UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Information Storage and Management I

Dr. Alejandro Arbelaez

Dr. Alejandro Arbelaez

- Lecturer – Computer Science Department
- Experience:
  - Assistant Lecturer – CIT, 2017-2019
  - Postdoctoral Researcher – University College Cork (Insight Centre), 2013 – 2016
  - Postdoctoral Researcher – University of Tokyo, Japan, 2011 – 2013
  - PhD. In Computer Science – Université Paris XI, France, 2007- 2011
  - Engineering degree in Computer Science – Xaveriana University, Colombia, 2006

✉ a.arbelaez@cs.ucc.ie

Dr. Alejandro Arbelaez

Dr. Alejandro Arbelaez

- 2019 – Present: Lecturer at UCC
- 2017 – 2019: Lecturer at CIT
  - Teaching mainly in the MSc in AI, MSc in Data Analytics, and Software Development programme
- 2013 – 2017: Senior Postdoc at University College Cork
  - Working in Data Science and Analytics projects
- 2011- 2013: Postdoc at University of Tokyo / Franco-Japanese Research Lab
  - Working in AI and Massively parallel computing
- 2007 - 2011 : PhD Candidate at Université Paris XI
  - Working in the interception between descriptive predictive and prescriptive analytics

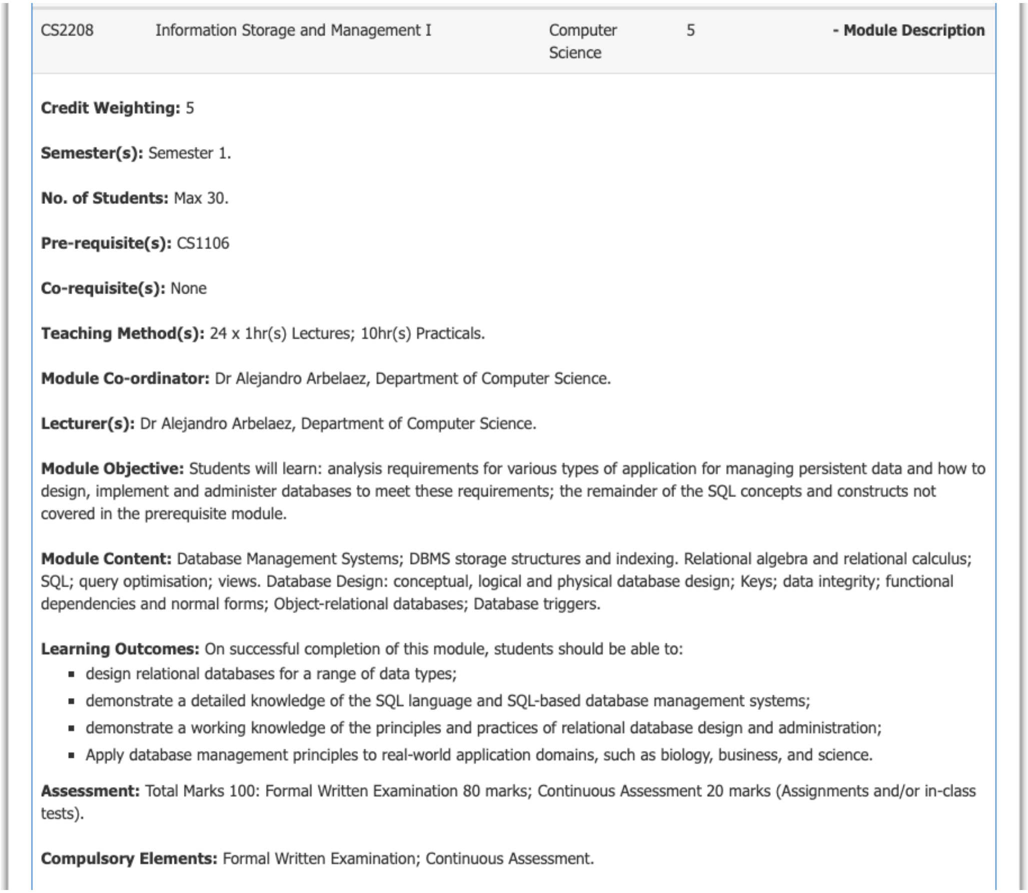
Learning Outcomes

Module Description

- Design relational databases for a range of data types;
- Demonstrate a detailed knowledge of the SQL language and SQL-based database management systems;
- Demonstrate a working knowledge of the principles and practices of relational database design and administration;
- Apply database management principles to real-world application domains, such as biology, business, and science.

Module Description

CS2208



CS2208	Information Storage and Management I	Computer Science	5	**- Module Description**

Credit Weighting: 5

Semester(s): Semester 1.

No. of Students: Max 30.

Pre-requisite(s): CS1106

Co-requisite(s): None

Teaching Method(s): 24 x 1hr(s) Lectures; 10hr(s) Practicals.

Module Co-ordinator: Dr Alejandro Arbelaez, Department of Computer Science.

Lecturer(s): Dr Alejandro Arbelaez, Department of Computer Science.

Module Objective: Students will learn: analysis requirements for various types of application for managing persistent data and how to design, implement and administer databases to meet these requirements; the remainder of the SQL concepts and constructs not covered in the prerequisite module.

Module Content: Database Management Systems; DBMS storage structures and indexing. Relational algebra and relational calculus; SQL; query optimisation; views. Database Design: conceptual, logical and physical database design; Keys; data integrity; functional dependencies and normal forms; Object-relational databases; Database triggers.

Learning Outcomes: On successful completion of this module, students should be able to:

- design relational databases for a range of data types;
- demonstrate a detailed knowledge of the SQL language and SQL-based database management systems;
- demonstrate a working knowledge of the principles and practices of relational database design and administration;
- Apply database management principles to real-world application domains, such as biology, business, and science.

Assessment: Total Marks 100: Formal Written Examination 80 marks; Continuous Assessment 20 marks (Assignments and/or in-class tests).

Compulsory Elements: Formal Written Examination; Continuous Assessment.

Module workload

- 2 hours lecture every week (24 hours)
- 1 hours weekly Lab → labs commence in week 2 (next week - Thursdays) - **G.24** & G.21

Assessment Breakdown

- This module includes a combination of Continuous Assessment and a Written Exam
  - Continuous Assessment: 20% -- Assignments and/or in-class tests
  - Written Exam: 80%

Plagiarism

1. Plagiarism is presenting someone else’s work as your own. It is a violation of UCC Policy and there are strict and severe penalties.
2. You must read and comply with the UCC Policy on Plagiarism www.ucc.ie/en/exams/procedures-regulations/
3. The Policy applies to *all* work submitted, including software.
4. You can expect that your work will be checked for evidence of plagiarism or collusion.
5. In some circumstances it may be acceptable to reuse a small amount of work by others, but *only* if you provide explicit acknowledgement and justification.
6. If in doubt ask your module lecturer *prior* to submission. Better safe than sorry!

This Module

Basic SQL

- Crate tables, basic join, primary keys, etc

DB Modelling

- ER Diagrams, Normal Forms, Relational Algebra/calculus

Advanced SQL

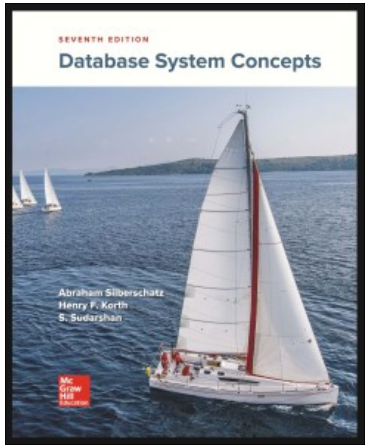
- Indexing, views, triggers, etc.

Resources → Books

GLOBAL EDITION
Database Systems
A Practical Approach to Design, Implementation, and Management
SIXTH EDITION
Thomas Connolly • Carolyn Begg
ALWAYS LEARNING
PEARSON

Carolyn E. Begg and Thomas M. Connolly, **Database Systems A Practical Approach to Design, Implementation, and Management [**ISBN-10: 0321523067]

A. Silberschatz, H. F. Korth, S. Sudarshanm **Database System Concepts**, McGraw-Hill



Databases

- What is a database?
  - A collection of files storing related data
- Example of databases
  - Accounts databases
  - Payroll database
  - UCC’s students database
  - Amazon’s products databases
  - Airline reservation database

Database Management System

- What is a DBMS?
  - A big program written by someone else that allows us to manage efficiently a large database and allows it to persists over long periods of time
- Examples of DBMSs
  - Oracle, IBM DB2, Microsoft SQL Server, etc
  - Open source: MySQL, PostgreSQL, CouchDB
  - Open source library: SQLite
- This semester we will focus on relational DBMS

Example – Online Bookseller

- What data do we need?
  - Data about books, customers, pending orders, order histories, trends performances, etc.
  - Data about sessions (clicks, pages, search history)
  - Note: **data must be persistent**!
  - Also note that data is large.. Won’t fit all in memory
- What capabilities on the data do we need?
  - Insert/remove books, find books by author/title/etc, analyze past order history, recommend books, etc.
  - Data must be accessed **efficiently** by many users
  - Data must be **safe from failures** and malicious users

Using Databases

- Jane and John both have a shared ID number for a gift (credit) of $200 they got as a wedding gift
  - Jane @ her office orders “The selfish Gene”, $80
  - John @ his office orders “Guns the Steel”, $100
- Questions
  - What is the ending credit?
  - What if the second book costs $130?
  - What if the system crashes?
- **A DBMS needs to handle various users issues!**

What functions should a DBMS provide?

1. Describe real-world entities in terms of stored data
2. Persistently store large datasets
3. Efficiently query & update
   - Must handle complex questions about data
   - Must handle sophisticated updates
   - Performance matters
4. Change structure (e.g., add attributes)
5. Concurrency control: enable simultaneous updates
6. Crash recovery
7. Security and integrity

DBMS Benefits

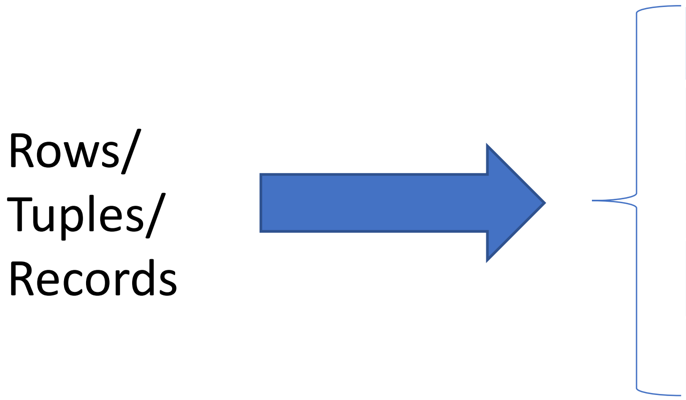
- Expensive to implement all these features inside the application
- DBMS provides these features (and more)
- DBMS simplifies application development

Key Data Management Concepts

- **Data models:** how to describe real-world data
  - Relational, NoSQL, etc..
- **Declarative query languages**
  - Say what you want not how to get it
- **Data independence**
  - Physical independence: can change how data is stored on disk without maintenance to applications
  - Logical independence: can change schema w/o affecting app
- **Query Optimizer**
  - Query plans and how they are executed
- **Physical design**
- **Transactions**
  - Isolation and atomicity

Relational Model

- Data is a collection of relations/tables:



Rows/
Tuples/
Records

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

- Mathematically, relation is a set of tuples
  - Each tuple appears 0 or 1 times in the table
  - Order the rows in unspecified

The Relational Data Model

- Degree or arity of a relation
  - Number of attributes
- Each attribute has a type
  - String: CHAR(20), VARCHAR(50), TEXT
  - Numbers: INT, FLOAT
  - Money, DateTime
  - Few more that are database specific
- **Statically and strictly enforced**

Keys

- An attribute that uniquely identifies a record
- A key can consists of multiple attributes
- Foreign key:
  - A attribute(s) that is a key for other relations

Relational Model: Example

Company(cname, country, no_employees, for_profit)

Country(name, population)

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

name	population
USA	320M
Japan	127M

Query Language

- SQL
  - **S**tructured **Q**uery **L**anguage
  - Developed by IBM in the 70s
  - Most widely used language to query relational data
- We will see other languages for the relational model later on
  - Relational Algebra, Relational Calculus

/ˈsiːkwəl/ “SQL” /ˈɛs kjuː ˈɛl/

1970s

/ˈsiːkwəl/

SEQUEL

Structured **E**nglish **QUE**ry **L**anguage

1970s

REGISTERED
TRADEMARK
REGISTERED

SEQUEL

Structured **E**nglish **QUE**ry **L**anguage

→

SQL

Structured **Q**uery **L**anguage

/ˈsiːkwəl/

/ˈɛs kjuː ˈɛl/

Discussion

- Tables are not ordered
  - They are sets or multisets (bags)
- Tables are flat
  - Not nested attributes
- Tables do not prescribe how they are implemented/stored on disk
  - This is called physical data independence

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

2D Array

```
table=[ ["IBM", "USA", 20000, "True"],
         ["Sony", "Japan", 5000, "True"],
         ["Nintendo", "Japan", 3000, "True"],
         ["AirCanada", "Canada", 5000, "True"]
       ]
```

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

2D Array

```
table=[ ["IBM", "USA", 20000, "True"],
        ["Sony", "Japan", 5000, "True"],
        ["Nintendo", "Japan", 3000, "True"],
        ["AirCanada", "Canada", 5000, "True"]
      ]
```

Row Major Order

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

- What if we store this table in a row major order?
  - What operations we will be able to do efficiently?

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

- What if we store this table in a row major order?
  - What operations we will be able to do efficiently?
- What if we store it in a column major order?

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

2D Array

```
table=[ ["IBM", “Sonny”, ”Nintendo”, ”AirCanada"],
          [”USA", "Japan", “Japan”, ”Canada"],
          [20000, ”5000", 3000, 5000],
          [“True”, “True”, ”True”, “True”]
        ]
```

Column Major Order

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
IBM	USA	20000	True
Sony	Japan	5000	True
Nintendo	Japan	3000	True
AirCanada	Canada	5000	True

- What happens when you alter a table?
- **Physical data independence**: the logical definition of the data remains unchanged, even when we make changes to the actual implementation

Selections in SQL

```sql
SELECT * FROM Product WHERE Price > 100.0
```

Joins in SQL

```
SELECT pname, price
FROM Product, Company
WHERE manufacturer=cname AND
        country = ‘Japan’ AND price < 150
```

Retrieve all Japanese Products that costs < $150

Simple SQL Query: Selection

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT *
FROM   Product
WHERE  Category = ‘Gadgets’
```

Simple SQL Query: Selection

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT *
FROM   Product
WHERE  Category = ‘Gadgets’
```

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Simple SQL Query: Projection

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT Pname, Price, Manufacturer
FROM   Product
WHERE  Category = ‘Gadgets’
```

PName	Price	Manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks

LIKE: Simple String Pattern Matching

```
SELECT *
FROM   Products
WHERE  PName LIKE ‘%gizmo%’
```

- s **LIKE** p: pattern matching on strings
- p may contain special symbols:
  - % = any sequence of characters

DISTINCT: Eliminating Duplicates

```
SELECT DISTINCT Category
FROM   Product
```

Category
Gadgets
Photography
Household

Versus

```
SELECT Category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

ORDER BY: Sorting the Results

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Category=‘gizmo’ AND Price > 50
ORDER BY Price, PName
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.

Selecting Data

The **SELECT** statement is used to retrieve data from one or more database tables.

SELECT *list_of_fields*

FROM *list_of_tables*

WHERE *where_clause*

GROUP BY *group_by_clause*

HAVING *having_clause*

ORDER BY *order_by_clause*

Updating Data

- The **UPDATE** statement is used to update information in database tables.
- The following statement a specific customer’s contact name:

```
UPDATE Customers
SET ContactName = 'Maria Anderson'
WHERE CustomerId = 'ALFKI'
```

Inserting Data

- The **INSERT** statement is used to add one or more rows to a database table.
- The following statement inserts a new record to the Order Details table:

```
INSERT INTO [Order Details]
(OrderId, ProductId, UnitPrice, Quantity, Discount)
VALUES (10248, 2, 19.00, 2, 0)
```

Deleting Data

- The **DELETE** statement is used to remove information from database tables.
- The following statement deletes a record from the Customers table:

```
DELETE FROM Customers
WHERE CustomerId = 'ALFKI'
```