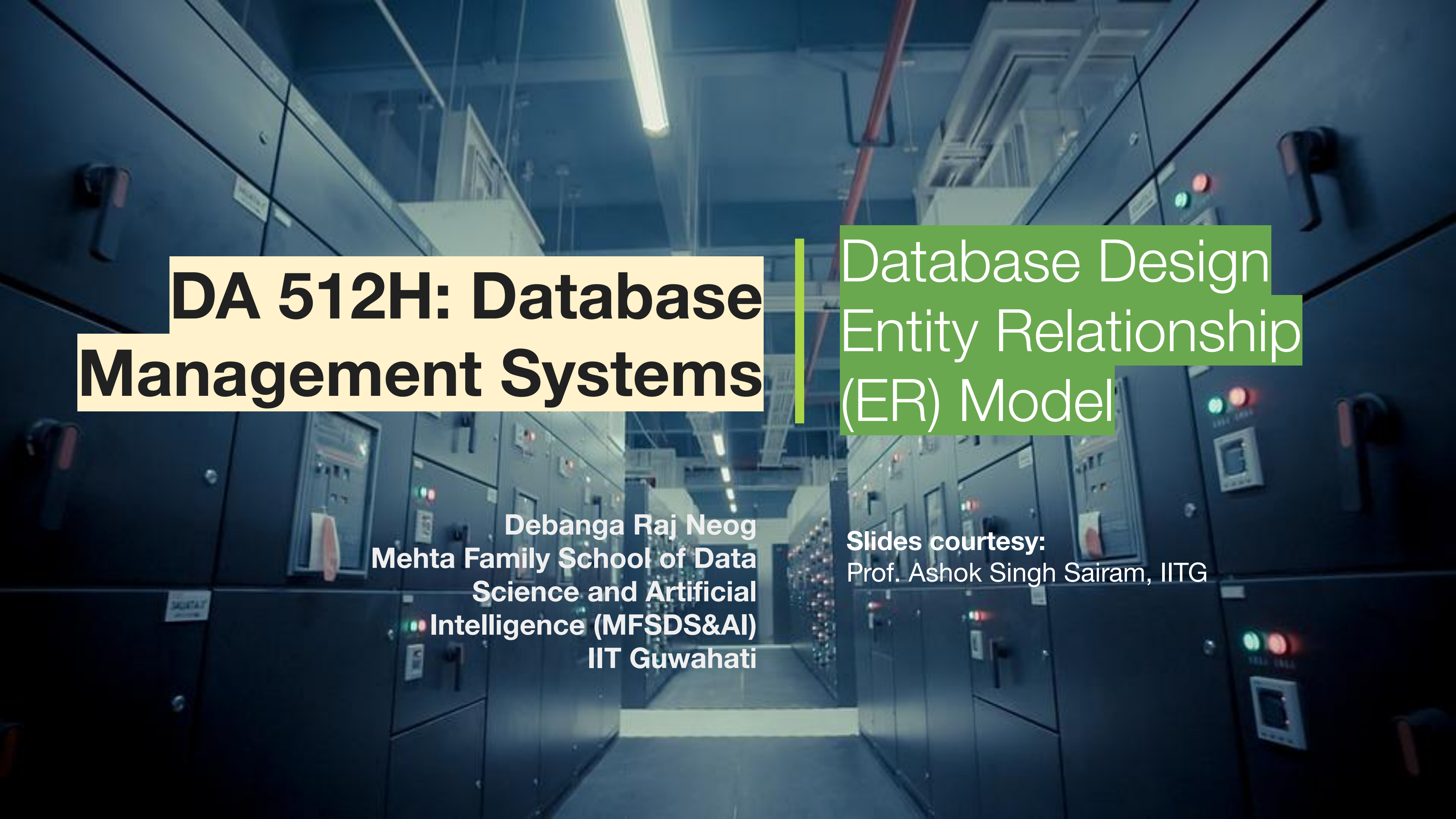


DA 512H: Database Management Systems

Database Design Entity Relationship (ER) Model

Debanga Raj Neog
Mehta Family School of Data Science and Artificial Intelligence (MFSDS&AI)
IIT Guwahati

Slides courtesy:
Prof. Ashok Singh Sairam, IITG

What is SQL?

What is SQL?

- SQL stands for **Structured Query Language**
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can set permissions on tables, procedures…

What is SQL?

- Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.
- However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.
- Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

Example:
To build a website that shows data from a database, you will need:

- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like PHP or ASP
- To use SQL to get the data you want
- To use HTML / CSS to style the page

Quick SQL Examples

- SELECT * FROM Orders
- SELECT EmployeeID FROM Orders
- SELECT DISTINCT EmployeeID FROM Orders
- SELECT * FROM Customers
- ORDER BY Country;
- …

We look at the SQL and its commands in the next class.

Online practice: https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all

SQL Introduction

Standard language for querying and manipulating data

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

Many standards out there:

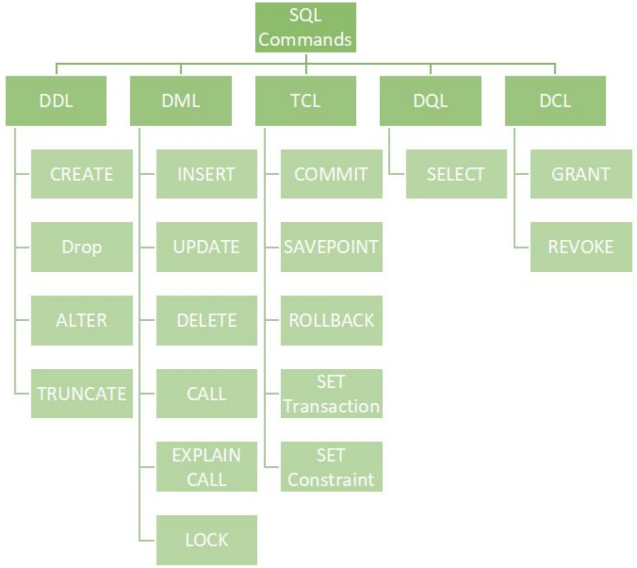
- ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ...

SQL

Structured Query Language(SQL) is the database language for RDBMS to create database or perform certain operations on the existing database and also we can use this language to create a database.

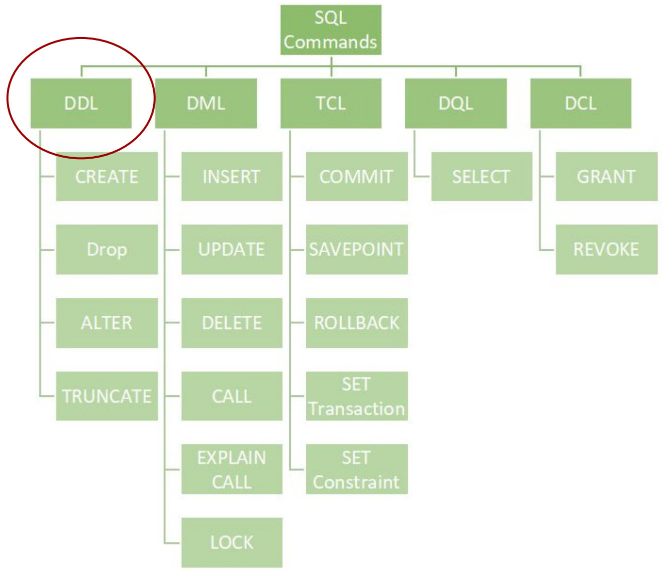
The commands are categorised into **four** categories:

1. DDL – Data Definition Language
2. DQl – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language



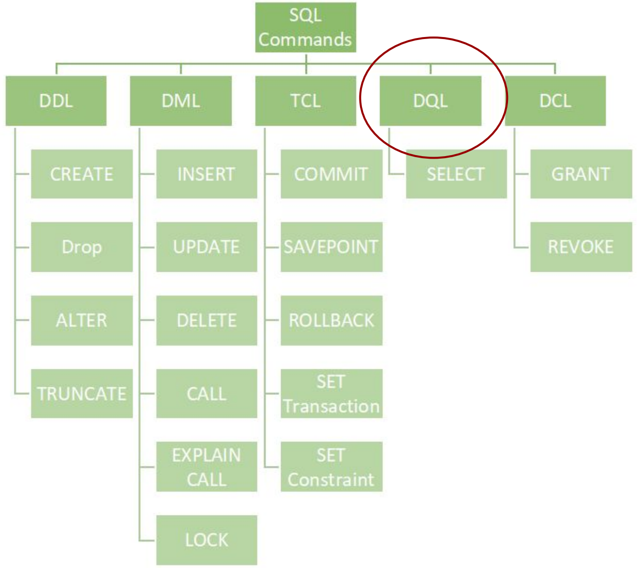
SQL: DDL (Data Definition Language)

- Data Definition Language actually consists of the SQL commands that can be used to define the database schema.
- DDL is a set of SQL commands used to create, modify, and delete database structures but not data.
- These commands are normally not used by a general user, who should be accessing the database via an application.



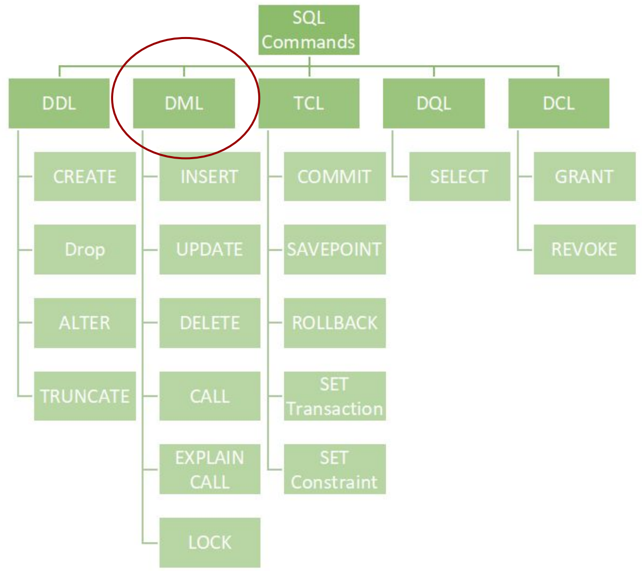
SQL: DQL (Data Query Language)

- DQL statements are used for performing queries on the data within schema objects.
- It is a component of SQL statement that allows getting data from the database and imposing order upon it.
- When a SELECT is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.



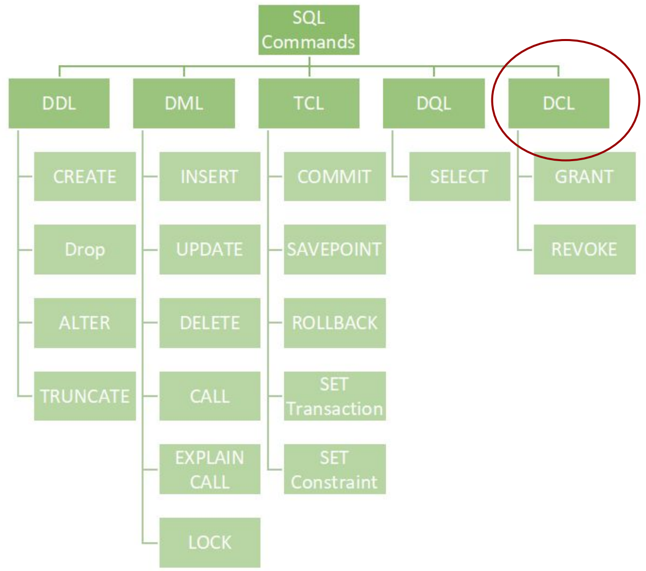
SQL: DML(Data Manipulation Language)

- The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation



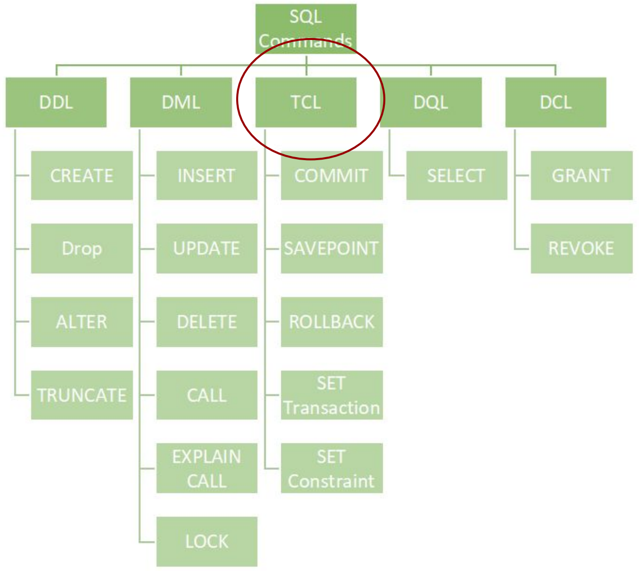
SQL: DCL (Data Control Language)

- DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.



SQL: TCL

There is one more category known as **Transaction Control Language (TCL)**. These commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.



Tables in SQL

Table name

Product

Attribute names

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

Tuples or rows

Tables Explained

- The *schema* of a table is the table name and its attributes:

  Product(PName, Price, Category, Manufacturer)

- A *key* is an attribute whose values are unique; we underline a key

  Product($\underline{\text{PName}}$, Price, Category, Manufacturer)

Data Types in SQL

- Atomic types:
    - Characters: CHAR(20), VARCHAR(50)
    - Numbers: INT, BIGINT, SMALLINT, FLOAT
    - Others: MONEY, DATETIME, …

- Every attribute must have an atomic type

Tables Explained

- A tuple = a record
    - Restriction: all attributes are of atomic type

- A table = a set of tuples
    - Like a list…
    - …but it is unordered:
      no **first()**, no **next()**, no **last()**.

SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM     <one or more relations>
WHERE  <conditions>
```

Simple SQL Query

Product

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

"selection"

Simple SQL Query

Product

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   Price > 100
```

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

"selection" and "projection"

Notation

```
SELECT   PName, Price, Manufacturer
FROM      Product
WHERE   Price > 100
```

Input Schema

Product(PName, Price, Category, Manufacturer)

$\Downarrow$

Answer(PName, Price, Manufacturer)

Output Schema

Details

- Case insensitive:
  - Same: SELECT Select select
  - Same: Product   product
  - Different: ‘Seattle’ ‘seattle’

- Constants:
  - ‘abc’ - yes
  - “abc” - no

The **LIKE** operator

```
SELECT   *
FROM     Product
WHERE   PName LIKE '%gizmo%'
```

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

Eliminating Duplicates

```
SELECT   DISTINCT category
FROM     Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT   category
FROM     Product
```

Category
Gadgets
Gadgets
Photography
Household

Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    price > 10
ORDER BY  manufacturer DESC, pname
```

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

Ordering is ascending, unless you specify the DESC keyword.

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   DISTINCT category
FROM     Product
ORDER BY category
```

⇨ ?

```
SELECT   Category
FROM     Product
ORDER BY  PName
```

⇨ ?

```
SELECT   DISTINCT category
FROM     Product
ORDER BY PName
```

⇨ ?

Keys and Foreign Keys

Company

Key

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

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

Foreign key

Joins

Product (<u>pname</u>, price, category, manufacturer)
Company (<u>cname</u>, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and prices.

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

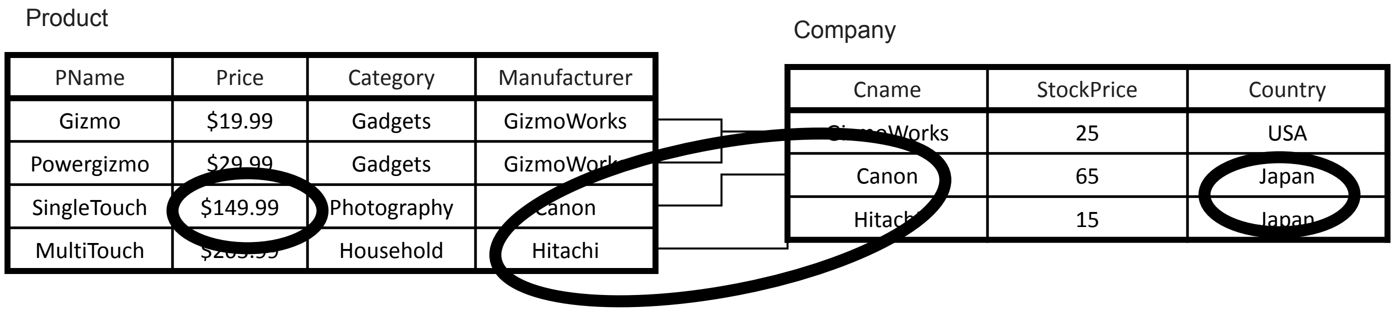
Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

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

Joins



Product

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

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   PName, Price
FROM      Product, Company
WHERE   Manufacturer=CName AND Country='Japan'
          AND Price <= 200
```

PName	Price
SingleTouch	$149.99

A Subtlety about Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all countries that manufacture some product in the 'Gadgets' category.

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   Country
FROM      Product, Company
WHERE   Manufacturer=CName AND Category='Gadgets'
```

Product

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

Unexpected duplicates

A Subtlety about Joins

Product

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

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT   Country
FROM      Product, Company
WHERE   Manufacturer=CName AND Category='Gadgets'
```

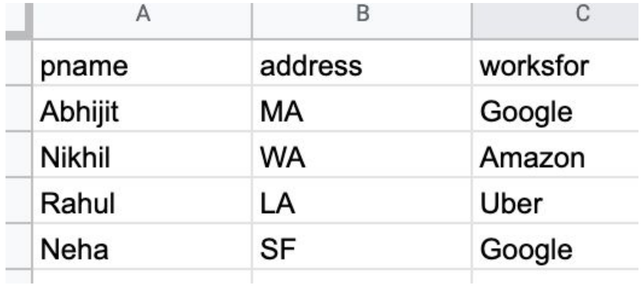
Country
??
??

What is
the problem ?
What's the
solution ?

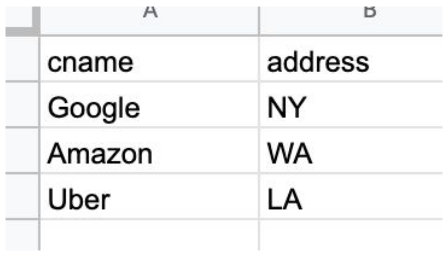
Tuple Variables

Person(<u>pname</u>, address, worksfor)

Company(<u>cname</u>, address)



A	B	C
pname	address	worksfor
Abhijit	MA	Google
Nikhil	WA	Amazon
Rahul	LA	Uber
Neha	SF	Google



A	B
cname	address
Google	NY
Amazon	WA
Uber	LA

```
SELECT   DISTINCT pname, address
FROM      Person, Company
WHERE   worksfor = cname
```

Which address ?

```
SELECT   DISTINCT Person.pname, Company.address
FROM      Person, Company
WHERE   Person.worksfor = Company.cname
```

```
SELECT   DISTINCT x.pname, y.address
FROM      Person AS x, Company AS y
WHERE   x.worksfor = y.cname
```

Subqueries Returning Relations

Company(<u>name</u>, city)
Product(<u>pname</u>, price, category, maker)
Purchase(<u>id</u>, product, buyer)

Return cities where one can find companies that manufacture products bought by Joe Blow

They are equivalent

```
SELECT DISTINCT Company.city
FROM    Company
WHERE  Company.name  IN
                  (SELECT Product.maker
                   FROM   Purchase , Product
                   WHERE Product.pname=Purchase.product
                       AND Purchase.buyer = 'Joe Blow');
```

```
SELECT DISTINCT Company.city
FROM     Company, Product, Purchase
WHERE   Company.name= Product.maker
        AND  Product.pname  = Purchase.product
        AND  Purchase.buyer = 'Joe Blow'
```

Subqueries Returning Relations

You can also use: s > ALL R
s > ANY R
EXISTS R

Product (pname, price, category, maker)
Find products that are more expensive than all those produced
By “Gizmo-Works”

```
SELECT  pname
FROM     Product
WHERE  price >  ALL (SELECT price
                     FROM     Purchase
                     WHERE  maker=‘Gizmo-Works’)
```

Complex Correlated Query

Product (pname, price, category, maker, year)

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT DISTINCT  pname, maker
FROM     Product AS x
WHERE  price > ALL  (SELECT  price
                     FROM    Product AS y
                     WHERE  x.maker = y.maker AND y.year < 1972);
```

Aggregation

```
SELECT  avg(price)
FROM     Product
WHERE   maker="Toyota"
```

```
SELECT  count(*)
FROM     Product
WHERE   year > 1995
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT  Count(category)
FROM    Product
WHERE   year > 1995
```

same as Count(*)

We probably want:

```
SELECT  Count(DISTINCT category)
FROM    Product
WHERE   year > 1995
```

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM     Purchase
WHERE   product = ‘bagel’
```

⇨ ?

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM     Purchase
WHERE   product = 'bagel'
```

⇨

50 (= 20+30)

Grouping and Aggregation

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product.

```
SELECT     product, Sum(price*quantity) AS TotalSales
FROM        Purchase
WHERE      date > ‘10/1/2005’
GROUP BY  product
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

Let’s see what this means…

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

Product	TotalSales
Bagel	50
Banana	15

```
SELECT     product, Sum(price*quantity) AS TotalSales
FROM        Purchase
WHERE      date > ‘10/1/2005’
GROUP BY  product
```

HAVING Clause

Same query, except that we consider only products that had at least 100 buyers.

```
SELECT      product, Sum(price * quantity)
FROM        Purchase
WHERE       date > '10/1/2005'
GROUP BY product
HAVING      Sum(quantity) > 30
```

HAVING clause contains conditions on aggregates.

NULLS in SQL

- Whenever we don’t have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if can be null (*nullable* attribute) or not
- How does SQL cope with tables that have NULLs ?

Null Values

- If x= NULL then 4*(3-x)/7 is still NULL
- If x= NULL then x=“Joe”   is UNKNOWN
- In SQL there are three boolean values:

FALSE	=	0
UNKNOWN	=	0.5
TRUE	=	1

Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM    Person
WHERE  age < 25  OR  age >= 25 OR age IS NULL
```

Now it includes all Persons

Modifying the Database

Three kinds of modifications

- Insertions
- Deletions
- Updates

Sometimes they are all called “updates”

Insertions

General form:

```
INSERT   INTO   R(A1,…., An)   VALUES  (v1,…., vn)
```

Example: Insert a new purchase to the database:

```
INSERT  INTO  Purchase(buyer, seller, product, store)
          VALUES  (‘Joe’, ‘Fred’, ‘wakeup-clock-espresso-machine’,
                       ‘The Sharper Image’)
```

Missing attribute → NULL.
May drop attribute names if give them in order.

Deletions

Example:

```
DELETE   FROM   PURCHASE

WHERE   seller = ‘Joe’   AND
         product = ‘Brooklyn Bridge’
```

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.

Updates

Example:

```
UPDATE   PRODUCT
SET    price = price/2
WHERE  Product.name  IN
                (SELECT product
                 FROM    Purchase
                 WHERE  Date ='Oct, 25, 1999');
```

DA 214: Database Management Systems

Connecting to SQL Database

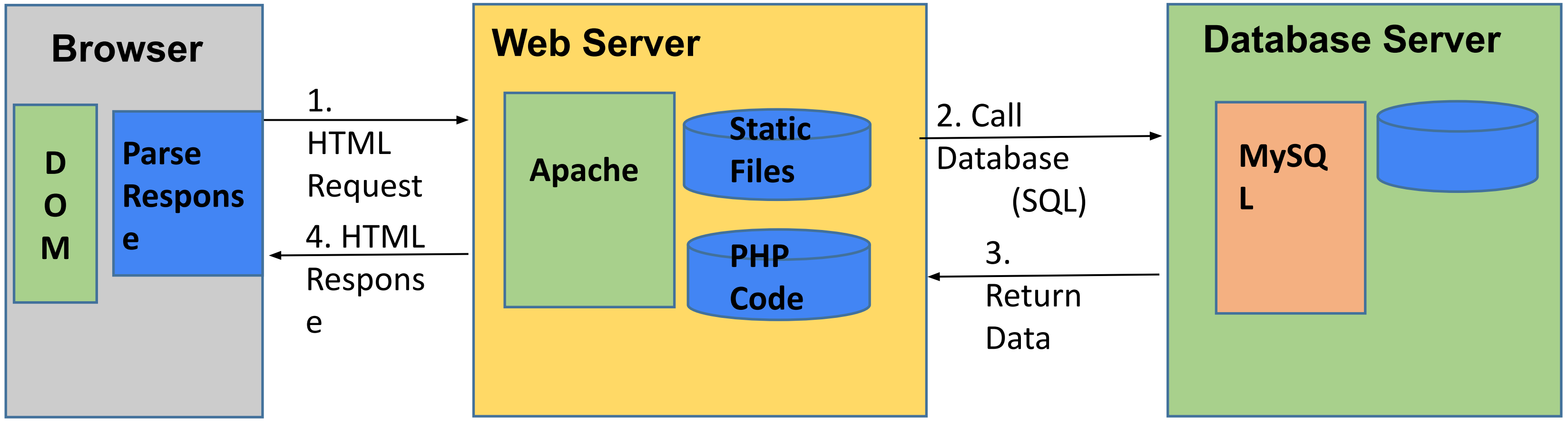
Debanga Raj Neog
Mehta Family School of Data Science and Artificial Intelligence (MFSDS&AI)
IIT Guwahati

Slides courtesy:
Prof. Ashok Singh Sairam, IITG

Lecture Plan

- Connecting to SQL Server
- Dump Contents in Array
- Print data in tabular format
- Insert data into table
- Delete data from table

Three-Tier Architecture



Ways to access MySQL

- PHP is evolving - there are three ways to access MySql
  - Legacy non-OO mysql_ routines (deprecated)
  - New mysqli (OO version that is similar to mysql_)
  - PDO - **PHP Data Objects**
- Recommendation: PDO MySQL extensions

- https://www.php.net/manual/en/mysqlinfo.api.choosing.php
- https://www.w3schools.com/php/php_mysql_connect.asp

Anatomy of PDO connection

```
$conn=new PDO('mysql:host=<hname>;port=<port#>; dbname=<dbname>',‘<user>',‘<pass>');
```

Review: Creating database

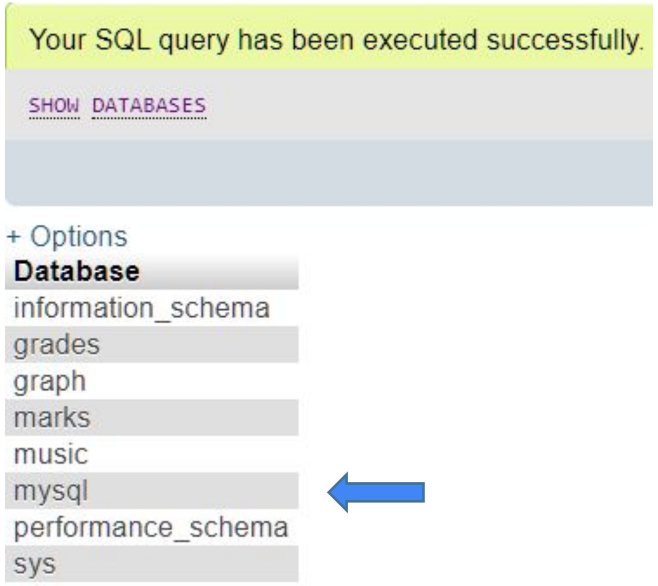
- CREATE DATABASE marks;
- USE marks;
- CREATE TABLE student (

```
          name varchar(28),
          rollno INT(7) not null,
          branch varchar(20),
          cpi decimal(4,2)
          );
```

- Insert into student values ("John", 12345, "MnC",9.1);

Create Users(1)

- Run the SQL statement “**show databases;**”
  - Note there is a database named mysql
- The database mysql has a table named user
  - Run the SQL statement “**DESCRIBE mysql.user;**” to check the fields of the table
- **SELECT * from mysql.user;**
  - List the users in the table



Your SQL query has been executed successfully.

```
SHOW DATABASES
```

Database
information_schema
grades
graph
marks
music
mysql
performance_schema
sys

Create Users(2)

- Syntax to create a user

CREATE USER 'debanga'@'localhost' IDENTIFIED BY 'test';

(for bug: `drop user debanga@localhost;`)

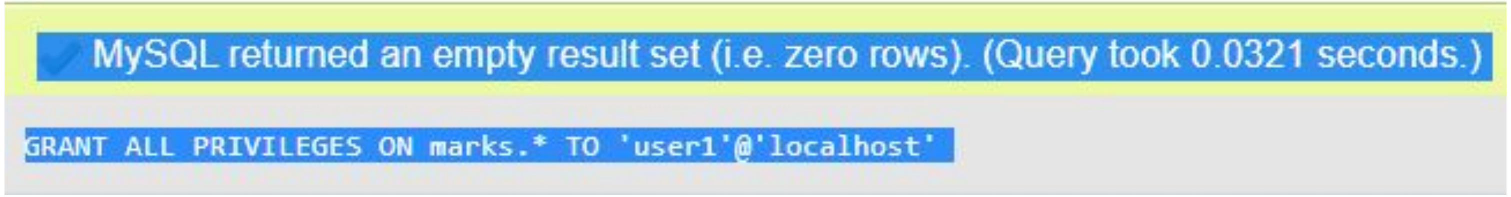
Create Users(3)

- Next grant privileges to the user to access a database

  GRANT privileges_names ON object TO user;

  - Ex: Grant all privileges to debanga;

  GRANT ALL PRIVILEGES ON marks.* to 'debanga'@'localhost';



```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0321 seconds.)

GRANT ALL PRIVILEGES ON marks.* TO 'user1'@'localhost'
```

Note that the GRANT statement is a Data Control Language

Connecting to the database

```
<!DOCTYPE HTML>
<HTML>
	<?php
		$servername = "localhost";
		$port_no = 8080;
		$username = "debanga";
		$password = "test";
		$myDB= "marks"; //Name of the database to access

		try {
		  $conn = new PDO("mysql:host=$servername;port=$port_no;dbname=$myDB", $username, $password);
		  // set the PDO error mode to exception
		  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
		  echo "Connected successfully";

		 } catch(PDOException $e) {
		  echo "Connection failed: " . $e->getMessage();
		 }
	?>
</HTML>
```

Second Program: Dump data from table

```
$stmt = $conn->query("SELECT * FROM marks.student");

echo "<pre>";
  while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
  print_r($row);
  }
  echo "</pre>";
```

Closing a connection

- To close the connection, set it to null

```
$stmt=null;

$conn=null;
```

Third Program: Output in tabular format

```
<?php
$servername = "localhost"; //Name of the server where database …..
$conn = new PDO("mysql:host=$servername; port= $port_no, dbname=$myDB",
$username, $password);
…..
$stmt = $conn->query("SELECT name, rollno, branch, cpi FROM marks.student");
echo '<table border="1">',"\n";
echo "<tr> <td> Name </td> <td> Roll No </td> <td> Branch </td> <td> CPI </td>
</tr>";
while ($row=$stmt->fetch(PDO::FETCH_ASSOC)) {
        echo "<tr> <td>";
        echo $row['name'];
        echo "</td><td>";
```

Third Program: Output in tabular format

```
echo '<table border="1">',"\n";
  echo "<tr> <td> Name </td> <td> Roll No </td> <td> Branch </td> <td> CPI </td> </tr>";
  while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
     echo "<tr>";
     echo "<td>";
     echo $row['name'];
     echo "</td>";
     echo "<td>";
     echo $row['rollno'];
     echo "</td>";
     echo "<td>";
     echo $row['branch'];
     echo "</td>";
     echo "<td>";
     echo $row['cpi'];
     echo "</td></tr>";
  }
  echo '</table>';
```

4th Program: Insert data in mySQL DB

```
$sql = "INSERT INTO marks.student values ('Debanga', 12113, 'EEE', 9.5)";
$conn->exec($sql);
```

5th Program: Delete data in mySQL DB

```
$sql = "DELETE FROM marks.student WHERE cpi>8.3";
$conn->exec($sql);
```