UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

CS2208: Information Storage and Management I

Dr. Alejandro Arbelaez

a.arbelaez@cs.ucc.ie

Rules for our online lectures/labs

- The chat of the Team is only for our live lectures and labs or exceptional circumstances (e.g., Canvas is down) – Canvas is the main communication channel.
- For additional questions please use the discussion boards (Canvas).
- If I miss your question, please reformulate your question in the discussion board.
- Stay on topic – only CS2208 related questions.
- Keep on top of the online content – both synchronous and asynchronous.
- I'll be recording our lectures (and/or lab), even if Teams isn't saying so.

Discussion – Relational Databases

- 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"]
      ]
```

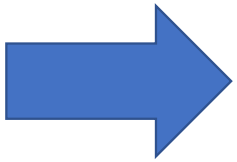
Table Implementation

- 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?

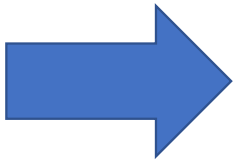
Table Implementation

- 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

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

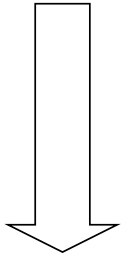
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
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'
```

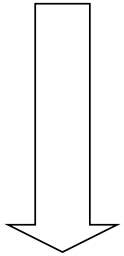


PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

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

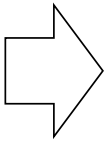
LIKE: Simple String Pattern Matching

```
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

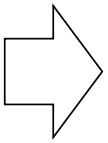
```
SELECT DISTINCT Category
FROM   Product
```



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'
```

The MySQL Server

- A MySQL server can be given SQL commands, executes them, and results the results to the connected application:
- Starting the server
- Stopping the server

Command Line Interface

- Starting the MySQL client
- Commands for looking around in the database
  - SHOW DATABASES
  - USE database
  - SHOW TABLES
  - DESCRIBE table

Tables

- A typical database table definition has:
- A name
- A list of columns and their data types
- A list of constraints
  - Primary key to ensure uniqueness (if needed)
  - Foreign keys to facilitate relationships with other tables
  - Indices to facilitate fast look ups

Primary Key

- A **primary key** is a field in a table which uniquely identifies each row/record in a database table.
- **Primary keys** must contain unique values.
- A **primary key** column cannot have NULL values.
- A table can have only one **primary key**, which may consist of single or multiple fields.

employee_id	course_id	taken_date
100	3	1987-06-17
101	3	1989-09-21
102	3	1993-01-13
103	3	1990-01-03
104	3	1991-05-21
105	3	1997-06-25
106	3	1998-02-05

Primary Key

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2)
);
```

Primary Key

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2)
);
```

We are missing the primary key(s)

```
ALTER TABLE Customers
ADD PRIMARY KEY (ID);
```

Primary Key

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

Primary Key (with Multiple Columns)

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2),
    PRIMARY KEY (ID, NAME)
);
```

Delete Primary Key

ALTER TABLE Customers
DROP PRIMARY KEY ;

You can clear the primary key constraints from the table with the syntax given below.

Database structure for Quizes

id	description	creation_date
1	Favorite Things Quiz	10/07/2014 10:22

quiz table

id	text	quiz_id
1	What is your favorite color?	1
2	What is your favorite book?	1
…		

question table

id	text	point_value	question_id
1	Red	1	1
2	Green	10	1
…			

answer table

Creating The Database Schema

```
CREATE TABLE quiz (
    id INT NOT NULL AUTO_INCREMENT,
    description VARCHAR(255),
    create_time DATETIME NOT NULL,
    PRIMARY KEY(id)
)

CREATE TABLE question (
    id INT NOT NULL AUTO_INCREMENT,
    text VARCHAR(255),
    quiz_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (quiz_id) REFERENCES
quiz(id) ON DELETE CASCADE
)
```

```
CREATE TABLE answer (
    id INT NOT NULL AUTO_INCREMENT,
    text VARCHAR(255) NOT NULL,
    point_value INT NOT NULL,
    question_id INT NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY (question_id) REFERENCES
question(id) ON DELETE CASCADE
)
```

Filling In the Details

- Data types:
  - Numbers: INT, LONGINT, NUMERIC, FLOAT, DOUBLE
  - Strings: VARCHAR(<<NUM CHARS>>), TEXT, BLOB
  - Other: DATETIME
- NOT NULL vs NULL: whether to allow empty values or not
- PRIMARY KEY and FOREIGN KEY
- CASCADE: Keeping the data clean and robust

Null Values

- It is possible for tuples to have null values, denoted by null, for some of their attributes
- Null signifies an unknown value or that a value does not exist
- The result of any arithmetic expression involving null is null
  - Example: 5 + null returns null
- The predicate is null can be used to check for null values
  - **SELECT** name **FROM** instructor **WHERE** salary **is null**
- The predicate null is not null success if the value on which it is applied is not null

Null Values (Cont.)

- SQL treats as unknown the result of any comparison involving a null value (other than predicates is null and is not null).
  - Example: 5 < null or null <> null or null = null
- The predicate in a where clause can involve Boolean operations (and, or, not); thus the definitions of the Boolean operations need to be extended to deal with the value unknown.
  - and : (true and unknown) = unknown,
    (false and unknown) = false,
    (unknown and unknown) = unknown
  - or: (unknown or true) = true,
    (unknown or false) = unknown
    (unknown or unknown) = unknown
- Result of where clause predicate is treated as false if it evaluates to unknown

Aggregate Functions

These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Aggregate Functions Examples

- Find the average salary of instructors in the Computer Science department
  - **SELECT AVG**(salary) **FROM** instructor **WHERE** dept_name= 'Comp. Sci.'
- Find the total number of instructors who teach a course in the Spring 2010 semester
  - **SELECT COUNT** (distinct ID) **FROM** teaches **WHERE** semester = 'Spring' and year = 2018
- Find the number of tuples in the course relation
  - **SELECT COUNT** (*) **FROM** course;

Aggregate Functions – Group By

- Find the average salary of instructors in each department

SELECT [?]

FROM instructor **GROUP BY** dept_name;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Aggregate Functions – Group By

- Find the average salary of instructors in each department

SELECT dept_name, AVG (salary) AS avg_salary **FROM** instructor **GROUP BY** dept_name;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

Aggregate Functions – Group By

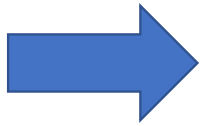
- Find the average salary of instructors in each department

SELECT dept_name, AVG (salary) AS avg_salary **FROM** instructor **GROUP BY** dept_name;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregation (Cont.)

Please Answer  Link

Attributes in select clause outside of aggregate functions must appear in group by list

correct or incorrect

SELECT dept_name, ID, **AVG** (salary)
FROM instructor
GROUP BY dept_name;

Aggregation (Cont.)

Attributes in select clause outside of aggregate functions must appear in group by list

```
/* erroneous query */
SELECT dept_name, ID, AVG (salary)
FROM instructor
GROUP BY dept_name;
```

Aggregate Functions – Having Clause

Find the names and average salaries of all departments whose average salary is greater than 42000

```
SELECT dept_name, AVG (salary) AS avg_salary
FROM instructor
GROUP BY dept_name
HAVING AVG (salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Null Values and Aggregates

- Total all salaries

  SELECT SUM (salary) FROM instructor

  - Above statement ignores null amounts
  - Result is null if there is no non-null amount
- All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

IN Operator

The IN operator is a shorthand for multiple OR conditions

- **SELECT** * **FROM** Customers
  WHERE Country **IN** ('Germany', 'France', 'UK');

Selects all customers that are located in "Germany", "France" or "UK":

- **SELECT** * **FROM** Customers
  WHERE Country NOT IN ('Germany', 'France', 'UK');

Selects all customers that are NOT located in "Germany", "France" or "UK":

EXISTS Operator

- The EXISTS operator is used to test for the existence of any record in a subquery
- The EXISTS operator returns true if the subquery returns one or more records
- If a subquery returns any rows at all, EXISTS subquery is TRUE

```
SELECT column_name(s) FROM table_name
WHERE EXISTS
    (SELECT column_name FROM table_name WHERE
    condition);
```

EXISTS Operator

Find all records from the *customers* table where there is at least one record in the *orders* table with the same *customer_id*

customer_id	last_name	website
4000	Jackson	www.ucc.ie
5000	Smith	www.google.ie
6000	Ferguson	www.microsoft.com
7000	Reynolds	www.facebook.com
8000	Anderson	www.youtube.com
9000	Johnson	www.thejournal.ie

```
SELECT * FROM customers WHERE EXISTS
    (SELECT * FROM orders WHERE
        customers.customer_id = orders.customer_id);
```

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20

customer_id	last_name	website
4000	Jackson	www.ucc.ie
5000	Smith	www.google.ie
7000	Reynolds	www.facebook.com
8000	Anderson	www.youtube.com

NOT EXISTS Operator

NOT condition can be combined with the **EXISTS** condition to create a **NOT EXISTS** condition. Let's look at an example that shows how to use the **NOT EXISTS** condition in SQL.

customer_id	last_name	website
4000	Jackson	www.ucc.ie
5000	Smith	www.google.ie
6000	Ferguson	www.microsoft.com
7000	Reynolds	www.facebook.com
8000	Anderson	www.youtube.com
9000	Johnson	www.thejournal.ie

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20
5	NULL	2016/04/01

```
SELECT * FROM customers WHERE NOT EXISTS
    (SELECT * FROM orders WHERE
        customers.customer_id = orders.customer_id);
```

customer_id	last_name	website
6000	Ferguson	www.microsoft.com
9000	Johnson	www.thejournal.ie

Not in the table

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries. A subquery is a select-from-where expression that is nested within another query.
- The nesting can be done in the following SQL query

SELECT A1, A2, ..., An **FROM** r1, r2, ..., rm **WHERE** P

as follows:

From clause: ri can be replaced by any valid subquery

Where clause: P can be replaced with an expression of the form:

B <operation> (subquery)

Where B is an attribute and <operation> to be defined later.

Select clause:

Ai can be replaced be a subquery that generates a single value.

Nested Subqueries

SELECT * **FROM** Customers **WHERE**

Country **IN** (**SELECT** Country **FROM** Suppliers);

Selects all customers that are from the same countries as the suppliers

Modification of the Database

- Deletion of tuples from a given relation/table.
- Insertion of new tuples into a given relation/table
- Updating of values in some tuples in a given relation/table

Deletion

- Delete all instructors

```
DELETE FROM instructor
```

- Delete all instructors from the Finance department

```
DELETE FROM instructor
WHERE  dept_name= 'Finance';
```

Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

```
DELETE FROM instructor
WHERE salary < (SELECT AVG(salary)
                FROM instructor);
```

Problem: as we delete tuples from deposit, the average salary changes

Solution used in SQL:

1. First, compute **avg** (salary) and find all tuples to delete
2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Insertion

- Add a new tuple to course

```
INSERT INTO course
VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- or equivalently

```
INSERT INTO course (course_id, title, dept_name, credits)
VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Add a new tuple to student with tot_creds set to null

```
INSERT INTO student
VALUES ('3003', 'Green', 'Finance', null);
```

Insertion (Cont.)

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.

```
INSERT INTO instructor
SELECT ID, name, dept_name, 18000
FROM  student
WHERE dept_name = 'Music' and total_cred > 144;
```

- The select from where statement is evaluated fully before any of its results are inserted into the relation.
- Otherwise queries like

```
INSERT INTO table1 SELECT * FROM table1
```

would cause problem

Updates

- Give a 5% salary raise to all instructors

```
UPDATE instructor
SET salary = salary * 1.05
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
UPDATE instructor
SET salary = salary * 1.05
WHERE salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
UPDATE instructor
SET salary = salary * 1.05
WHERE salary < (select avg (salary)
                from instructor);
```

Destroying and Altering Relations

DROP TABLE Students

- Destroys Students

ALTER TABLE Students **ADD COLUMN** firstYear

- Students will be altered by adding anew field
- Every tuple in the current instance is extended with a null value in the new field