LOWER(last_name) = 'peng';
```

1 rows returned.

Character Manipulation Functions

Function	Result
SUBSTR('HelloWorld', 1 ,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W') – Search for 'W' in the 'HelloWorld' string	6
LPAD(salary, 10, '*') – left Pad with '*' up to 10 characters	*****24000
RPAD(salary, 10, '*') – right Pad	24000*****

Number Functions

Function	Result
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
Remainder = MOD(1600, 300)	100

Group Functions

Function	Description
AVG	Average
COUNT	Number of rows
SUM	Sum values
MAX/MIN	Maximum / Minimum value

GROUP Functions and Null Value

- Group functions ignore null values in the column

```
SELECT    AVG(commision_pct)
FROM      employees;
```

Ignore commision_pct column when value is NULL.

Creating Groups of Data

- You can divide rows in a table into smaller groups (chunks) using the GROUP BY clause

```
SELECT      column, group_function(column)
FROM        table
[WHREE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

Example: GROUP BY

- All columns in the SELECT clause that are not in group functions must be in the GROUP BY clause

```
SELECT      department_id, AVG(salary)
FROM        employees
GROUP BY    department_id;
```

Illegal Queries with Group Functions

```
SELECT      department_id, COUNT(name)
FROM        employees;
```

A GROUP_BY clause must be added to count the name for each dept!!

```
SELECT      department_id, job_id, COUNT(name)
FROM        employees
GROUP BY    department_id;
```

Either remove job_id, or

Add job_id in the GROUP_BY

Illegal Queries with Group Functions

- You cannot use the WHERE clause to restrict groups

```
SELECT      department_id,  AVG(salary)
FROM        employees
WHERE       AVG(salary) > 8000
GROUP BY    department_id;
```

- Use the **HAVING** clause to restrict groups

```
SELECT      department_id,  AVG(salary)
FROM        employees
GROUP BY    department_id
HAVING      AVG(salary) > 8000 ;
```

Restricting Group Results with the HAVING clause

- When you use the HAVING clause, Oracle server restricts groups as follows
  1. Rows are grouped
  2. The group function is applied
  3. Groups matching the HAVING clause are displayed

```
SELECT      department_id,  AVG(salary)
FROM        employees
GROUP BY    department_id
HAVING      AVG(salary) > 8000 ;
```

Subquery

Subquery (Nested SELECT)

```
SELECT    select_list
FROM      table
WHERE     expr  operator
                (SELECT    select_list
                 FROM      table);
```

- The subquery (inner query) executes before the main query (outer query)
- The result of the subquery is used by the main query

Using Group Functions in a Subquery

```
SELECT   last_name, job_id, salary
FROM     employees
WHERE    salary =
         (SELECT   MIN(salary)
          FROM     employees);
```

- Select employee with minimum salary
- Note the subquery returns a single value, say 2500, to the outer query.

What's Wrong with this Statement?

```
SELECT      last_name, salary
FROM        employees
WHERE       salary  =
            (SELECT     MIN(salary)
             FROM       employees
             GROUP BY   department_id) ;
```

- The subquery returns multiple values, one for each department. The = operator is a single-row comparison operator that expects only one value.

Multi-Row Subqueries

```
SELECT    last_name, salary
FROM      employees
WHERE     salary   IN
          (SELECT    MIN(salary)
          FROM       employees
          GROUP BY   department_id);
```

- The subquery returns multiple values, one for each group. We use IN operator here, which is a multi-row operator that expects one or more values.

Security and User Authorization in SQL

Authorization

- A file system identifies certain privileges on the objects (files) it manages:
  - Typically: <read, write, execute>

- A file system identifies certain participants to whom privileges may be granted.
  - Typically: <owner, a group, all users>

Privileges – (1)

- SQL identifies a more detailed set of privileges on objects (relations) than the typical file system

- **Nine privileges in all**, some of which can be restricted to one column of one relation

Privileges – (2)

- **Some important privileges on a relation:**
  1. SELECT = right to query the relation
  2. INSERT = right to insert tuples
     - May apply to only one attribute
  3. UPDATE = right to update tuples
     - May apply to only one attribute
  4. DELETE = right to delete tuples

Example: Privileges

- For the statement below:

```
INSERT INTO Beers(name)
    SELECT beer FROM Sells
    WHERE NOT EXISTS
        (SELECT * FROM Beers
         WHERE name = beer);
```

beers that do not appear in Beers. We add them to Beers with a NULL manufacturer.

- We require privileges SELECT on Sells and Beers, and INSERT on Beers or Beers.name

Database Objects

- The objects on which privileges exist include stored tables and views
- Other privileges are the right to create objects of a type, e.g., triggers
- Views form an important tool for access control

Example: Views as Access Control

- We might not want to give the SELECT privilege on Emps(name, addr, salary)
- But it is safer to give SELECT on:

```
CREATE VIEW SafeEmps AS
    SELECT name, addr FROM Emps;
```

- Queries on SafeEmps do not require SELECT privilege on Emps, just on SafeEmps

Authorization ID's

- A user is referred to by *authorization ID*, typically their login name
- There is an **authorization ID** PUBLIC:
  - Granting a privilege to PUBLIC makes it available to any authorization ID

Granting Privileges

- You have all possible privileges on the objects, such as relations, that you create
- You may grant privileges to other users (authorization ID's), including PUBLIC
- You may also grant privileges WITH GRANT OPTION, which lets the grantee also grant this privilege

The GRANT Statement

- To grant privileges, say:

  GRANT <list of privileges>

  ON <relation or other object>

  TO <list of authorization ID's>;

- If you want the recipient(s) to be able to pass the privilege(s) to others add:

  WITH GRANT OPTION

Example: GRANT

- Suppose you are the owner of Sells. You may say:

```
GRANT SELECT, UPDATE(price)
ON    Sells
TO    sally;
```

- Now Sally has the right to issue any query on Sells and can update the price component/attribute only

Example: Grant Option

- Suppose we also grant:

```
GRANT UPDATE ON Sells TO sally
WITH GRANT OPTION;
```

- Now, Sally not only can update any attribute of Sells, but can grant to others the privilege UPDATE ON Sells:
  - Also, she can grant more specific (restricted) privileges like `UPDATE(price) ON Sells`

Revoking Privileges

REVOKE <list of privileges>
ON <relation or other object>
FROM <list of authorization ID's>;

- Your grant of these privileges can no longer be used by these users to justify their use of the privilege:
  - But they may still have the privilege because they obtained it independently from elsewhere

REVOKE Options

- We must append to the REVOKE statement either:
  1. **CASCADE:** Now, any grants made by a revokee are also not in force, no matter how far the privilege was passed
  2. **RESTRICT:** If the privilege has been passed to others, the REVOKE fails as a warning that something else must be done to “chase the privilege down”

Grant Diagrams

- Nodes = user/privilege/grant option? / is owner?
  - UPDATE ON R, UPDATE(a) on R, and UPDATE(b) ON R live in different nodes
  - SELECT ON R and SELECT ON R WITH GRANT OPTION live in different nodes
- Edge *X* → *Y* means that node *X* was used to grant *Y*

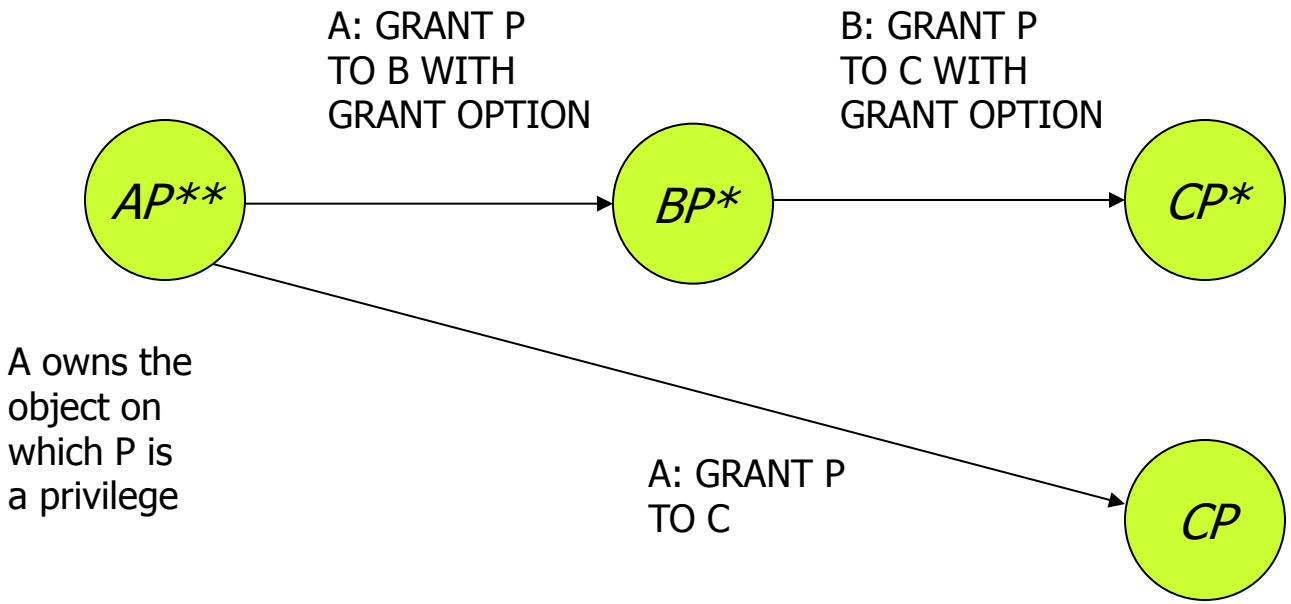
Notation for Nodes

- Use *AP* for the node representing authorization ID *A* having privilege *P*:
  - *P* * = privilege *P* with grant option
  - *P* ** = the source of the privilege *P*
    - I.e., *A* is the owner of the object on which *P* is a privilege
    - Note ** implies grant option

Manipulating Edges – (1)

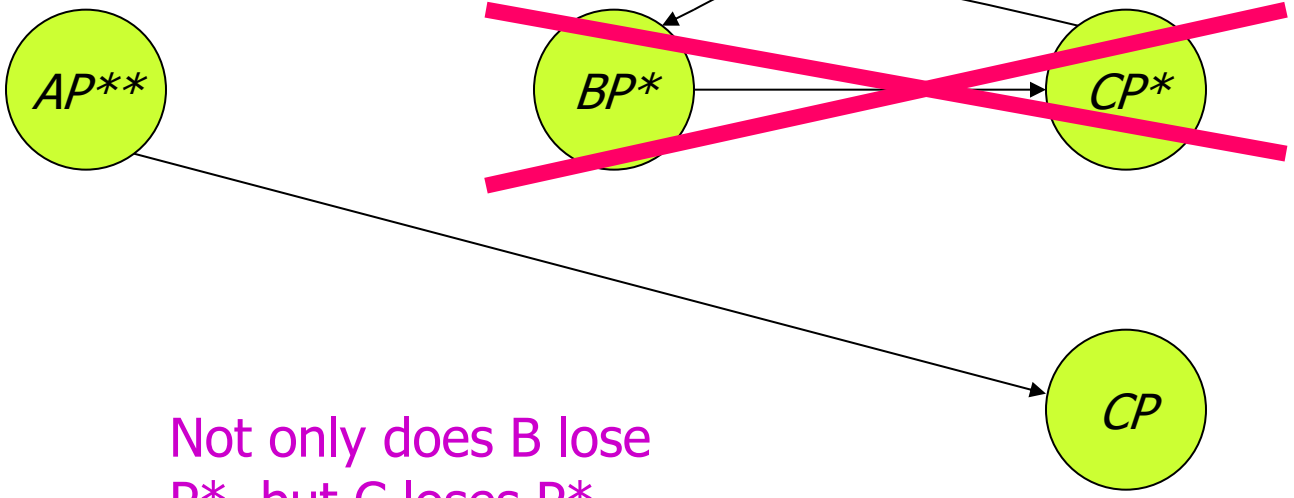
- When *A* grants *P* to *B*, We draw an edge from *AP* * (A is not owner) or *AP* ** (if A is owner) to *BP*
  - Or to *BP* * if the grant is with grant option
- If *A* grants a subprivilege *Q* of *P* [say UPDATE(a) on R when *P* is UPDATE ON R] then the edge goes to *BQ* or *BQ* *, instead

Example: Grant Diagram



Example: Grant Diagram

A executes
REVOKE P FROM B CASCADE;



Even had C passed P to B, both nodes are still cut off

Not only does B lose P*, but C loses P*. Delete BP* and CP*

However, C still has P without grant option because of the direct grant.

Summary

- Functions
  - String function
  - Numeric functions
  - Group functions
- Outer Join
- Sub-query
- Security

END