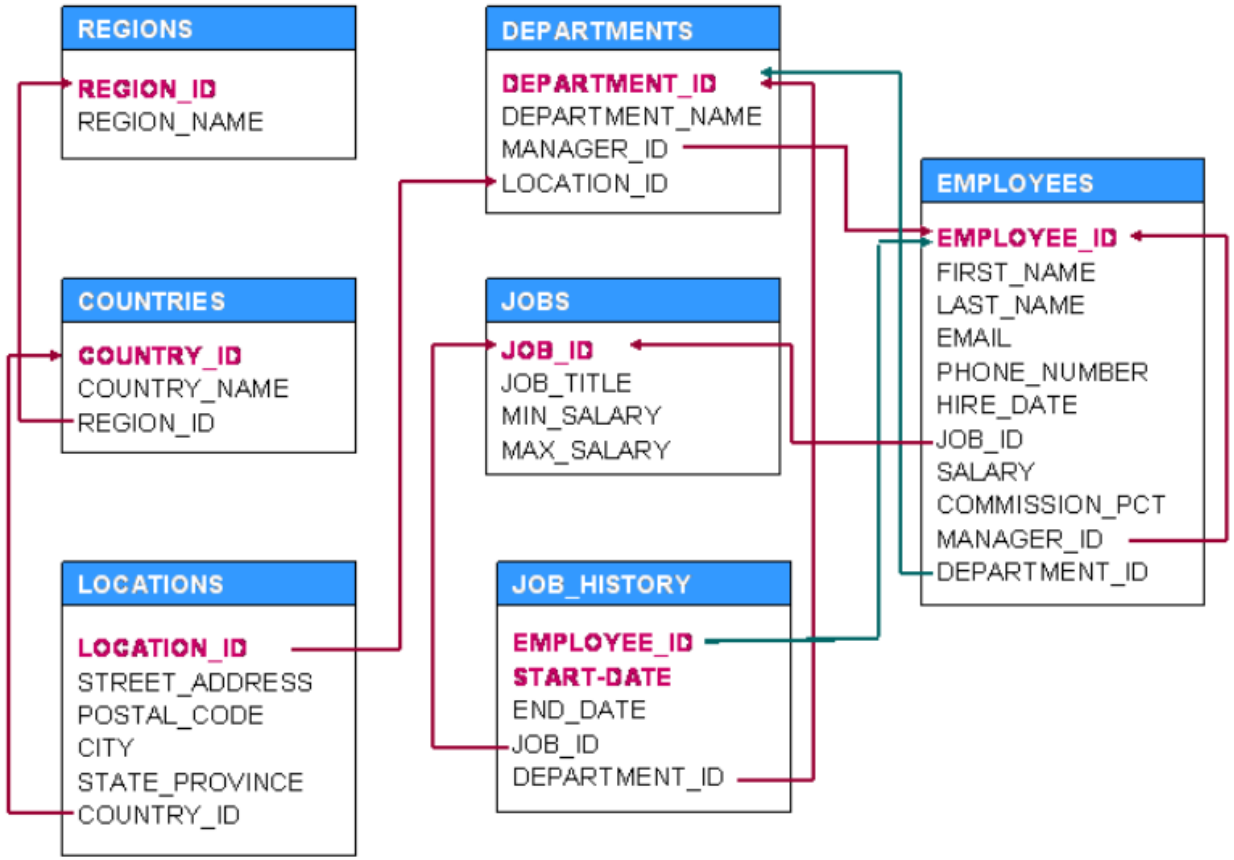
SQL ...

CS 341 Database Systems Lab

HR Schema


REGIONS
REGION_ID
REGION_NAME
DEPARTMENTS
DEPARTMENT_ID
DEPARTMENT_NAME
MANAGER_ID
LOCATION_ID
EMPLOYEES
EMPLOYEE_ID
FIRST_NAME
LAST_NAME
EMAIL
PHONE_NUMBER
HIRE_DATE
JOB_ID
SALARY
COMMISSION_PCT
MANAGER_ID
DEPARTMENT_ID
COUNTRIES
COUNTRY_ID
COUNTRY_NAME
REGION_ID
JOBS
JOB_ID
JOB_TITLE
MIN_SALARY
MAX_SALARY
LOCATIONS
LOCATION_ID
STREET_ADDRESS
POSTAL_CODE
CITY
STATE_PROVINCE
COUNTRY_ID
JOB_HISTORY
EMPLOYEE_ID
START-DATE
END_DATE
JOB_ID
DEPARTMENT_ID

Subqueries

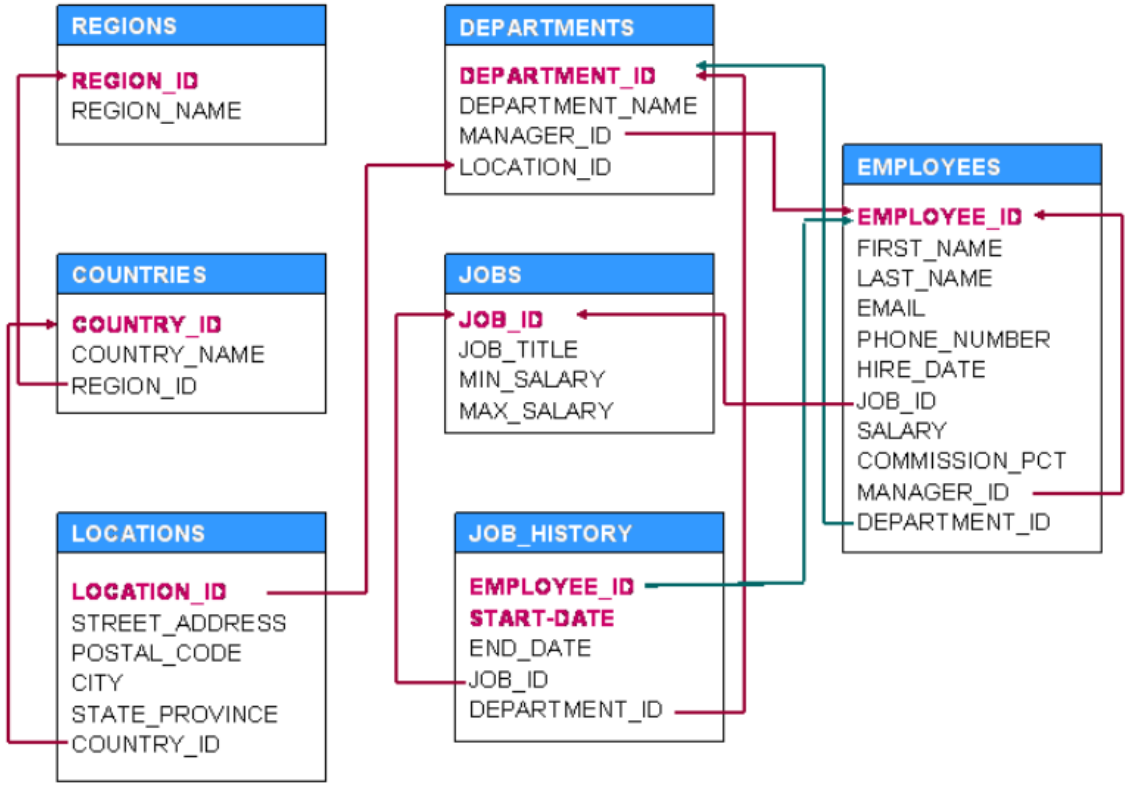
- Output of one query can be input to another (nesting)
- A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.
- By definition subquery is a query nested in another query such as SELECT, INSERT, UPDATE or DELETE.

Subqueries

- You can use a subquery in many places such as:
  - With the `IN` or `NOT IN` operator
  - With comparison operators
  - With the `EXISTS` or `NOT EXISTS` operator
  - With the `ANY` or `ALL` operator
  - In the `FROM` clause
  - In the `SELECT` clause

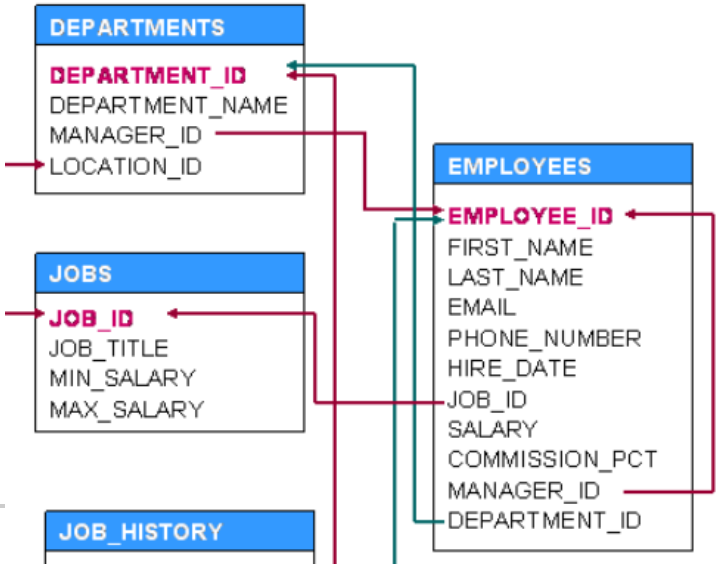
Subquery Example

Retrieve employees who are located in location # 1700



```sql
SELECT
    employee_id, first_name,
last_name
FROM
    employees
WHERE
    department_id IN
        (SELECT
            department_id
         FROM
            departments
         WHERE
            location_id = 1700)
ORDER BY first_name , last_name;
```

Example



Break the task into 2 queries

- Get department id of those at location_id=1700
- Get employee information of those having the department_id from the previous query

Relational Algebra Subqueries

Sailors (sid, sname, rating, age)

Reserves (sid, bid, day)

Boats (bid, bname, color)

- Find the names of sailors who've reserved boat #103
  - Join sailors and reserves then apply the selection on boat # 103

$\sigma_{bid=103}(Sailors \bowtie_{sailors.sid = reserves.sid} Reserves)$

  - First filter rows of boat # 103 and then run a join

$(Sailors \bowtie_{sailors.sid = reserves.sid} (\sigma_{bid=103} Reserves))$

SQL Subqueries

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

- Find the names of sailors who've reserved boat #103
  - Join sailors and reserves then apply the selection on boat # 103

```
SELECT DISTINCT sname FROM Sailors S JOIN Reserves R ON S.SID=R.SID
WHERE bid=103
```

  - First filter rows of boat # 103 and then run a join

```
SELECT DISTINCT sname FROM Sailors S JOIN (Select SID FROM Reserves R
WHERE R.bid=103) R2 ON S.Sid = R2.Sid
```

  - Find sid of sailors who have reserved 103 and then check if sid of Sailors table is part of the retrieved list.

```
SELECT DISTINCT sname FROM Sailors WHERE SID IN
(Select SID FROM Reserves R WHERE R.bid=103)
```

WITH clause

```
WITH cte_name (column1, column2, ...) AS (
    -- Subquery or SQL statement here
)
SELECT *
FROM cte_name;
```

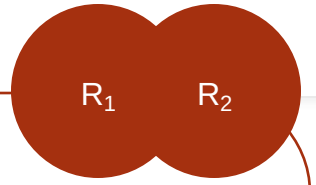
```
WITH sid_103 AS (
    SELECT sid FROM Reserves WHERE bid=103
)
SELECT * FROM sid_103;
```

```
WITH sid_103 AS (
    SELECT sid FROM Reserves WHERE bid=103
)
SELECT DISTINCT sname FROM Sailors S WHERE
sid IN (SELECT * FROM sid_103);
```

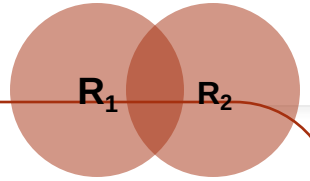
** You can use the CTE as another table available to this specific query*

- Also known as CTE – Common Table Expression used to define a temporary table or result set that can be referenced in another query in SELECT, INSERT, UPDATE or DELETE.
- Works like Assignment operator in relational algebra allowing to break complex SQL queries into smaller and manageable parts.

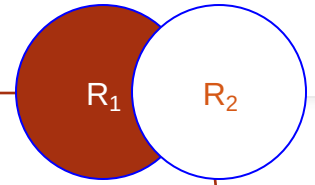
Set Operators




```
SELECT
    column1, column2
FROM
    table1
UNION [ALL]
SELECT
    column3, column4
FROM
    table2;
```




```
SELECT
    column1, column2
FROM
    table1
INTERSECT
SELECT
    column3, column4
FROM
    table2;
```




```
SELECT
    column1, column2
FROM
    table1
MINUS
SELECT
    column3, column4
FROM
    table2;
```

- To retain the duplicate rows in the result set, you use the UNION ALL operator.
- Ensure Union Compatibility
- Can apply order by after the second SELECT query if we wish to sort the output.

Comparison

Comparison operators
(=,>, >=, <, <=, <>)

Find the employee(s) who have the highest salary

```
SELECT
    employee_id, first_name, last_name,
salary
FROM
    employees
WHERE
    salary = (SELECT
            MAX(salary)
        FROM
            employees)
ORDER BY first_name , last_name;
```

Correlated Subqueries

- A subquery that uses values from the outer query.
- The inner query depends on the row that is currently being examined in the outer query.

```
SELECT
    employee_id, first_name, last_name, salary,
department_id
FROM
    employees e
WHERE
    salary > (SELECT
            AVG(salary)
        FROM
            employees
        WHERE
            department_id = e.department_id)
ORDER BY department_id, first_name, last_name;
```

EXISTS
NOT EXISTS

EXISTS Checks for the existence of rows returned from the subquery.

It returns *true* if the subquery contains any rows. Otherwise, it returns false.

Finds all departments which have *at least one employee* with the salary is greater than 10,000

```
SELECT
    department_name
FROM
    departments d
WHERE
    EXISTS( SELECT
            1
        FROM
            employees e
        WHERE
            salary > 10000
                AND e.department_id = d.department_id)
ORDER BY department_name;
```

ANY - ALL

Find all employees whose salaries are greater than the lowest salary of **every** department

X > ALL (subquery) – evaluates to true of x is greater than every value returned by the subquery.

X > ANY (subquery) – evaluates to true of x is greater than any value returned by the subquery.

```
SELECT
    employee_id, first_name, last_name,
salary
FROM
    employees
WHERE
    salary >= ALL (SELECT
            MIN(salary)
        FROM
            employees
        GROUP BY department_id)
ORDER BY first_name , last_name;
```

FROM

```
SELECT
    *
FROM
    (subquery) table_name
```

```
SELECT
    *
FROM
    (subquery) sub_table
JOIN table 2 ON sub_table.id = table2.id
```

SELECT

Find the salaries of all employees, the average salary at the company, and the difference between their salary and the company average.

```
SELECT
    employee_id,
    first_name,
    last_name,
    salary,
    (SELECT
            ROUND(AVG(salary), 0)
        FROM
            employees)
average_salary,
    salary - (SELECT
            ROUND(AVG(salary), 0)
        FROM
            employees) difference
FROM
    employees
-- WHERE department_ID=50
ORDER BY first_name , last_name;
```

Views

- A view is a virtual table based on the result-set of an SQL statement

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;

SELECT * FROM view_name;
```

Case

- The SQL CASE expression allows you to evaluate a list of conditions and returns one of the possible results.
- Can have values or Boolean expression in place of when_expression
- Useful to create new columns based on conditions

```
SELECT name, grade,
    CASE
    WHEN marks<60 THEN ‘Poor’
    WHEN marks>=60 AND marks<=80 THEN
    ‘Average’
    WHEN marks>80 THEN ‘Excellent’
    END Performance
FROM
   Students
```

```
CASE expression
WHEN when_expression_1 THEN
       result_1
WHEN when_expression_2 THEN
       result_2
WHEN when_expression_3 THEN
       result_3
...
ELSE
       else_result
END
```

Update

- Modify data of the existing rows a table

```
UPDATE table_name
SET column1 = value1,
 column2 = value2
WHERE
        condition;
```

```
UPDATE employees
SET
    last_name = ‘Alfred'
WHERE
    employee_id = 192;
```

```
UPDATE employees
SET
    salary = 1000
WHERE
    department_id = (SELECT
department_id FROM departments
WHERE department_name = ‘IT’);
```

Delete

- Delete one or more rows in a table

```
DELETE
FROM
        table_name
WHERE
        condition;
```

```
DELETE FROM employees
WHERE
    department_id = 16;
```

```
DELETE FROM employees
WHERE
    department_id = (SELECT
department_id FROM departments
WHERE department_name =
‘Temporary Department’;
```

Lab06
SQL...