COSC 304
Introduction to Database Systems

SQL

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

SQL Queries

Querying with SQL is performed using a `SELECT` statement. The general form of the statement is:

SELECT $A_1, A_2, \ldots, A_n$ ← **attributes in result**
FROM $R_1, R_2, \ldots, R_m$ ← **tables in query**
WHERE (*condition*)

Notes:

- 1) The "*" is used to select all attributes.
- 2) Combines the relational algebra operators of selection, projection, and join into a single statement.
- 3) Comparison operators: =, !=, >, <, >=, <=.

SQL and Relational Algebra

The `SELECT` statement can be mapped directly to relational algebra.

SELECT $A_1, A_2, \ldots, A_n$
FROM $R_1, R_2, \ldots, R_m$
WHERE P

is equivalent to:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

Example Relations

Relations:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

Foreign keys:

- emp: emp.supereno to emp.eno, emp.dno to dept.dno
- proj: proj.dno to dept.dno
- dept: dept.mgreno to emp.eno
- workson: workson.eno to emp.eno, workson.pno to proj.pno

Example Relation Instances

emp

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	1975-01-05	EE	30000	E2	null
E2	M. Smith	1966-06-04	SA	50000	E5	D3
E3	A. Lee	1966-07-05	ME	40000	E7	D2
E4	J. Miller	1950-09-01	PR	20000	E6	D3
E5	B. Casey	1971-12-25	SA	50000	E8	D3
E6	L. Chu	1965-11-30	EE	30000	E7	D2
E7	R. Davis	1977-09-08	ME	40000	E8	D1
E8	J. Jones	1972-10-11	SA	50000	null	D1

workson

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

proj

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

dept

dno	dname	mgreno
D1	Management	E8
D2	Consulting	E7
D3	Accounting	E5
D4	Development	null

One Relation Query Example

Return the employee name and salary of all employees whose title is 'EE':

```
SELECT ename, salary
FROM   emp
WHERE  title = 'EE'
```

Emp Relation

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Result

ename	salary
J. Doe	30000
L. Chu	30000

Algorithm: Scan each tuple in table and check if matches condition in `WHERE` clause.

One Relation Query Examples

Return the birth date and salary of employee 'J. Doe':

```
SELECT bdate, salary
FROM   emp
WHERE  ename = 'J. Doe'
```

Return all information on all employees:

```
SELECT *
FROM   emp
```

← * returns all attributes

Return the employee number, project number, and number of hours worked where the hours worked is > 50:

```
SELECT eno, pno, hours
FROM   workson
WHERE  hours > 50
```

Duplicates in SQL

One major difference between SQL and relational algebra is that relations in SQL are ***bags*** instead of sets.

- It is possible to have two or more identical rows in a relation.

Consider the query: Return all titles of employees.

```
SELECT title
FROM   emp
```

Emp Relation

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Result

title
EE
SA
ME
PR
SA
EE
ME
SA

Duplicates in SQL - DISTINCT clause

To remove duplicates, use the **DISTINCT** clause in the SQL statement:

```
SELECT DISTINCT title
FROM   emp
```

Result

title
EE
SA
ME
PR

SQL Practice Questions Single Table

Relational database schema:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

1) Return the project names that have a budget > 250000.

2) Return the employee numbers who make less than $30000.

3) Return the list of `workson` responsibilities (`resp`) with no duplicates.

4) Return the employee (names) born after July 1, 1970 that have a salary > 35000 and have a title of 'SA' or 'PR'.

- Write the equivalent relational algebra expression.

Join Query Example

Multiple tables can be queried in a single SQL statement by listing them in the `FROM` clause.

- Note that if you do not specify any join condition to relate them in the `WHERE` clause, you get a *cross product* of the tables.

Example: Return the employees who are assigned to the 'Management' department.

```
SELECT ename
FROM   emp, dept
WHERE  dname = 'Management'
        and emp.dno = dept.dno
```

Result

ename
R. Davis
J. Jones

Join Query Examples

Return the department names and the projects in each department:

```
SELECT dname, pname
FROM   dept, proj
WHERE  dept.dno = proj.dno
```

Return the employees and the names of their department:

```
SELECT ename, dname
FROM   emp, dept
WHERE  emp.dno=dept.dno
```

Return all projects who have an employee working on them whose title is 'EE':

```
SELECT pname
FROM   emp, proj, workson
WHERE  emp.title = 'EE' and workson.eno=emp.eno
        and workson.pno = proj.pno
```

SQL Query Question

Question: What query would return the name and salary of employees working on project `'P3'`:

A)
```
SELECT ename, salary
 FROM emp, workson
 WHERE emp.eno = workson.eno and pno = 'P3'
```

B)
```
SELECT ename, salary
 FROM emp, workson, proj
 WHERE emp.eno = workson.eno and pno = "P3"
```

SQL Practice Questions
Joins

Relational database schema:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

1) For each employee, return their name and their department name.

2) Return the list of project names for the department with name `'Consulting'`.

3) Return `workson` records (eno, pno, resp, hours) where project budget is > $50000 and hours worked is < 20.

4) Return a list of all department names, the names of the projects of that department, and the name of the manager of each department.

Calculated Fields

Mathematical expressions are allowed in the `SELECT` clause to perform simple calculations.

- When an expression is used to define an attribute, the DBMS gives the attribute a unique name such as `col1`, `col2`, etc.

Example: Return how much employee 'A. Lee' will get paid for his work on each project.

```
SELECT ename, pname, salary/52/5/8*hours
FROM   emp, workson, proj
WHERE  emp.eno = workson.eno and ename='A. Lee'
        and proj.pno = workson.pno
```

Result

ename	pname	col3
A. Lee	Budget	192.31
A. Lee	Maintenance	923.08

Renaming and Aliasing

Often it is useful to be able to rename an attribute in the final result (especially when using calculated fields). Renaming is accomplished using the keyword **AS**:

```
SELECT ename, pname, salary/52/5/8*hours AS pay
FROM   emp, workson, proj
WHERE  emp.eno = workson.eno and ename='A. Lee'
        and proj.pno = workson.pno
```

Result

ename	pname	pay
A. Lee	Budget	192.31
A. Lee	Maintenance	923.08

Note: AS keyword is optional.

Renaming and Aliasing (2)

Renaming is also used when two or more copies of the same table are in a query. Using ***aliases*** allows you to uniquely identify what table you are talking about.

Example: Return the employees and their managers where the managers make less than the employee.

```
SELECT E.ename, M.ename
FROM   emp as E, emp as M
WHERE  E.supereno = M.eno and E.salary > M.salary
```

Advanced Conditions - BETWEEN

Sometimes the condition in the WHERE clause will request tuples where one attribute value must be in a *range* of values.

Example: Return the employees who make at least $20,000 and less than or equal to $45,000.

```
SELECT ename
FROM   emp
WHERE  salary >= 20000 and salary <= 45000
```

We can use the keyword **BETWEEN** instead:

```
SELECT ename
FROM   emp
WHERE  salary BETWEEN 20000 and 45000
```

Advanced Conditions - LIKE

For string valued attributes, the **LIKE** operator is used to search for partial matches.

- Partial string matches are specified by using either "**%**" that replaces an arbitrary number of characters or underscore "**_**" that replaces a single character.

Example: Return all employee names that start with 'A'.

```
SELECT ename
FROM   emp
WHERE  ename LIKE 'A%'
```

Example: Return all employee names who have a first name that starts with 'J' and whose last name is 3 characters long.

```
SELECT ename
FROM   emp
WHERE  ename LIKE 'J. ___'
```

Performance Concerns of `LIKE`

Warning: Do not use the `LIKE` operator if you do not have to.

It is often an inefficient operation as the DBMS may not be able to optimize lookup using `LIKE` as it can for equal (=) comparisons. The result is the DBMS often has to examine ALL TUPLES in the relation.

In almost all cases, adding indexes will ***not*** increase the performance of `LIKE` queries because the indexes cannot be used.

- Most indexes are implemented using B-trees that allow for fast equality searching and efficient range searches.

Advanced Conditions - IN

To specify that an attribute value should be in a given set of values, the **IN** keyword is used.

- Example: Return all employees who are in any one of the departments {'D1', 'D2', 'D3'}.

```
SELECT ename
FROM   emp
WHERE  dno IN ('D1','D2','D3')
```

Note that this is equivalent to using OR:

```
SELECT ename
FROM   emp
WHERE  dno = 'D1' OR dno = 'D2' OR dno = 'D3'
```

However, we will see more practical uses of `IN` and `NOT IN` when we study nested subqueries.

Advanced Conditions - NULL

Remember NULL is used to indicate that a given attribute does not have a value. To determine if an attribute is NULL, we use the clause **IS NULL**.

- Note that you cannot test NULL values using = and <>.

Example: Return all employees who are not in a department.

```
SELECT ename
FROM   emp
WHERE  dno IS NULL
```

Example: Return all departments that have a manager.

```
SELECT dname
FROM   dept
WHERE  mgreno IS NOT NULL
```

Set Operations

The set operations of union, intersection, and difference are used to combine the results of two SQL queries.

- UNION , INTERSECT, EXCEPT
- Note: UNION ALL returns all rows

Example: Return the employees who are either directly supervised by 'R. Davis' or directly supervised by 'M. Smith'.

```
(SELECT E.ename
FROM   emp as E, emp as M
WHERE  E.supereno = M.eno and M.ename='R. Davis')

UNION

(SELECT E.ename
FROM   emp as E, emp as M
WHERE  E.supereno = M.eno and M.ename='M. Smith')
```

SELECT INTO

The result of a select statement can be stored in a temporary table using the **INTO** keyword.

```
SELECT E.ename
INTO   davisMgr
FROM   emp as E, emp as M
WHERE  E.supereno = M.eno and M.ename='R. Davis'
```

This can be used for set operations instead of using parentheses:

```
SELECT E.ename
INTO   smithMgr
FROM   emp as E, emp as M
WHERE  E.supereno = M.eno and M.ename='M. Smith'
davisMgr UNION smithMgr;
```

Ordering Result Data

The query result returned is not ordered on any attribute by default. We can order the data using the **ORDER BY** clause:

```
SELECT    ename, salary, bdate
FROM      emp
WHERE     salary > 30000
ORDER BY  salary DESC, ename ASC
```

- 'ASC' sorts the data in ascending order, and 'DESC' sorts it in descending order. The default is 'ASC'.
- The order of sorted attributes is significant. The first attribute specified is sorted on first, then the second attribute is used to break any ties, etc.
- NULL is normally treated as less than all non-null values.

LIMIT and OFFSET

If you only want the first *N* rows, use a **LIMIT** clause:

```
SELECT   ename, salary FROM emp ORDER BY salary DESC
LIMIT 5
```

To start from a row besides the first, use **OFFSET**:

```
SELECT   eno, salary FROM emp ORDER BY eno DESC
LIMIT 3  OFFSET 2
```

- LIMIT improves performance by reducing amount of data processed and sent by the database system.
- OFFSET 0 is first row, so OFFSET 2 would return the 3rd row.
- LIMIT/OFFSET syntax supported differently by systems.
  - MySQL, PostgreSQL – use LIMIT syntax
  - Oracle – uses ROWNUM field that can be filtered in WHERE
  - SQL Server – use SELECT TOP *N*

SQL Querying with NULL and LIKE

Question: What query would return the department names that do not have a manager or contain `'ent'`.

A)
```
SELECT dname
FROM   dept
WHERE mgreno = NULL OR dname LIKE '_ent'
```

B)
```
SELECT dname
FROM   dept
WHERE mgreno IS NULL OR dname LIKE '%ent%'
```

SQL Practice Questions Expressions, LIKE, IS NULL

Relational database schema:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

1) Calculate the monthly salary for each employee.

2) List all employee names who do not have a supervisor.

3) List all employee names where the employee's name contains an 'S' and workson responsibility that ends in 'ER'.

4) Return the list of employees (names) who make less than their managers and how much less they make.

5) Return only the top 3 project budgets in descending order.

SQL Practice Questions
Set Operations, ORDER BY

Relational database schema:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

1) Return the list of employees sorted by salary (desc) and then title (asc).

2) Return the employees (names) who either manage a department or manage another employee.

3) Return the employees (names) who manage an employee but do not manage a department.

4) Give a list of all employees who work on a project for the 'Management' department ordered by project number (asc).

5) **Challenge:** Return the projects (names) that have their department manager working on them.

Aggregate Queries and Functions

Several queries cannot be answered using the simple form of the `SELECT` statement. These queries require a summary calculation to be performed. Examples:

- What is the maximum employee salary?
- What is the total number of hours worked on a project?
- How many employees are there in department 'D1'?

To answer these queries requires the use of aggregate functions. These functions operate on a single column of a table and return a single value.

Aggregate Functions

The five basic aggregate functions are:

- `COUNT` - returns the # of values in a column
- `SUM` - returns the sum of the values in a column
- `AVG` - returns the average of the values in a column
- `MIN` - returns the smallest value in a column
- `MAX` - returns the largest value in a column

Notes:

- 1) `COUNT`, `MAX`, and `MIN` apply to all types of fields, whereas `SUM` and `AVG` apply to only numeric fields.
- 2) Except for `COUNT(*)` all functions ignore nulls. `COUNT(*)` returns the number of rows in the table.
- 3) Use `DISTINCT` to eliminate duplicates.

Aggregate Function Example

Return the number of employees and their average salary.

```
SELECT COUNT(eno) AS numEmp, AVG(salary) AS avgSalary
FROM   emp
```

Result

numEmp	avgSalary
8	38750

GROUP BY Clause

Aggregate functions are most useful when combined with the `GROUP BY` clause. The **`GROUP BY`** clause groups the tuples based on the values of the attributes specified.

When used in combination with aggregate functions, the result is a table where each tuple consists of unique values for the group by attributes and the result of the aggregate functions applied to the tuples of that group.

GROUP BY Example

For each employee title, return the number of employees with that title, and the minimum, maximum, and average salary.

```
SELECT   title, COUNT(eno) AS numEmp,
         MIN(salary) as minSal,
         MAX(salary) as maxSal, AVG(salary) AS avgSal
FROM     emp
GROUP BY title
```

Result

title	numEmp	minSal	maxSal	avgSal
EE	2	30000	30000	30000
SA	3	50000	50000	50000
ME	2	40000	40000	40000
PR	1	20000	20000	20000

GROUP BY Clause Rules

There are a few rules for using the GROUP BY clause:

- 1) A column name cannot appear in the SELECT part of the query unless it is part of an aggregate function or in the list of group by attributes.
  - Note that the reverse is allowed: a column can be in the GROUP BY without being in the SELECT part.

- 2) Any WHERE conditions are applied before the GROUP BY and aggregate functions are calculated.

HAVING Clause

The **HAVING** clause is applied ***AFTER*** the `GROUP BY` clause and aggregate functions are calculated.

It is used to filter out entire *groups* that do not match certain criteria.

The **HAVING** clause can contain any condition that references aggregate functions and the group by attributes themselves.

- However, any conditions on the `GROUP BY` attributes should be specified in the `WHERE` clause if possible due to performance reasons.

HAVING Example

Return the title and number of employees of that title where the number of employees of the title is at least 2.

```
SELECT   title, COUNT(eno) AS numEmp
FROM     emp
GROUP BY title
HAVING   COUNT(eno) >= 2
```

Result

title	numEmp
EE	2
SA	3
ME	2

GROUP BY/HAVING Example

For employees born after December 1, 1965, return the average salary by department where the average is > 40,000.

```
SELECT    dname, AVG(salary) AS avgSal
FROM      emp, dept
WHERE     emp.dno = dept.dno and
             emp.bdate > DATE '1965-12-01'
GROUP BY  dname
HAVING    AVG(salary) > 40000
```

Step #1: Perform Join and Filter in `WHERE` clause

eno	ename	bdate	title	salary	supereno	dno	dname	mgreno
E2	M. Smith	1966-06-04	SA	50000	E5	D3	Accounting	E5
E3	A. Lee	1966-07-05	ME	40000	E7	D2	Consulting	E7
E5	B. Casey	1971-12-25	SA	50000	E8	D3	Accounting	E5
E7	R. Davis	1977-09-08	ME	40000	E8	D1	Management	E8
E8	J. Jones	1972-10-11	SA	50000	null	D1	Management	E8

GROUP BY/HAVING Example (2)

Step #2: GROUP BY on dname

eno	ename	bdate	title	salary	supereno	dno	dname	mgreno
E2	M. Smith	1966-06-04	SA	50000	E5	D3	Accounting	E5
E5	B. Casey	1971-12-25	SA	50000	E8	D3	Accounting	E5
E3	A. Lee	1966-07-05	ME	40000	E7	D2	Consulting	E7
E7	R. Davis	1977-09-08	ME	40000	E8	D1	Management	E8
E8	J. Jones	1972-10-11	SA	50000	null	D1	Management	E8

Step #3: Calculate aggregate functions

dname	avgSal
Accounting	50000
Consulting	40000
Management	45000

Step #4: Filter groups using HAVING clause

dname	avgSal
Accounting	50000
Management	45000

GROUP BY Examples

Return the average budget per project:

```
SELECT AVG(budget)
FROM   proj
```

Return the average # of hours worked on each project:

```
SELECT   pno, AVG(hours)
FROM     workson
GROUP BY pno
```

Return the departments that have projects with at least 2 'EE's working on them:

```
SELECT   DISTINCT proj.dno
FROM     proj, workson, emp
WHERE    emp.title = 'EE' and workson.eno=emp.eno
           and workson.pno = proj.pno
GROUP BY proj.dno, proj.pno
HAVING   COUNT(*) >=2
```

GROUP BY/HAVING
Multi-Attribute Example

Return the employee number, department number and hours the employee worked per department where the hours is >= 10.

```
SELECT   W.eno, D.dno, SUM(hours)
FROM     workson AS W, dept AS D, proj AS P
WHERE    W.pno = P.pno and P.dno = D.dno
GROUP BY W.eno, D.dno
HAVING   SUM(hours) >= 10
```

Result:

eno	dno	SUM(hours)
E1	D1	12
E2	D1	24
E3	D2	48
E3	D3	10
E4	D2	18
E5	D2	24
E6	D2	48
E7	D3	36

Question:
1) How would you only return records for departments D2 and D3?

SQL Querying with GROUP BY

Question: Of the following queries, select one which is invalid.

A)
```
SELECT dname
FROM   dept
GROUP BY dno
```

B)
```
SELECT COUNT(*)
FROM   dept
```

C)
```
SELECT dno, COUNT(*)
FROM   dept
```

D)
```
SELECT dno, COUNT(*)
FROM   dept  WHERE mgreno > 'A'
GROUP BY dno, dname
```

GROUP BY Practice Questions

Relational database schema:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

1) Return the highest salary of any employee.

2) Return the smallest project budget.

3) Return the department number and average budget for its projects.

4) For each project, return its name and the total number of hours employees have worked on it.

5) For each employee, return the total number of hours they have worked. Only show employees with more than 30 hours.

Subqueries

SQL allows a single query to have multiple subqueries nested inside of it. This allows for more complex queries to be written.

When queries are nested, the outer statement determines the contents of the final result, while the inner `SELECT` statements are used by the outer statement (often to lookup values for `WHERE` clauses).

A subquery can be in the `SELECT`, `FROM`, `WHERE` or `HAVING` clause.

```
SELECT    ename, salary, bdate
FROM      emp
WHERE     salary > (SELECT AVG(salary) FROM emp)
```

Types of Subqueries

There are three types of subqueries:

- 1) ***scalar subqueries*** - return a single value. Often value is then used in a comparison.
  - ⇨ If query is written so that it expects a subquery to return a single value, and if it returns multiple values or no values, a run-time error occurs.
- 2) ***row subquery*** - returns a single row which may have multiple columns.
- 3) ***table subquery*** - returns one or more columns and multiple rows.

Scalar Subquery Examples

Return the employees that are in the 'Accounting' department:

```
SELECT ename
FROM   emp
WHERE  dno = (SELECT dno FROM dept
              WHERE dname = 'Accounting')
```

Return all employees who work more hours than average on a single project:

```
SELECT ename
FROM   emp, workson
WHERE  workson.eno = emp.eno AND
       workson.hours > (SELECT AVG(hours) FROM workson)
```

Table Subqueries

A table subquery returns a relation. There are several operators that can be used:

- `EXISTS` *R* - true if *R* is not empty
- *s* `IN` *R* - true if *s* is equal to one of the values of *R*
- *s* > `ALL` *R* - true if *s* is greater than **every** value in *R*
- *s* > `ANY` *R* - true if *s* is greater than **any** value in *R*

Notes:

- 1) Any of the comparison operators (<, <=, =, etc.) can be used.
- 2) The keyword `NOT` can proceed any of the operators.
  - Example: *s* `NOT IN` *R*

Table Subquery Examples

Return all departments who have a project with a budget greater than $300,000:

```
SELECT dname FROM dept WHERE dno IN
(SELECT dno FROM proj WHERE budget > 300000)
```

Return all projects that 'J. Doe' works on:

```
SELECT pname FROM proj WHERE pno IN
       (SELECT pno FROM workson WHERE eno =
          (SELECT eno FROM emp WHERE ename = 'J. Doe'))
```

EXISTS Example

The EXISTS function is used to check whether the result of a nested query is empty or not.

- EXISTS returns true if the nested query has 1 or more tuples.

Example: Return all employees who have the same name as someone else in the company.

```
SELECT ename
FROM   emp as E
WHERE  EXISTS (SELECT * FROM emp as E2
               WHERE E.ename = E2.ename AND
                     E.eno <> E2.eno)
```

ANY and ALL Example

ANY means that any value returned by the subquery can satisfy the condition.

ALL means that all values returned by the subquery must satisfy the condition.

Example: Return the employees who make more than all the employees with title 'ME' make.

```
SELECT ename
FROM   emp as E
WHERE  salary > ALL (SELECT salary FROM emp
                     WHERE title = 'ME')
```

Subquery Syntax Rules

1) The `ORDER BY` clause may not be used in a subquery.

2) The number of attributes in the `SELECT` clause in the subquery must match the number of attributes compared to with the comparison operator.

3) Column names in a subquery refer to the table name in the `FROM` clause of the subquery by default. You must use aliasing if you want to access a table that is present in both the inner and outer queries.

Correlated Subqueries

Most queries involving subqueries can be rewritten so that a subquery is not needed.

- This is normally beneficial because query optimizers may not do a good job at optimizing queries containing subqueries.

A nested query is ***correlated*** with the outside query if it must be re-computed for every tuple produced by the outside query. Otherwise, it is ***uncorrelated***, and the nested query can be converted to a non-nested query using joins.

A nested query is correlated with the outer query if it contains a reference to an attribute in the outer query.

Correlated Subquery Example

Return all employees who have the same name as another employee:

```
SELECT ename
FROM   emp as E
WHERE  EXISTS (SELECT eno FROM emp as E2
               WHERE E.ename = E2.ename AND
                     E.eno <> E2.eno)
```

A more efficient solution with joins:

```
SELECT E.ename
FROM   emp as E, emp as E2
WHERE  E.ename = E2.ename AND E.eno <> E2.eno
```

Explicit Join Syntax

You can cpecify a join condition directly in the `FROM` clause instead of the `WHERE`.

Example #1: Return the employees who are assigned to the `'Management'` department:

```
SELECT ename
FROM   emp JOIN dept ON emp.dno = dept.dno
WHERE  dname = 'Management'
```

Example #2: Return all projects who have an employee working on them whose title is `'EE'`:

```
SELECT pname
FROM   emp E JOIN workson W ON E.eno = W.eno
          JOIN proj AS P ON W.pno = P.pno
WHERE  E.title = 'EE'
```

Outer Joins

Using joined tables in the `FROM` clause allows outer joins and natural joins to be specified as well.

- Types: `NATURAL JOIN`, `FULL OUTER JOIN`, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`
  - The keyword "`outer`" can be omitted for outer joins.

Example: Return all departments (even those without projects) and their projects.

```
SELECT dname, pname
FROM   dept LEFT OUTER JOIN proj ON dept.dno = proj.dno

SELECT dname, pname
FROM   dept LEFT OUTER JOIN proj USING (dno)

SELECT dname, pname
FROM   dept NATURAL LEFT JOIN proj
```

Subqueries in FROM Clause

Subqueries are used in the FROM clause to produce temporary table results for use in the current query.

Example: Return the departments that have an employee that makes more than $40,000.

```
SELECT dname
FROM   Dept D, (SELECT ename, dno FROM Emp
                WHERE salary > 40000) E
WHERE D.dno = E.dno
```

- Note: The alias for the derived table is required.

SQL Querying with Subqueries

Question: What query below is equivalent to:

```
SELECT ename
FROM   emp as E
WHERE  salary > ALL (SELECT salary FROM emp)
```

A)

```
SELECT ename
FROM   emp as E
WHERE  salary > (SELECT MAX(salary) FROM emp)
```

B)

```
SELECT ename
FROM   emp as E
WHERE  salary > (SELECT SUM(salary) FROM emp)
```

Subquery Practice Questions

Relational database schema:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

1) List all departments that have at least one project.

2) List the employees who are not working on any project.

3) List the employees with title 'EE' that make more than all employees with title 'PR'.

4) Find all employees who work on some project that 'J. Doe' works on.

SQL Functions

Databases have many built-in functions that can be used when writing queries. Syntax and support varies between systems.

- Date: DATEDIFF, YEAR, GETDATE
- String: CONCAT, UPPER, LEFT, SUBSTRING
- Logical: CASE, IIF, ISNULL
- Aggregate: SUM, COUNT, AVG
- Note: Case-insensitive function names.

Example:

```
SELECT eno, UPPER(ename),
  CASE title WHEN 'EE' THEN 'Engineer'
  WHEN 'SA' THEN 'Admin' ELSE 'Other' END as role,
  year(bdate) as birthYear
FROM emp
WHERE salary * 2 > 60000
```

SQL Query Summary

The general form of the `SELECT` statement is:

SELECT <attribute list>
FROM <table list>
[WHERE (*condition*)]
[GROUP BY <grouping attributes>]
[HAVING <group condition>]
[ORDER BY <attribute list>]
[LIMIT <num> [OFFSET <offset>]]

- Clauses in square brackets ([,]) are optional.
- There are often numerous ways to express the same query in SQL.

Your Own Practice Questions

Relational database schema:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

Given the above schema answer your own English questions using SQL.

Are there some questions that you cannot answer?

Conclusion

SQL is the standard language for querying relational databases.

The **`SELECT`** statement is used to query data and combines the relational algebra operations of selection, projection, and join into one statement

- There are often many ways to specify the same query.
- Aggregate functions, recursive queries, outer joins

`INSERT`, **`DELETE`**, and **`UPDATE`** statements are used to modify the data stored in a single relation.

Objectives

- Be able to write an English query in SQL.
- Be able to convert basic SQL queries into relational algebra expressions.
- Be able to write the various types of queries:
  - basic queries involving selection, projection, and join
  - queries involving renaming and aliasing including queries involving multiple copies of the same relation
  - queries containing aggregate functions and calculated fields
  - nested subqueries and the use of the appropriate operators:
    - comparison operators for single value subqueries
    - `IN`, `NOT IN`, `ANY`, `ALL` for table result subqueries
    - `EXISTS` and `NOT EXISTS` for multiple result subqueries which may or may not contain results