

LECTURE 4: STRUCTURE QUERY LANGUAGE (SQL) 2

COMP2004J: Databases and Information Systems

Dr. Ruihai Dong (Ruihai.dong@ucd.ie)

UCD School of Computer Science

Beijing-Dublin International College

SQL Queries

- SQL queries are expressed by the **SELECT** statement

- Syntax:

```
SELECT attr_expr [ AS alias ] {, attr_expr [AS alias ] }  
FROM table [AS alias ] {, table [ AS alias ] }  
[ WHERE condition ] ;
```

- the three parts of the query are usually called:
 - target list
 - from clause
 - where clause

Square brackets indicate optional portions: []

Curly brackets (also known as “braces”) indicate optional lists: {}

SQL Queries: Order of operations

1. The query considers the Cartesian product of the tables in the FROM clause.
2. Then considers only the rows that satisfy the condition in the WHERE clause.
3. And finally for each row evaluates the attribute expressions in the target list.

Cartesian Product

- Cartesian product of two relations R and S ($R \times S$) is a relation composed of **all possible tuples** where the first components are tuples of R and the last components are tuples from S.
- If R has degree **x** and S has degree **y** then $R \times S$ has degree **(x+y)**
- Attributes in the new relation keep their names from the original relations prefixed by a given table name or table alias, plus a dot.
- If R has a tuples and S has b tuples, then $R \times S$ has (a x b) tuples

Cartesian Product Example

Relation R

A	B	C
1	2	3
4	5	6
7	8	9

Relation S

D	E	F
9	8	7
1	2	3
3	2	1

Cartesian Product Example

Relation R

A	B	C
1	2	3
4	5	6
7	8	9

Relation S

D	E	F
9	8	7
1	2	3
3	2	1

R X S

A	B	C	D	E	F
1	2	3	9	8	7
1	2	3	1	2	3
1	2	3	3	2	1
4	5	6	9	8	7
4	5	6	1	2	3
4	5	6	3	2	1
7	8	9	9	8	7
7	8	9	1	2	3
7	8	9	3	2	1

Examples

employee

empno	name	job	salary	deptno
1234	Sean Russell	Teacher	50000	10
4567	Jamie Heaslip	Manager	47000	10
6542	Leo Cullen	Teacher	45000	10
1238	Brendan Macken	Technician	25000	20
1555	Sean O'Brien	Designer	50000	20
1899	Brian O'Driscoll	Manager	45000	20
2525	Peter Stringer	Designer	25000	30
1585	Denis Hickey	Architect	20000	30
1345	Ronan O'Gara	Manager	29000	30

department

deptno	deptname	office	division	managerno
10	Training	Lansdown	D1	4567
20	Design	Belfield	D2	1899
30	Implementation	Donnybrook	D1	1345

Query Examples

- employee(empno, name, job, salary, *deptno*)
- department(deptno, deptname, office, division, *managerno*)
- SELECT salary FROM employees WHERE job = 'Technician';

salary
25000

- SELECT * FROM employees WHERE job = 'Manager';

empno	name	job	salary	deptno
4567	Jamie Heaslip	Manager	47000	10
1899	Brian O'Driscoll	Manager	45000	20
1345	Ronan O'Gara	Manager	29000	30

Attribute expressions

- employee(empno, name, job, salary, *deptno*)
- department(deptno, deptname, office, division, managerno)
- `SELECT name, salary/12 AS monthllysalary FROM employee;`

<u>name</u>	<u>monthllysalary</u>
Sean Russell	4166.6667
Brendan Macken	2083.3333
Ronan O'Gara	2416.6667
Sean O'Brien	4166.6667
Denis Hickey	1666.6667
Brian O'Driscoll	3750
Peter Stringer	2083.3333
Jamie Heaslip	3916.6667
<u>Leo Cullen</u>	<u>3750</u>

Join Query

- Find the names of the employees and the office they work in:
 - `SELECT name, deptname FROM employee, department
WHERE employee.deptno = department.deptno;`

name	deptname
Sean Russell	Training
Brendan Macken	Design
Ronan O'Gara	Implementation
Sean O'Brien	Design
Denis Hickey	Implementation
Brian O'Driscoll	Design
Peter Stringer	Implementation
Jamie Heaslip	Training
Leo Cullen	Training

Aliases

- We can rename attributes and tables in our queries.
- We have previously seen attribute renaming.
- We can also rename tables to shorten our queries.
- `SELECT name, deptname FROM employee AS e,
department AS d WHERE e.deptno = d.deptno;`

More Complex Queries

- Find the names of the employees who work in the Lansdowne office of division D1:
- ```
SELECT name FROM employee AS e, department AS d
 WHERE e.deptno = d.deptno
 AND d.division = 'D1'
 AND d.office = 'Lansdowne';
```

---

name

Sean Russell

Jamie Heaslip

Leo Cullen

# More Complex Queries

- Find the names of the employees who work in either the Lansdowne office or the Belfield:

```
SELECT name FROM employee AS e, department AS d
WHERE e.deptno=d.deptno
 AND (d.office='Belfield' OR d.office='Lansdowne');
```

---

name

Sean Russell

Brendan Macken

Sean O'Brien

Brian O'Driscoll

Jamie Heaslip

Leo Cullen

---

# The LIKE Operator

- Find the details of all employees where the second letter of their name is 'e'.
- `SELECT * FROM employee WHERE name LIKE '_e%';`
  - The '\_' represents any single character.
  - The '%' represents any number of characters.
- Find the details of all employees whose job title contains exactly seven letters
- `SELECT * FROM employee WHERE job like '_____';`

# Null Values

- NULL values may mean that:
  - a value is not applicable.
  - a value is applicable but unknown.
  - it is unknown if a value is applicable or not.
- Previous standards of SQL used two-valued logic
  - Comparison with NULL returns FALSE.
- SQL-2 (and later) use a three-valued logic
  - a comparison with NULL returns UNKNOWN.

# Testing for Null Values

- In a SELECT query, we may want to test if an attribute contains a NULL value.
  - `attribute IS NULL`
  - `attribute IS NOT NULL`
- Example
  - `SELECT * FROM employee WHERE job IS NULL;`
  - `SELECT * FROM employee WHERE deptno IS NOT NULL;`



# Duplicates

- In SQL, result tables may have identical rows.
- Duplicates can be removed using the keyword **DISTINCT**.
- `SELECT DISTINCT job FROM employee;`

# Inserting Data

- Syntax (two options):

- `INSERT INTO table_name [ (attribute_list) ] VALUES (list_of_values);`
- `INSERT INTO table_name SELECT ...;`

When inserting only some of the attributes, we must list the attribute we set.

- Using values:

- `INSERT INTO department (deptno, deptname) VALUES (13, 'Finance');`
- `INSERT INTO department VALUES (13, 'Finance', 1234);`

We don't need to list the attributes if we're inserting all of them.

- Using a query:

- `INSERT INTO londonproducts (SELECT code, description FROM product WHERE product_area = 'London');`

# Inserting Data

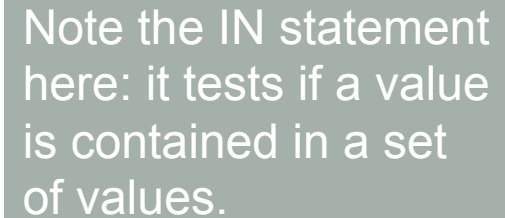
- The ordering of the attributes (if present) and of values is meaningful (first value with the first attribute, and so on).
- If we don't include an attribute list, all the attributes in the table are considered, in the same order as in the table definition
- If the attribute list does not contain all the table's attributes, the remaining attributes are assigned the default value (if defined) or the null value.

# Inserting Multiple Rows of Data

- It is also possible to insert multiple rows in the same query.
  - This will be faster than using a separate INSERT statement for each row.
- With attribute list
  - ```
INSERT INTO department (deptno, deptname, managernumber)
VALUES (13, 'Finance', 1234),
      (18, 'Human Resources', 8888),
      (19, 'Production', 7897);
```
- Without attribute list
 - ```
INSERT INTO department VALUES
 (13, 'Finance', 'Raglan Road', 'D3' , 1234),
 (18, 'Human Resources', 'Wexford St.', 'D1', 8888),
 (19, 'Production' 'South Circular Road', 'D5', 7897);
```

# Deleting Data

Note the IN statement here: it tests if a value is contained in a set of values.



- Syntax:
  - `DELETE FROM table_name [ WHERE condition ]`
- Remove the production department:
  - `DELETE FROM department WHERE deptname = 'Production';`
- Remove the departments without employees:
  - `DELETE FROM department WHERE deptno NOT IN (SELECT deptno FROM employee);`
- The delete statement removes from the table all the tuples that satisfy the condition.
  - The WHERE clause works in the same way as a SELECT statement, but all matching tuples are deleted.

# Deleting Data

- The removal may produce deletions from other tables if a referential integrity constraint with cascade policy has been defined.
- If there is no WHERE clause, delete removes **all** the tuples in the table.
- To remove all the tuples from Departments (the table will still exist, but be empty)
  - `DELETE FROM department;`
- To remove table departments completely (including the definition of the table)
  - `DROP TABLE department;`

# Updating Data

- The UPDATE statement is used to change the attributes of tuples that have already been inserted into a table.
- Syntax:

```
UPDATE table_name
SET attribute = <expression | SELECT... | NULL | DEFAULT>
{, attribute = <expression | SELECT... | NULL | DEFAULT>}
[WHERE condition];
```

- Set the salary of employee 1555 to 54000:
  - `UPDATE employee SET salary=54000 WHERE empno=1555;`
- Give all employees of department number 20 a 5% increase in salary (UPDATE changes every tuple that matches the WHERE clause):
  - `UPDATE employee SET salary=salary*1.05 WHERE deptno=20;`

# JOINING TABLES

---



# Joining Tables

- SQL-2 introduced a new syntax for joins, representing them explicitly in the `FROM` clause:

```
SELECT attr_expr [AS alias] {, attr_expr [AS alias] }
FROM table [AS alias] { [JoinType] JOIN table [AS alias]
ON JoinConditions }
[WHERE OtherCondition]
```

# Join Types

- JoinType describes the way in which the two tables are joined.
- The type can be any of following;
  - INNER
  - RIGHT [ OUTER ],
  - LEFT [ OUTER ]
  - FULL [ OUTER ]

# Tables

employee

| empno | name         | job     | deptno |
|-------|--------------|---------|--------|
| 1     | Sean Russell | Teacher | 1      |
| 2     | Ruihai Dong  | Teacher | 1      |
| 3     | Lina Xu      | Teacher | NULL   |
| 4     | Ning Cao     | Manager | 2      |

department

| deptno | deptname |
|--------|----------|
| 1      | CS       |
| 2      | IoT      |
| 3      | SE       |

# INNER JOIN

- Join two tables together based on some condition.
- Results are only returned if there is matching results in **both** tables.
- If there is no match for a row in either table, then the row is not shown.

# INNER JOIN

employee

| empno | name         | job     | deptno |
|-------|--------------|---------|--------|
| 1     | Sean Russell | Teacher | 1      |
| 2     | Ruihai Dong  | Teacher | 1      |
| 3     | Lina Xu      | Teacher | NULL   |
| 4     | Ning Cao     | Manager | 2      |

department

| deptno | deptname |
|--------|----------|
| 1      | CS       |
| 2      | IoT      |
| 3      | SE       |

- Find the details of the employees and the department in which they work:
- `SELECT * FROM employee AS e INNER JOIN department AS d ON e.deptno = d.deptno;`

| empno | name         | job     | deptno | deptno | deptname |
|-------|--------------|---------|--------|--------|----------|
| 1     | Sean Russell | Teacher | 1      | 1      | CS       |
| 2     | Ruihai Dong  | Teacher | 1      | 1      | CS       |
| 4     | Ning Cao     | Manager | 2      | 2      | IoT      |

# LEFT JOIN

- Join two tables together based on some condition.
- Results are returned for **every row** in the table on the **left** of the join.
- If there is no match for a row in the right table, then the columns all show the value **NULL**.

# LEFT JOIN

employee

| empno | name         | job     | deptno |
|-------|--------------|---------|--------|
| 1     | Sean Russell | Teacher | 1      |
| 2     | Ruihai Dong  | Teacher | 1      |
| 3     | Lina Xu      | Teacher | NULL   |
| 4     | Ning Cao     | Manager | 2      |

department

| deptno | deptname |
|--------|----------|
| 1      | CS       |
| 2      | IoT      |
| 3      | SE       |

- Find the details of the employees and departments including those whose deptno details is not available:
- `SELECT * FROM employee AS e LEFT JOIN department AS d ON e.deptno = d.deptno;`

| empno | name         | job     | deptno | deptno | deptname |
|-------|--------------|---------|--------|--------|----------|
| 1     | Sean Russell | Teacher | 1      | 1      | CS       |
| 2     | Ruihai Dong  | Teacher | 1      | 1      | CS       |
| 4     | Ning Cao     | Manager | 2      | 2      | IoT      |
| 3     | Lina Xu      | Teacher | NULL   | NULL   | NULL     |

# RIGHT JOIN

- Join two tables together based on some condition
- Results are returned for **every row** in the table on the **right** of the join.
- If there is no match for a row in the left table, then the columns all show the value **NULL**.



# RIGHT JOIN

employee

| empno | name         | job     | deptno |
|-------|--------------|---------|--------|
| 1     | Sean Russell | Teacher | 1      |
| 2     | Ruihai Dong  | Teacher | 1      |
| 3     | Lina Xu      | Teacher | NULL   |
| 4     | Ning Cao     | Manager | 2      |

department

| deptno | deptname |
|--------|----------|
| 1      | CS       |
| 2      | IoT      |
| 3      | SE       |

- Find the details of the employees and departments including those where no employees work:
- `SELECT * FROM employee AS e RIGHT JOIN department AS d ON e.deptno = d.deptno;`

| empno | name         | job     | deptno | deptno | deptname |
|-------|--------------|---------|--------|--------|----------|
| 1     | Sean Russell | Teacher | 1      | 1      | CS       |
| 2     | Ruihai Dong  | Teacher | 1      | 1      | CS       |
| 4     | Ning Cao     | Manager | 2      | 2      | IoT      |
| NULL  | NULL         | NULL    | NULL   | 3      | SE       |

# FULL JOIN

- Join two tables together based on some condition.
- Results are returned for **every row** in **both tables**.
- If there is no match for a row from the right table in the left table, then the columns all show the value **NULL**.
- If there is no match for a row from the left table in the right table, then the columns all show the value **NULL**.

# FULL JOIN

employee

| empno | name         | job     | deptno |
|-------|--------------|---------|--------|
| 1     | Sean Russell | Teacher | 1      |
| 2     | Ruihai Dong  | Teacher | 1      |
| 3     | Lina Xu      | Teacher | NULL   |
| 4     | Ning Cao     | Manager | 2      |

department

| deptno | deptname |
|--------|----------|
| 1      | CS       |
| 2      | IoT      |
| 3      | SE       |

- Find the details of the employees and the department in which they work (including those where the office details is not available and departments with no employees):
- `SELECT * FROM employee AS e FULL JOIN department AS d ON e.deptno = d.deptno;`

| empno | name         | job     | deptno | deptno | deptname |
|-------|--------------|---------|--------|--------|----------|
| 1     | Sean Russell | Teacher | 1      | 1      | CS       |
| 2     | Ruihai Dong  | Teacher | 1      | 1      | CS       |
| 4     | Ning Cao     | Manager | 2      | 2      | IoT      |
| 3     | Lina Xu      | Teacher | NULL   | NULL   | NULL     |
| NULL  | NULL         | NULL    | NULL   | 3      | SE       |

**NOTE:**  
MySQL does  
not support  
FULL JOIN

# USING: a shorter way to join

- If we want to join two tables and the join condition is that two attributes with the **same name** should be equal, there is a shorter way to write it.
- Example (normal way):
  - `SELECT * FROM employee AS e INNER JOIN department AS d ON e.deptno=d.deptno;`
- Example (USING):
  - `SELECT * FROM employee AS e INNER JOIN department AS d USING(deptno);`