



# **DATABASE SYSTEMS**

*(Part 2 of 2: SQL)*

**CT102:**  
**Information**  
**Systems**

# DATABASE LANGUAGES

The programming language for Relational Databases is called SQL - Structured Query Language

SQL is a **standardised** Query language across all relational DBMS (with some minor variations):

- First version SQL-89
- SQL-92 (SQL-2)
- SQL-99 (SQL-3)
- Recent standards include XML-related features

Standardised by American National Standards Institute (**ANSI**) and International Standards Organization (**ISO**)

# SQL

- SQL is a **declarative language**
- It allows you specify the results you require ... not the order of the operations to retrieve those results
- In comparison, C, C++, Java, Python are considered **Imperative Languages** ... which facilitate computation by means of state changes, e.g., can specify

```
int a;  
a = 3;
```

SQL allows you to create tables and links between tables, manipulate and query data (CRUD operations) and specify privileges.

# SQL QUERIES

Allows for specification of *queries*

Queries represent *information needs*

Queries can be run to produce results

Result might be:

- Output to user
- Modification of Data in Database
- CRUD operations: **C**reate **R**ead **U**ppdate **D**eleate

# INSERT STATEMENT

The INSERT statement allows data to be inserted as part of a query (rather than via the graphical user interface (GUI))

General format is:

```
INSERT INTO table (<attribute list> )  
VALUES (<value list>);
```

appointments	AddressBook							
FullName	HseNum	Address1	Address2	County	Country	HomePh	MobPh	

## EXAMPLE 1

Add a new tuple to the AddressBook table for name 'Ann Lawlor' and house number (HseNum) 12

```
INSERT INTO AddressBook
    (FullName, HseNum)
VALUES    ('Ann Lawlor', 12 );
```

**Note:** If primary key exists, must specify it for any INSERT statement, e.g. if mobile phone is Primary Key

## WITH PRIMARY KEY ...

appointments		AddressBook					
FullName	HseNum	Address1	Address2	County	Country	HomePh	MobPh

Add a new tuple to the AddressBook table for name 'Ann Lawlor' and house number (HseNum) 12 with mobile phone, '086858585'

```
INSERT INTO AddressBook
```

```
(FullName, HseNum, MobPh)
```

```
VALUES ('Ann Lawlor', 12, '086858585');
```

# UPDATE

Can modify one or more records

General format is:

**UPDATE** *table*

**SET**        *<attribute name> = <some value>*

**WHERE**    *<condition>;*



## EXAMPLE 2

Update the house number of Peter Smith in the  
AddressBook Table to 90

```
UPDATE AddressBook
SET      HseNum = 90
WHERE    FullName = 'Peter Smith';
```

# DELETE

The DELETE statement does not remove the table structure (e.g. attributes), only the data in the tables

General format:

```
DELETE *  
FROM table  
WHERE condition;
```

## EXAMPLE 3

Delete appointment number 8 from the table appointments:

```
DELETE *  
FROM   appointments  
WHERE  id = 8;
```

## MORE EXAMPLES:

### Example 4: for school table

Using `INSERT`, insert a new tuple into the school table for student “R. Sandip” with ID 181111 and Code GY350 and modCode ‘CT441’

```
INSERT INTO School (ID, Sname, Code,  
modCode)
```

```
VALUES (181111, 'R. Sandip', 'GY350',  
'CT441') ;
```

School									
ID	SName	Code	ModName	Lecturer	Location	Grade	ModCode	Yr	

## EXAMPLE 5: again with school table:

School									
ID	SName	Code	ModName	Lecturer	Location	Grade	ModCode	Yr	

Using UPDATE, *change the grade* for student with ID 21112 and modcode MA160 from “B” to “A”

Note: Boolean AND is written “AND” in SQL

```
UPDATE School
SET      Grade = 'A'
WHERE    ID = 21112 AND
         modcode = 'MA160' AND
         Grade = 'B';
```

## Example 6 with school table

Using DELETE, *delete student* “A. Alabbad”, with ID 20343

```
DELETE *  
FROM School  
WHERE ID = 20343
```

School									
ID	SName	Code	ModName	Lecturer	Location	Grade	ModCode	Yr	

# Read using SQL `SELECT` statement

Most important and often-used query is that of **selecting** tuples (rows) from a table (or multiple tables) that satisfy some condition

`SELECT` statement allows this

Has 6 possible “clauses”, we will consider the first 3:

**`SELECT`** [`DISTINCT`] *<attribute list>*

**`FROM`** *<table list>*

**`WHERE`** *<condition>*

# Examples using addressbook table

appointments		AddressBook						
FullName	HseNum	Address1	Address2	County	Country	HomePh	MobPh	

7 Using the original table 1, write a query to find the names and mobile phone numbers of all people in Galway.

```
SELECT  FullName, MobPh
FROM    AddressBook
WHERE   county = 'Galway';
```



appointments		AddressBook					
FullName	HseNum	Address1	Address2	County	Country	HomePh	MobPh

**8** Using the original table 1, write a query to find the name of the person with mobile phone number '087 123456'

```

SELECT  FullName
FROM    AddressBook
WHERE   MobPh = '087123456'

```

## Example using the appointments table

9 Using the appointments table, write a query to find the names and date of all appointments for the consultant “Dr Garvey”

appointments						
ID	PatientName	BirthYear	ConsultantName	Room	Speciality	AptDate

```
SELECT PatientName, AptDate
FROM appointments
WHERE ConsultantName = 'Dr Garvey';
```

# QUERYING ACROSS MULTIPLE TABLES

- A number of different approaches can be used if query needs to select data from multiple tables.
- The query becomes more complex. One approach is use two queries – an outer and a sub-query.
- If the subquery returns a single number then can connect the two with a simple mathematical operator such as  $=$ ,  $\neq$ ,  $>$ ,  $<$ , etc.
- If the subquery returns a single string then can connect the two with a string comparison using an operator such as  $=$ ,  $\neq$

## EXAMPLE 10:

appointments						
ID	PatientName	BirthYear	ConsultantName	Room	Speciality	AptDate

Assume you are given the following three tables:

patient(pID, pName, BirthYear)

counsultant(cID, cName, room,  
speciality)

appointments(ID, pID, cID, AptDate)

Find what room Ali Byrne should attend for  
appointments

patient(pid, pname, birthyear)  
counsellant(cid, cname, room, speciality)  
appointments(id, pid, cid, aptdate)

```
SELECT  room
FROM    counsellant
WHERE   cid IN
        (SELECT cid
         FROM  appointment
         WHERE pid =
              (SELECT pid
               FROM  patient
               WHERE pname = 'Ali Byrne')
        );
```

# What does the query look like using the original appointments table?

appointments						
ID	PatientName	BirthYear	ConsultantName	Room	Speciality	AptDate

```
SELECT    room
FROM      appointments
WHERE     PatientName = 'Ali Byrne';
```

## EXAMPLE 11: USING SCHOOL TABLE

Using the school table, write a query to find the names of all students with an “A” grade in the subject with name ‘Mathematics’

<div>appointments AddressBook School</div>									
ID	SName	Code	Yr	ModName	ModCod	Lecturer	Location	Grade	

```
SELECT  SName
FROM    School
WHERE   grade = 'A' AND ModName = 'Mathematics';
```

# USING AGGREGATE FUNCTIONS

SQL supports a number of aggregate functions which can be used in the SELECT clause

Examples include:

- **SUM, AVG, MIN, MAX** applied to numeric fields
- **COUNT** returns the number of tuples/values specified in a query



## EXAMPLE 12

Using the school table, write a query to find *how many people received an “A” grade across all subjects*

<div>appointments AddressBook School</div>									
ID	SName	Code	Yr	ModName	ModCod	Lecturer	Location	Grade	

```
SELECT  COUNT(Sname)
FROM    School
WHERE   grade = 'A';
```

## EXAMPLE 13

Using the appointments table, (and using a subquery) write a query to find the *youngest person* who has an appointment

aptID ▾	PatientName ▾	DateOfBirth ▾	ConsultantName ▾	Consult Room ▾	Consult Area ▾	AptDate
---------	---------------	---------------	------------------	----------------	----------------	---------

```
SELECT PatientName
FROM appointments
WHERE DateOfBirth =
    (SELECT MAX(DateOfBirth)
     FROM appointments);
```

## EXAMPLE 14: LOOKING AT 2 NEW TABLES:

**employees**(employeeNumber, lastName, firstName, extension, email, officeCode, reportsTo, jobTitle)

**offices**(officeCode, city, phone, addressLine1, addressLine2, state, country, postalCode, territory)

# LOOKING AT THE DATA TYPES

**employees**(employeeNumber, lastName, firstName, extension, email, officeCode, reportsTo, jobTitle)

**offices**(officeCode, city, phone, addressLine1, addressLine2, state, country, postalCode, territory)

Column	Type
<b>officeCode</b>	varchar(10)
<b>city</b>	varchar(50)
<b>phone</b>	varchar(50)
<b>addressLine1</b>	varchar(50)
<b>addressLine2</b>	varchar(50) <i>NULL</i>
<b>state</b>	varchar(50) <i>NULL</i>
<b>country</b>	varchar(50)
<b>postalCode</b>	varchar(15)
<b>territory</b>	varchar(10)

Column	Type
<b>employeeNumber</b>	int(11)
<b>lastName</b>	varchar(50)
<b>firstName</b>	varchar(50)
<b>extension</b>	varchar(10)
<b>email</b>	varchar(100)
<b>officeCode</b>	varchar(10)
<b>reportsTo</b>	int(11) <i>NULL</i>
<b>jobTitle</b>	varchar(50)

## EXAMPLE 14 QUESTIONS: Write SELECT statements to find the following answers:

14.1 Find all the countries where there are offices.

14.2 Find all the employees (their names) with job Title “Sales Rep”.

14.3 Find the cities in country “USA” where there are offices.

14.4 Find the email address of employee “Julie Firrelli”.

14.5 Find the postcode of the Paris office.

# SOLUTIONS

-- 14.1

```
SELECT DISTINCT country  
FROM offices
```

-- 14.2

```
SELECT firstName, lastName  
FROM employees  
WHERE jobTitle = 'Sales Rep';
```

# SOLUTIONS *ctd.*

-- 14.3

SELECT city

FROM offices

WHERE country = 'USA';

-- 14.4

SELECT email

FROM employees

WHERE firstName = 'Julie' AND lastName = 'Firrelli';

# SOLUTIONS *ctd.*

-- 14.5

SELECT postalCode

FROM offices

WHERE city = 'Paris';



# DATABASE SYSTEM SUMMARY PART 2

A database requires some data access method in order to query and modify data - SQL is the programming language for Relational Databases

Many other languages for structured data are similar to SQL

SQL SELECT statement: 3 clauses we considered:

*SELECT FROM WHERE with 1 table only*

*Also: MIN, MAX, AVG, SUM, COUNT()*

SQL INSERT INTO, UPDATE, DELETE *on 1 table only*