COM106: Introduction to Databases

SQL – Queries on a Single Table

SQL – Queries on a Single Table

So far, we've seen a number of **CRUD** operations (**C** – INSERT, **U** – UPDATE & **D** – DELETE)

Now look at **R** operations – the **SELECT** statement – used to retrieve data/information from the database

The general form of the SELECT statement is as follows:

SELECT *<attribute list>*
FROM *<table list>*
WHERE *<selection clause>*
*[**AND/OR/NOT** <selection clause>….]*
GROUP BY *<attribute list>* ***[grouping]***
HAVING *<aggregate function selection>* ***[between groups]***
ORDER BY *<attribute list>* ***[sorting]***

For example, the natural language query: ***Get the names and salaries of all employees who are on the first floor***

can be answered using SQL query:

```
SELECT ename, salary
FROM EMPLOYEE
WHERE floor = 1 ;
```

SELECT always returns a table
- even if a single value is returned..

May want information returned from more than one table
List all tables used in the query.

SQL – Queries on a Single Table

In the examples that follow the EMPLOYEE, PROJECT, WORKS_ON database is used -

with relational schema

and sample data as shown

EMPLOYEE (enum, ename, salary, floor)
PROJECT (pnum, pname, leader)
WORKS_ON (enum*, pnum*, role)

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Jones	19000	3
852407	Brown	16000	3
852455	White	25100	2
852491	Adams	30500	1
852514	Doyle	11650	2
852530	Evans	26980	4

PROJECT

pnum	pname	leader
121	IT	Gates
135	Design	Sinclair
147	Analysis	Einstein
216	Publicity	Saatchi
227	Theatre	Dench
251	Sport	Shearer

WORKS_ON

enum	pnum	role
852341	121	Manager
852341	135	Designer
852358	147	Consultant
852358	135	Consultant
852407	216	Assistant
852455	121	Assistant
852455	227	Manager
852491	135	Designer
852491	216	Manager
852514	121	Assistant
852514	216	Consultant
852514	251	Manager
852530	147	Manager

SQL – Queries on a Single Table

How to Construct an SQL SELECT Query

Need to interpret the meaning of the **natural language query** (sentence) to find:

1. What columns do we want to see in the query result table?
   (Which **attributes** (columns) go into the **SELECT** clause?)
2. What other **attributes** are involved in the query
   (for selecting in a **WHERE/AND/OR/NOT** clause)
   or grouping **(GROUP BY** clause)
   or choosing between groups (**HAVING** clause)
   or sorting (**ORDER BY** clause)
3. What table(s) are required to get all the attributes involved in the **SELECT** clause or from the **WHERE**, **GROUP BY**, **HAVING** or **ORDER BY** clauses
   (all the tables used will be included in the **FROM** clause)

For example: ***Get the names and salaries of all employees who are on the first floor***

1. Attributes in **SELECT** clause:
   We want the names of the employees (**ename**)
   and the salaries of the employees (**salary**)
2. The **WHERE** clause attributes:
   We need to find employees on the first floor (**floor = 1**)
3. Table(s) with required attributes in **FROM** clause:
   ename, **salary** and **floor** all come from the **EMPLOYEE** table

SQL – Queries on a Single Table

So, the **natural language** query: ***Get the names and salaries of all employees who are on the first floor***

can be answered using an **SQL** query:

```
SELECT ename, salary
FROM EMPLOYEE
WHERE floor = 1 ;
```

Names and salaries.

All attributes are in the EMPLOYEE table.

On the first floor.

EMPLOYEE

enum	ename	salary	floor
852341	***Smith***	***15000***	***1***
852358	Jones	19000	3
852407	Brown	16000	3
852455	White	25100	2
852491	***Adams***	***30500***	***1***
852514	Doyle	11650	2
852530	Evans	26980	4

Result Table

ename	salary
Smith	15000
Adams	30500

Get the employee details of all employees on the first floor

```
SELECT *
FROM EMPLOYEE
WHERE floor = 1 ;
```

** - all attributes*

Result Table

enum	ename	salary	floor
852341	Smith	15000	1
852491	Adams	30500	1

SELECT *

is equivalent to

SELECT enum, ename, salary, floor

SQL – Queries on a Single Table

WHERE Clauses with an AND

Get the employee numbers for employees in project 121 who have an Assistant role

```
SELECT enum
FROM WORKS_ON
WHERE pnum = 121
AND role = 'Assistant' ;
```

Result Table

enum
852455
852514

WORKS_ON

enum	pnum	role
852341	121	Manager
852341	135	Designer
852358	147	Consultant
852358	135	Consultant
852407	216	Assistant
852455	***121***	***Assistant***
852455	227	Manager
852491	135	Designer
852491	216	Manager
852514	***121***	***Assistant***
852514	216	Consultant
852514	251	Manager
852530	147	Manager

Alternative WHERE Clauses with an OR

Get the name of employees working on the 1st or 2nd floor

```
SELECT ename
FROM EMPLOYEE
WHERE floor = 1
OR floor = 2 ;
```

EMPLOYEE

enum	ename	salary	floor
852341	***Smith***	***15000***	***1***
852358	Jones	19000	3
852407	Brown	16000	3
852455	***White***	***25100***	***2***
852491	***Adams***	***30500***	***1***
852514	***Doyle***	***11650***	***2***
852530	Evans	26980	4

Result Table

ename
Smith
White
Adams
Doyle

SQL – Queries on a Single Table

Other Useful Operators with the WHERE Clause

Using the **IN** Operator to Specify Multiple Values in a WHERE Clause.

```
SELECT ename
FROM EMPLOYEE
WHERE floor IN (1, 2);
```

Using the **BETWEEN** Operator to Select Values within a Range

```
SELECT ename
FROM EMPLOYEE
WHERE ename BETWEEN 'Adams' AND 'Evans';
```

values can be numbers, text, or dates.

The **NOT** Operator

Get the names of employees on all floor, except floors 1 and 3

```
SELECT ename
FROM EMPLOYEE
WHERE floor NOT IN (1, 3);
```

Combining Operators in the WHERE Clause.

Operators can be combined to form more complex queries, using parenthesis as necessary to establish precedence.

Get details of all projects between project number 121 and 227, except those led by Sinclair or Saatchi

```
SELECT *
FROM PROJECT
WHERE pnum BETWEEN 121 AND 227
AND leader NOT IN ('Sinclair', 'Saatchi');
```

SQL – Queries on a Single Table

Aggregate Functions

SQL provides a number of built-in aggregate functions that return a **single** value, calculated from values in a column.

SUM( )	Returns the Total Value
AVG( )	Returns the Average Value
MAX( )	Returns the Highest Value
MIN( )	Returns the Lowest Value
COUNT( )	Returns the number of tuples (rows) in result table

Operate only on numeric attributes

Aggregate functions can **ONLY** appear in **SELECT** and **HAVING** clauses

What is the total salary of all employees?

```
SELECT SUM (salary)
FROM EMPLOYEE;
```

Result
144230

Aggregate functions normally appear alone in a SELECT clause (except for the special case where a GROUP BY clause is present)

What is the average salary for employees on the 3rd floor?

```
SELECT AVG (salary)
FROM EMPLOYEE
WHERE floor = 3 ;
```

Result
20604

** - can be used ONLY with COUNT*
*COUNT using PK or * - other attributes may be NULL.*

How many employees are on the 1st floor?

```
SELECT COUNT (*)
FROM EMPLOYEE
WHERE floor = 1;
```

or

```
SELECT COUNT (enum)
FROM EMPLOYEE
WHERE floor = 1;
```

Result
2

SQL – Queries on a Single Table

The ORDER BY Clause

Uses options **ASC** - ascending or **DESC** - descending

e.g. ***List the employee names in alphabetic order***

```
SELECT ename
FROM EMPLOYEE
ORDER BY ename ASC;
```

ename
Adams
Brown
Doyle
Evans
Jones
Smith
White

***ASC** is the **default** value and thus not strictly necessary in this example*

The DISTINCT Predicate

Eliminates duplicate tuples (rows)

e.g. ***Get a list of roles worked on (without repeat values)***

```
SELECT DISTINCT role
FROM WORKS_ON;
```

role
Manager
Designer
Consultant
Assistant
Administrator

DISTINCT clauses are most useful when used with an attribute which does NOT have unique values and the duplicated result rows are not required

Without DISTINCT, the query would return multiple entries for each role – as per the original WORKS_ON table

SQL – Queries on a Single Table

The GROUP BY Clause

Groups result table by attribute(s).

GROUP BY clauses group the data **by value** using a field that **MUST** therefore have duplicate values so that at least some groups have multiple records.

An aggregate function can then be applied to each group.

All simple attributes appearing in the SELECT clause **MUST** appear in the GROUP BY clause

e.g. ***Get the average salary by floor of all employees***

```
SELECT floor, AVG (salary)
FROM EMPLOYEE
GROUP BY floor ;
```

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Jones	19000	3
852407	Brown	16000	3
852455	White	25100	2
852491	Adams	30500	1
852514	Doyle	11650	2
852530	Evans	26980	4

Result Table

floor	AVG (salary)
1	22750
2	18375
3	17500
4	26980

SQL – Queries on a Single Table

When GROUP BY clauses are being used:

Can't use a normal attribute in the GROUP BY clause that **does not already appear** in the SELECT clause

Don't use more than one attribute in the GROUP BY clause (unless you intend to use grouping and subgrouping – not covered in these notes)

The attribute you group by should normally be **repeated** in the SELECT clause

The HAVING Clause

Can **ONLY** be used when a **GROUP BY** clause is present - **used to choose between groups**.

Can **ONLY** contain expressions using **aggregate functions**

e.g. ***Get the floors having two or more employees***

```
SELECT floor
FROM EMPLOYEE
GROUP BY floor
HAVING COUNT(*) >= 2 ;
```

Result Table

floor
1
2
3

HAVING clauses are used to choose **between** groups and **cannot** appear **without** a preceding GROUP BY clause.

HAVING clauses **MUST** be constructed as:

Aggregate function	*compared with*	*a value*
count( ), avg( ), etc	**=, <, >, etc**	**number**