COM106: Introduction to Databases

SQL – Multiple Table Queries

SQL – Multiple Table Queries

So far, we've looked at queries where the information required can be found in a **single table** - often, the information required is spread across **two or more tables**.

For example - ***Get the names of employees and the roles they have performed for employees on the 1st floor.***

ename *and* **floor** *are in the* **EMPLOYEE** *table*

role *is in the* **WORKS_ON** *table*

Retrieving data from two or more tables requires **JOINS** (normally **across PK/FK matches**)

There are **TWO** SQL approaches

Original (**Mark 1**): specify join clause as an extra **WHERE/AND** clause

New (**Mark 2**): specify join clause in a subsidiary **ON** clause after FROM clause

For standard SQL, **JOINS** using a query block require that:

1. all attributes occurring in two tables are referred to as <relation>.<attribute>
   e.g. EMPLOYEE.enum
2. all relations (tables) participating in the join are listed in the FROM clause
3. join clauses must be inserted as **WHERE/AND** clause(s)
   e.g. WHERE EMPLOYEE.enum = WORKS_ON.enum

SQL – Multiple Table Queries

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 – Multiple Table Queries

Natural Language Query: ***Get the names of employees and the roles they have performed for employees on the 1st floor.***

```
SELECT ename, role
FROM EMPLOYEE, WORKS_ON
WHERE EMPLOYEE.enum = WORKS_ON.enum
AND floor = 1;
```

Mark 1 Join

*Join on PK/FK match – **EMPLOYEE.enum (PK)** and **WORKS_ON.enum (FK)***

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	role
Smith	Manager
Smith	Designer
Adams	Designer
Adams	Manager

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 – Multiple Table Queries

Natural Language Query: ***Get the names and salaries of employees who work on projects led by 'Gates'***

*Requires attributes from all **three** tables*

```
SELECT ename, salary
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE EMPLOYEE.enum = WORKS_ON.enum
AND WORKS_ON.pnum = PROJECT.pnum
AND leader = 'Gates';
```

*Mark 1 Join on **PK/FK match***

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

Result Table

ename	salary
Smith	15000
White	25100
Doyle	11650

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

WORKS_ON.*pnum* = PROJECT.*pnum*

SQL – Multiple Table Queries

Mark 2 Joins - Alternative method for SQL joins

1. all relations participating in the join are listed in the FROM clause
2. Use of **INNER JOIN** as the join operator
3. join clauses must be specified in a subsidiary **ON** clause

*There are a number of different **JOIN** operators – see later*

e.g. ***Get the names of employees and the roles they have performed for employees on the 1st floor.***

Mark 1

```
SELECT ename, role
FROM EMPLOYEE, WORKS_ON
WHERE EMPLOYEE.enum = WORKS_ON.enum
AND floor = 1;
```

Mark 2

```
SELECT ename, role
FROM EMPLOYEE INNER JOIN WORKS_ON
ON EMPLOYEE.enum = WORKS_ON.enum
WHERE floor = 1;
```

e.g. ***Get the names and salaries of employees who work on projects led by 'Gates'***

Mark 1

```
SELECT ename, salary
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE EMPLOYEE.enum = WORKS_ON.enum
AND WORKS_ON.pnum = PROJECT.pnum
AND leader = 'Gates';
```

Mark 2

```
SELECT ename, salary
FROM EMPLOYEE INNER JOIN WORKS_ON
ON EMPLOYEE.enum = WORKS_ON.enum
INNER JOIN PROJECT
ON WORKS_ON.pnum = PROJECT.pnum
WHERE leader = 'Gates';
```

SQL – Multiple Table Queries

Nested Queries

Join queries can be formulated as a set of nested queries whenever:

- the **SELECT** clause contains attributes from only **one** relation
- extra information is required **for comparison** in the **WHERE** clause
  e.g. an aggregate function, or a list of attributes from another table

Links between the query blocks can be achieved:

- using the **IN predicate** if multiple values may be returned from the lower clause
- using >,>=,<,<=,=,<> whenever only one value will be returned from the lower clause

e.g. ***Get the names of employees who have worked in the role of 'Consultant'.***

As Mark 1 Join:

```
SELECT ename
FROM EMPLOYEE, WORKS_ON
WHERE EMPLOYEE.enum = WORKS_ON.enum
AND role = 'Consultant';
```

Result Table

ename
Jones
Jones
Doyle

*The nested **SELECT** clause returns a **list of enum** of staff who are consultants*

As a Nested Query:

```
SELECT ename
FROM EMPLOYEE
WHERE enum IN
      (SELECT enum
      FROM WORKS_ON
      WHERE role = 'Consultant');
```

So, WHERE clause is equivalent to:

WHERE enum IN (852358, 852514);

SQL – Multiple Table Queries

e.g. ***Get the employee names for employees earning salaries above the average salary for all employees***

```
SELECT ename
FROM EMPLOYEE
WHERE salary >
        (SELECT AVG (salary)
        FROM EMPLOYEE);
```

Result Table

ename
White
Adams
Evans

*The nested **SELECT** clause returns a **single value** – average salary*

***RECALL:** Aggregate functions can only appear in **SELECT** or **HAVING** clauses*

e.g. ***How many projects have less than 2 people working on them?***

```
SELECT COUNT (pnum) AS Num_Projects
FROM PROJECT
WHERE pnum IN
        (SELECT pnum
        FROM WORKS_ON
        GROUP BY pnum
        HAVING COUNT(enum) < 2);
```

Result Table

Num_Projects
2

*The subclause **AS <columnname>**, where columnname contains **NO** spaces, is used to give the result table a meaningful name*

The query is evaluated **backwards** as:

subquery: get the **pnums** of those projects having less than 2 employees and pass them upwards into the top query

top query: get the number of projects (pnums) passed into the **IN** list

SQL – Multiple Table Queries

Self Joins

In some cases a table needs to be **joined to itself**, particularly if extra information is stored in the table but not explicitly available by a normal query.

e.g. ***Find all pairs of employees who work on the same floor as each other.***

```
SELECT EMPLOYEE.ename, TEMP.ename
FROM EMPLOYEE, EMPLOYEE AS TEMP
WHERE EMPLOYEE.floor = TEMP.floor
AND EMPLOYEE.enum < TEMP.enum ;
```

This is an example of a **SELF JOIN** whereby a table is joined to itself.

Result Table

ename	ename
Smith	Adams
Jones	Brown
White	Doyle

To do this one copy of the table is given the **alias TEMP** (signified as **TABLENAME AS ALIAS**)

Joins are performed where the floor numbers are the same:

EMPLOYEE.floor = TEMP.floor

the clause:

EMPLOYEE.enum < TEMP.enum

eliminates an employee being shown as on the **same floor as himself** and **duplicate pairs** (e.g. either Adams, Smith OR Smith, Adams) are eliminated

EMPLOYEE

enum	ename	floor
852341	Smith	*1*
852358	Jones	*3*
852407	Brown	3
852455	White	*2*
852491	Adams	1
852514	Doyle	2
852530	Evans	4

TEMP

floor	enum	ename
1	852341	Smith
3	852358	Jones
3	852407	Brown
2	852455	White
1	852491	Adams
2	852514	Doyle
4	852530	Evans

SQL – Multiple Table Queries

Nested Queries with Negation

e.g. ***Get all details of employees who do not work on project 121.***

```
SELECT *
FROM EMPLOYEE
WHERE enum NOT IN
        (SELECT enum
        FROM WORKS_ON
        WHERE pnum = 121);
```

The query is evaluated **backwards** as:

subquery: pass the **enums** of those employees working on project 121 to top query

top query: get the details of the employees **NOT IN** that list of **enums**

e.g. ***Get the names of employees working on projects that have no consultants working on them***

OR ***Get the names of employees NOT working on projects that have a consultant on them***

Sometimes, it is helpful to **reframe** the query.

It can also be useful to build the query from the **bottom up.**

lowest subquery: Get the projects which have consultants working on them

middle subquery: Get the employee numbers for projects not in this list

top query: Get the employee names of employees from this employee list

SQL – Multiple Table Queries

Get the names of employees NOT working on projects that have a consultant on them

lowest subquery: Get the projects which have consultants working on them

```
SELECT pnum
FROM WORKS_ON
WHERE role = 'Consultant'
```

middle subquery: Get the employee numbers for projects not in this list

```
SELECT enum
FROM  WORKS_ON
WHERE pnum NOT IN
        (SELECT pnum
        FROM WORKS_ON
        WHERE role = 'Consultant')
```

top query: Get the employee names of employees from this employee list

```
SELECT DISTINCT ename
FROM EMPLOYEE
WHERE enum IN
        (SELECT enum
        FROM  WORKS_ON
        WHERE pnum NOT IN
                (SELECT pnum
                FROM WORKS_ON
                WHERE role = 'Consultant')) ;
```

Result Table

ename
Smith
White
Doyle

SQL – Multiple Table Queries

Views

A view is a **virtual table** defined as a subset of one or more tables and stored in the data dictionary

It is only materialized (actually created as a table) at run time from the base table(s)

Views can be manipulated mostly as though it were a base table (from user perspective)

SQL View Syntax:

```
CREATE VIEW viewname  (<new attribute list>)
AS SELECT <attribute list>
    FROM <table list>
    [ other clauses];
```

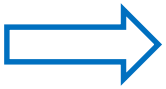
Can give the attributes new names

e.g.
```
CREATE VIEW FLOOR_1_EMP (employee_no, name)
AS SELECT enum, ename
    FROM EMPLOYEE
    WHERE floor = 1;
```

Views can be used in queries, just like any other table

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



FLOOR_1_EMP

employee_no	name
852341	Smith
852491	Adams