Database Fundamentals

SQL INTRODUCTION – DATA MANIPULATION LANGUAGE

SQL Commands Overview - Data Manipulation

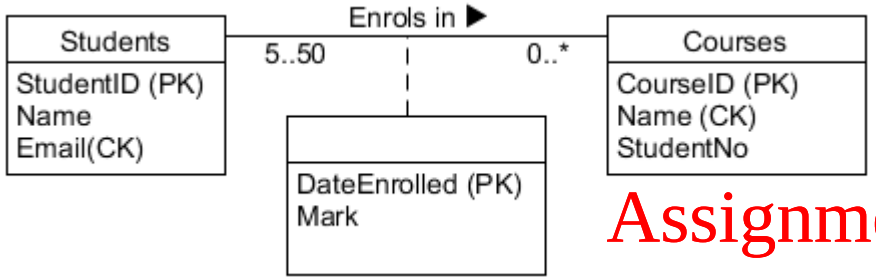
- Key Data Manipulation SQL commands:
  - **INSERT INTO** - Creates new tuple(s) in a table in the database
  - **DELETE FROM** - Permanently removes tuple(s)
  - **UPDATE** - Modifies existing tuple(s) in a table in the database
  - **SELECT** - Retrieves one or more tuples of data (a query)

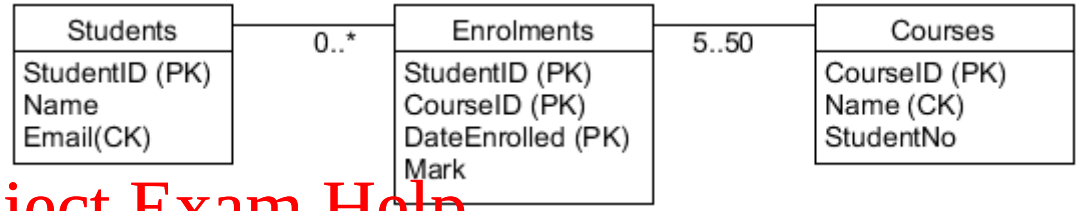
What's this for?

- So we have created a set of tables.
- How do we actually put data in our database?
- How do we modify data that is already in our database?
- How do we answer questions using our database data?

Example Database Design

Possible Conceptual Designs



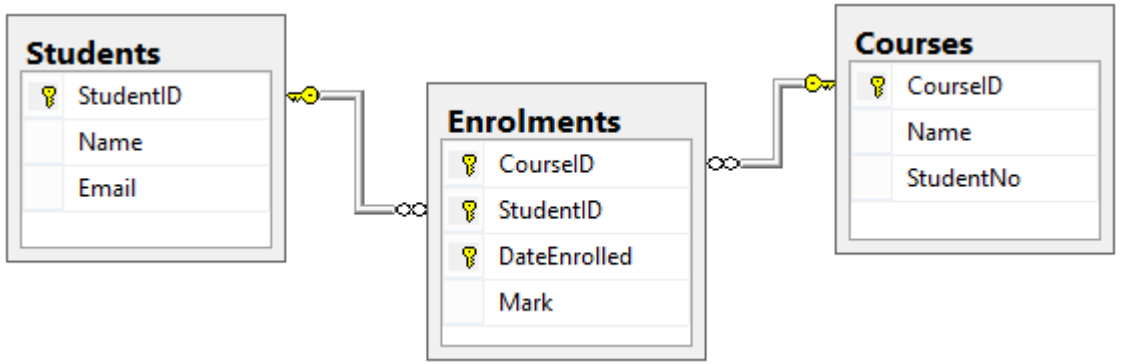


Students(StudentID, Name, Email)
PK(StudentID)
CK(Email)

Courses(CourseID, Name, StudentNo)
PK(CourseID)
CK(Name)

Enrolments(StudentID, CourseID, DateEnrolled, Mark)
PK(StudentID, CourseID, DateEnrolled)
FK(StudentID) -> Students(StudentID)
FK(CourseID) -> Courses(CourseID)

Database Implementation



Example Database Design

```
CREATE TABLE Students (
StudentID int,
Name varchar(100) NOT NULL,
Email varchar(100) NOT NULL,
CONSTRAINT StudentPK PRIMARY KEY(StudentID),
CONSTRAINT UniqueEmail UNIQUE(Email)
);
```

Students(<u>StudentID</u>, Name, Email)
PK(StudentID)
CK(Email)


```
CREATE TABLE Courses (
CourseID int,
Name varchar(100) NOT NULL,
StudentNo int DEFAULT 50,
CONSTRAINT CoursePK PRIMARY KEY(CourseID),
CONSTRAINT UniqueName UNIQUE(Name),
CONSTRAINT maxStudents CHECK(StudentNo BETWEEN 5 AND 150)
);
```

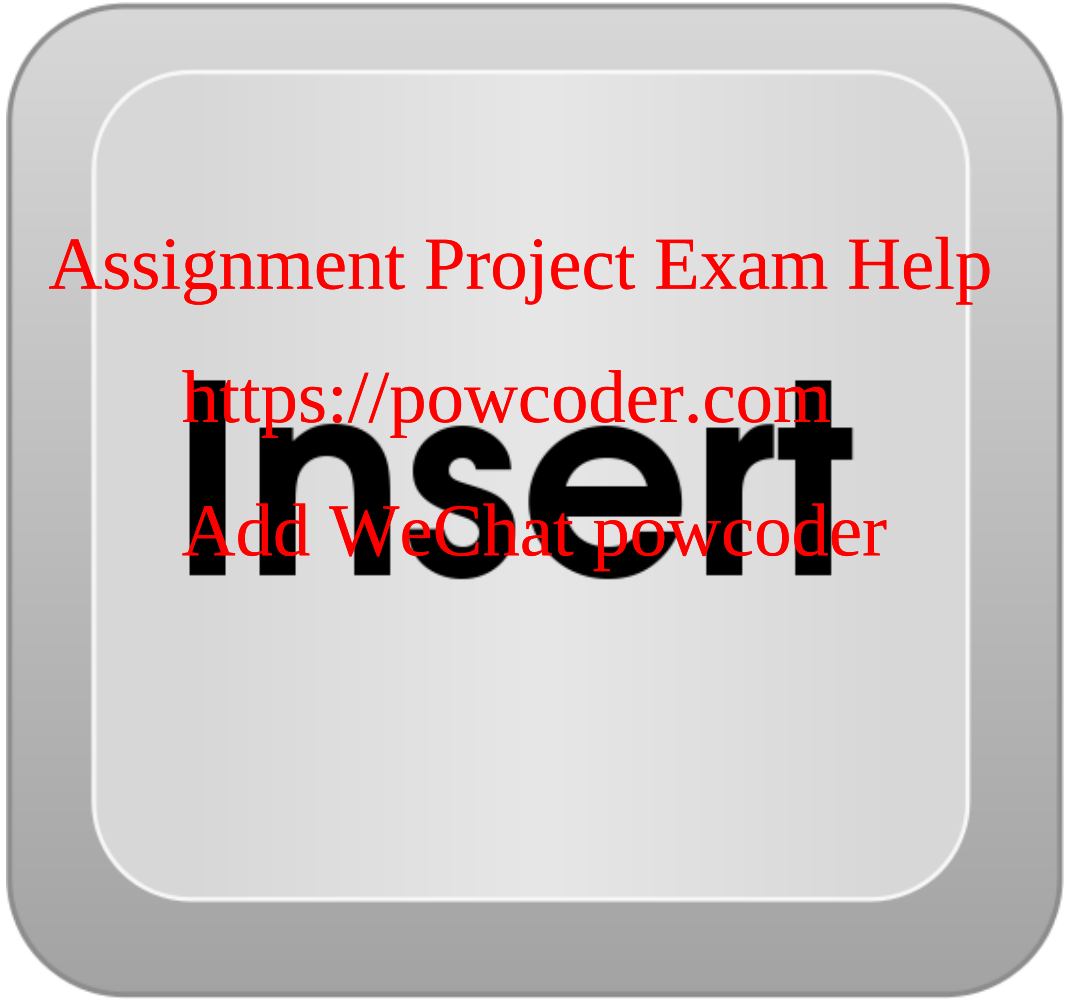
Courses(<u>CourseID</u>, Name, StudentNo)
PK(CourseID)
CK(Name)

Example Database Design

```
CREATE TABLE Enrolments (
CourseID int NOT NULL,
StudentID int NOT NULL,
DateEnrolled date DEFAULT getDate(),
Mark int CHECK(Mark BETWEEN 0 AND 100),
CONSTRAINT EnrolmentPK PRIMARY KEY(CourseID, StudentID, DateEnrolled),
CONSTRAINT theStudent FOREIGN KEY(StudentID) REFERENCES Students(StudentID),
CONSTRAINT theCourse FOREIGN KEY(CourseID) REFERENCES Courses(CourseID)
);
```


Enrolments(<u>StudentID</u>, <u>CourseID</u>, <u>DateEnrolled</u>, Mark)
PK(StudentID, CourseID, DateEnrolled)
FK(StudentID) -> Students(StudentID)
FK(CourseID) -> Courses(CourseID)

INSERT INTO



SQL Commands Overview - **INSERT INTO**

- Used to populate with data - your newly created database!
  - String data (text) must be wrapped in single ‘quotations’
    - Same applies to date strings: ‘01/03/2014’

Be careful using this method if the order of columns changes.

- Syntax:
  - **INSERT INTO** <tableName> **VALUES** (x,y,z)
    - Inserts new values in the default order of the columns in the database table

  - **INSERT INTO** <tableName> (att3, att2, att1) **VALUES** (z, y, x)
    - Inserts the values in a different order to the default order of the columns in the database table

The SAME order of attributes when the table was created!!

- Examples:
  - **INSERT INTO** Student **VALUES** (‘50011’, ‘Barry’);
  - **INSERT INTO** Student (StudentName, StudentID) **VALUES** (‘Stacy’, ‘

Order specified in this statement

Note the different order of StudentName vs StudentID. If the order is not specified, you **must** insert the values in order of the table column names (Left → Right)

SQL Commands Overview - **INSERT INTO**

- The order of the attributes (if present) **are important** (first value for the first attribute, and so on)
  - If the ***AttributeList*** is omitted, all the values must be supplied in the same order used when the table structure was created
    - CREATE Table (Col1, Col2, Col3)
    - INSERT INTO Table VALUES (val1, val2, val3)
      - Where Val1 is for Col1, Val2 for Col2 …
  - If the ***AttributeList*** does not contain all the attributes, the remaining attributes are assigned their **DEFAULT** values (or **NULL** if no default has been defined)
    - INSERT INTO Table(col1, col2) VALUES (val1, val2)
      - The new row inserted will be (val1, val2, defaultVal/NULL)
      - Assuming the table has 3 columns ☺

This WILL throw an ERROR if the attribute is set as **NOT NULL** and no DEFAULT has been specified

SQL Commands Overview - **INSERT INTO**

```
INSERT INTO Departments VALUES ('CIS', 'Computer and Information Science');
INSERT INTO Employees VALUES  ('E001', 'John', 'Smiths', 'CIS',  40000, '23-JAN-1980');
/* The value order MUST match the attribute order specified when the table was created (previous slide!) */
/* Dates with the month in text must be encased in single quotes if the Month contains text */
```



```
INSERT INTO Employees   (Dept, Regno, Firstname, Surname)
VALUES ('CIS','E002','Peter','Smiths');
/* The value order MUST match the attribute order of the insert query  */
```

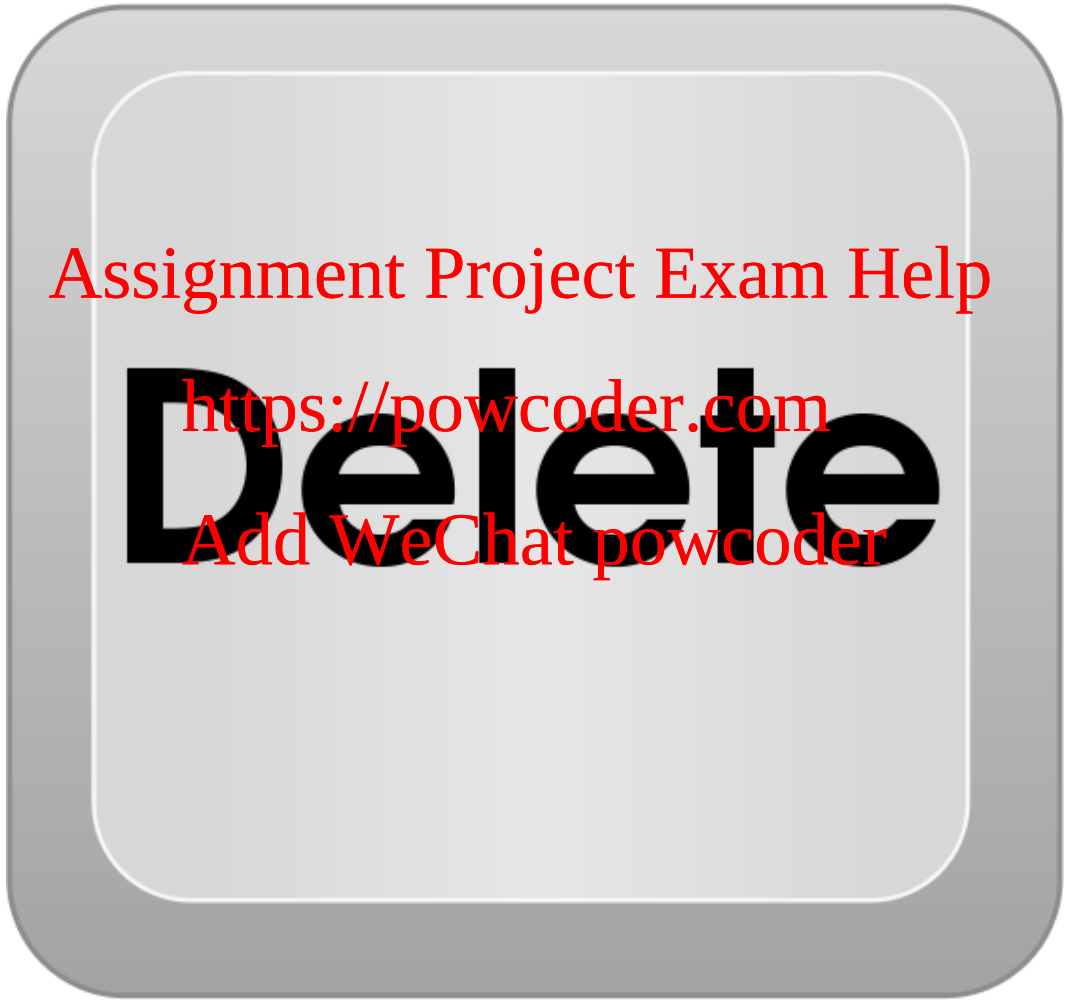
Best Approach

In both cases:

- values for PK and unique constraints must be satisfied
- values must satisfy FK and other constraints

```
SQL> SELECT * FROM employees;
REGNO  FIRSTNAME     SURNAME     DEPT        SALARY BDATE
------ ------------ ---------- ------ ---------- -----------
E001   John         Smiths      CIS         40000 23-JAN-1980
E002   Peter        Smiths      CIS             0     NULL
```

Delete From



SQL Commands Overview - **DELETE FROM**

- The **DELETE FROM** statement removes all the tuples that satisfy the condition from a given table
  - The removal may result in deletions from other tables if a FOREIGN KEY constraint with CASECADE ON DELETE has been used

- If the **WHERE** clause is omitted, **DELETE FROM** removes **ALL** tuples from the table:


```
DELETE FROM Departments; /* removes ALL departments from the table */

DELETE FROM Departments WHERE DeptName = 'ITMS' ;
        /* removes ALL departments from the table with the name ITMS */
```

SQL Commands Overview - **DELETE FROM**

- The **WHERE** clause dictates which rows of data should be deleted
  - Each row is compared to the **WHERE** clause and if it returns TRUE, the row is **deleted**

- If you want to delete a specific record, what should the **WHERE** clause contain?

DELETE FROM Departments **WHERE** ??

SQL Commands Overview - **DELETE FROM**

- Delete tuples
  - DELETE FROM Employees;
    - Deletes ALL tuples from the table, but not the table/schema itself
  - DELETE FROM employees WHERE empno = '001';
    - Deletes matching tuples (one tuple is deleted in previous example)
- Delete the tuples and the schema (remove the table)
  - DROP TABLE Departments

```
DROP TABLE Employees;
DROP TABLE Departments;
```

    - Deletes all employees and Employee table followed by departments
  - Can the opposite be done?
    - If Employees contains a reference (FK) to Departm

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

Update

SQL Commands Overview – UPDATE tableName SET

- Syntax

```
UPDATE TableName
  SET Attribute=<Expression|SelectSQL|null|default>
  {, SET  Attribute=<Expression|SelectSQL|null|default>}
  [WHERE Condition]
```


- Examples


```
UPDATE Employees
SET Salary = Salary + 5
WHERE RegNo = 'M2047'
```


Note correction

```
UPDATE Employees
SET Salary = Salary * 2.0
WHERE Dept = 'Middle Management'
```

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

SQL Commands Overview – UPDATE tableName SET

- Since the language is set-oriented, the order of the statements **can be important**

Query 1

```
UPDATE Employees
SET Salary = Salary * 1.1
WHERE Salary < 30
```


Query 2


```
UPDATE Employees
SET Salary = Salary * 1.15
WHERE Salary >= 30
```

  - If the statements are issued in this order, some employees may get a double raise!

Yay! ☺

Remember SQL can be Fun!!

TABLE CREATION, DATA MANIPULATION. QUERY WRITING

SQL INTRODUCTION – QUERY WRITING (SELECT)

So What's This For?

- You have got your tables/Database
- You have got data in your database
- Now lets start asking questions about our data

SQL as a query language

- Remember - SQL expresses queries in a declarative way
  - Queries focus on the result, not how to obtain it
    - Queries are translated by the query optimiser into the procedural language internal to the DBMS
    - The DBMS Optimises the query and executes it against the database

- The programmer should focus on readability, not on efficiency
    - In most cases! – only return the results you need

Select

SQL Commands Overview – **SELECT** x,y,z **FROM**

- **SELECT** is used to retrieve data from the database
  - Remember SQL is Declarative
    - You don't tell the DBMS how to get the data only about the data you want
- Syntax:

  SELECT <attributes> **FROM** <tableName> **WHERE** <conditions>

  - Also called a **SFW** query

Examples:

Query	Purpose
SELECT StudentID, StudentName FROM Student;	Select **specific** columns from **all** tuples from Student

SQL Commands Overview – **SELECT** x,y,z **FROM**

- Syntax (abbreviated)

  SELECT AttrExpr [[as] Alias {, AttrExpr [[as] Alias]}

  FROM <Table> [[as] Alias] {**JOIN** <Table2> [[as]] Alias]}

  [**WHERE** Condition]

  - < > - the part needs to be replaced by the actual table of interest
  - [ ] - the part is optional

- The **SELECT** clause defines the target list of attributes/values to be returned
- The **FROM** keyword defines the tables used by the query to obtain the attribute values
- The **WHERE** clause Is included to determine which tuples should be retrieved
  - It specifies the conditions each tuple must match in order to be included in the final result

SQL Commands Overview – SELECT x,y,z FROM

- When writing a query in SQL, you need to know the table schema (the attribute names) and what needs to be retrieved

- **Keywords** (SELECT, WHERE etc) are not case sensitive but strings of text may be case sensitive depending on DBMS settings

- A Simple test:

SQLQuery1.sql - DOUG-PC...

```
1 SELECT 'sensitive' WHERE 'S' <> 's';
2 SELECT 'insensitive' WHERE 'S' = 's';
```

Results | Messages

	(No column name)

	(No column name)
1	insensitive

In the above query 'S' = 's' returned true so MS-SQL in this instance is **case insensitive**

SQL Commands Overview – SELECT x,y,z FROM

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"

SQL Commands Overview – SELECT x,y,z FROM

- SQL Queries return a new relations that are composed of the attributes used in the query

Students

StudID	Name
1	Mei
2	Phil

```
SELECT Name FROM Students
```


Results

Name
Mei
Phil


```
SELECT * FROM Students WHERE Name = 'Phil'
```

Students

StudID	Name
1	Mei
2	Phil

Results

StudID	Name
2	Phil

SQL Commands Overview – SELECT x,y,z FROM

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
FROM Product
WHERE  Category = 'Gadgets'
```


PName	Price
Gizmo	$19.99
Powergizmo	$29.99

"selection" AND "projection"

SQL Commands Overview – **SELECT** x,y,z **FROM**

- Notation

Input Schema

Product(PName, Price, Category, Manfacturer)


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


Answer(PName, Price, Manfacturer)

Results Output Schema

WHERE Condition – Predicates

- **Predicate**:
  - These are the expressions in the WHERE clause
  - They evaluate to TRUE, FALSE, or UNKNOWN for each row of data tested
  - Predicates are used in the search condition of WHERE clauses and HAVING clauses, the JOIN conditions of FROM clauses, and other constructs where a Boolean value is required

- Predicates make use of Equality (=) and Inequality (<>)
  - See next slide for list of options

- BETWEEN… AND…
  - price **BETWEEN** 100 and 150
  - price >= 100 **AND** price <= 150

SQL Commands Overview: **WHERE**

- Logical conditions often used to filter results:
  - Use these to test values and combine tests with **AND/OR** as required

Condition	Description
=	Equals
<>	Not equal
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
LIKE	Partial matches (string comparison). Use '_' for single unknown character Use '%' for zero or more unknown characters: LIKE '%a%' returns all words containing 'a'
NOT LIKE	Not like the partial matches
IS NULL	Test an attribute value is empty (ie, Null)
IS NOT NULL	Test an attribute has a value (ie, not null)

Really Important

SQL Commands Overview: Duplicates - **DISTINCT**

- To remove duplicate records use **DISTINCT**

```
SELECT City
FROM Departments
```

same as:

```
SELECT ALL City
FROM Departments
```

City
London
Toulouse
Brighton
London
San Jose

```
SELECT DISTINCT City
FROM Departments
```

City
London
Toulouse
Brighton
San Jose

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

SQL Commands Overview: Duplicates - **DISTINCT**

- To remove duplicate records use **DISTINCT**

```
SELECT Surname
FROM Employees
```

Results

	Surname
1	Jackson
2	Brown
3	White
4	Green
5	Neri
6	Chen
7	Brown
8	Bradshaw


```
SELECT DISTINCT Surname
FROM Employees
```

Results

	Surname
1	Bradshaw
2	Brown
3	Chen
4	Green
5	Jackson
6	Neri
7	White

SQL Commands Overview: Renaming - **AS**

- Columns in a select query can also be manipulated and **renamed** using the keyword '**AS**'

```
SELECT FirstName + ' ' + Surname AS Name, Salary
FROM Employees
```


Results | Messages

	Name	Salary
1	Alice Jackson	46
2	Charles Brown	80
3	Charles White	36
4	Gus Green	40
5	Jackson Neri	45
6	Laurence Chen	73
7	Mary Brown	45
8	Pauline Bradshaw	40

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

SQL Commands Overview: **ORDER BY**

- The **ORDER BY** re-orders the output of the query based on one or more selected attributes
  - **ORDER BY** Attr1 [**ASC**| **DESC**], …Attr_n [**ASC**| **DESC**]
    - Order By is ascending unless you specify the DESC keyword
    - ASC = ASCENDING, DESC = DESCENDING

- Example
  - List the contents of Automobiles in descending order of make and model: (order model if make is the same)


```
SELECT *
FROM Automobiles
ORDER BY Make DESC,
Model DESC
```

CarRegNo	Make	Model	DriverID
GHI789	Lancia	Delta	PZ10124436B
DEF456	BMW	Z3	VR2030030Y
ABC123	BMW	323	VR2030030Y
BBB421	BMW	316	MI2020030U

Drivers (FirstName, Surname, DriverID)
Automobiles (CarRegNo, Make, Model,

SQL Commands Overview: **ORDER BY**

- The **ORDER BY** re-orders the output of the query based on one or more selected attributes
  - **ORDER BY** Attr1 [**ASC**| **DESC**], …Attr_n [**ASC**| **DESC**]
    - Order By is ascending unless you specify the DESC keyword
    - ASC = ASCENDING, DESC = DESCENDING


```
SELECT * FROM Employees
ORDER BY Employees.Surname ASC;
```


Is the same as:

```
SELECT * FROM Employees
ORDER BY Employees.Surname;
```

(ascending is assumed if not specified)

Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Pauline	Bradshaw	Administration	75	40	Brighton
2	Mary	Brown	Administration	10	45	London
3	Charles	Brown	Planning	14	80	London
4	Laurence	Chen	Planning	7	73	Worthing
5	Gus	Green	Administration	20	40	Oxford
6	Alice	Jackson	Production	75	46	Toulouse
7	Jackson	Neri	Distribution	16	45	Dover
8	Charles	White	Production	20	36	Toulouse

SQL Commands Overview: **ORDER BY**

- The **ORDER BY** re-orders the output of the query based on one or more selected attributes
  - **ORDER BY** Attr1 [**ASC**| **DESC**], …Attr_n [**ASC**| **DESC**]
    - Order By is ascending unless you specify the DESC keyword
    - ASC = ASCENDING, DESC = DESCENDING


```
SELECT * FROM Employees
ORDER BY Employees.Surname ASC,
Employees.FirstName ASC;
```


Is the same as:

```
SELECT * FROM Employees
ORDER BY Employees.Surname,
Employees.FirstName;
```

(ascending is assumed if not specified)

Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Pauline	Bradshaw	Administration	75	40	Brighton
2	Charles	Brown	Planning	14	80	London
3	Mary	Brown	Administration	10	45	London
4	Laurence	Chen	Planning	7	73	Worthing
5	Gus	Green	Administration	20	40	Oxford
6	Alice	Jackson	Production	75	46	Toulouse
7	Jackson	Neri	Distribution	16	45	Dover
8	Test	Test	Planning	7	NULL	Worthing

SQL Commands Overview: **SELECT TOP n**

- The [TOP n <attr list>] clause can be used to limit the number of matching results returned
  - Mainly used with ordered results (enables paging through results) - some system require the results to be ordered to use this function.

```
SELECT TOP 3 * FROM Employees
ORDER BY Surname
(ascending is assumed if not specified)
```


Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Pauline	Bradshaw	Administration	75	40	Brighton
2	Charles	Brown	Planning	14	80	London
3	Mary	Brown	Administration	10	45	London

Is the same as:

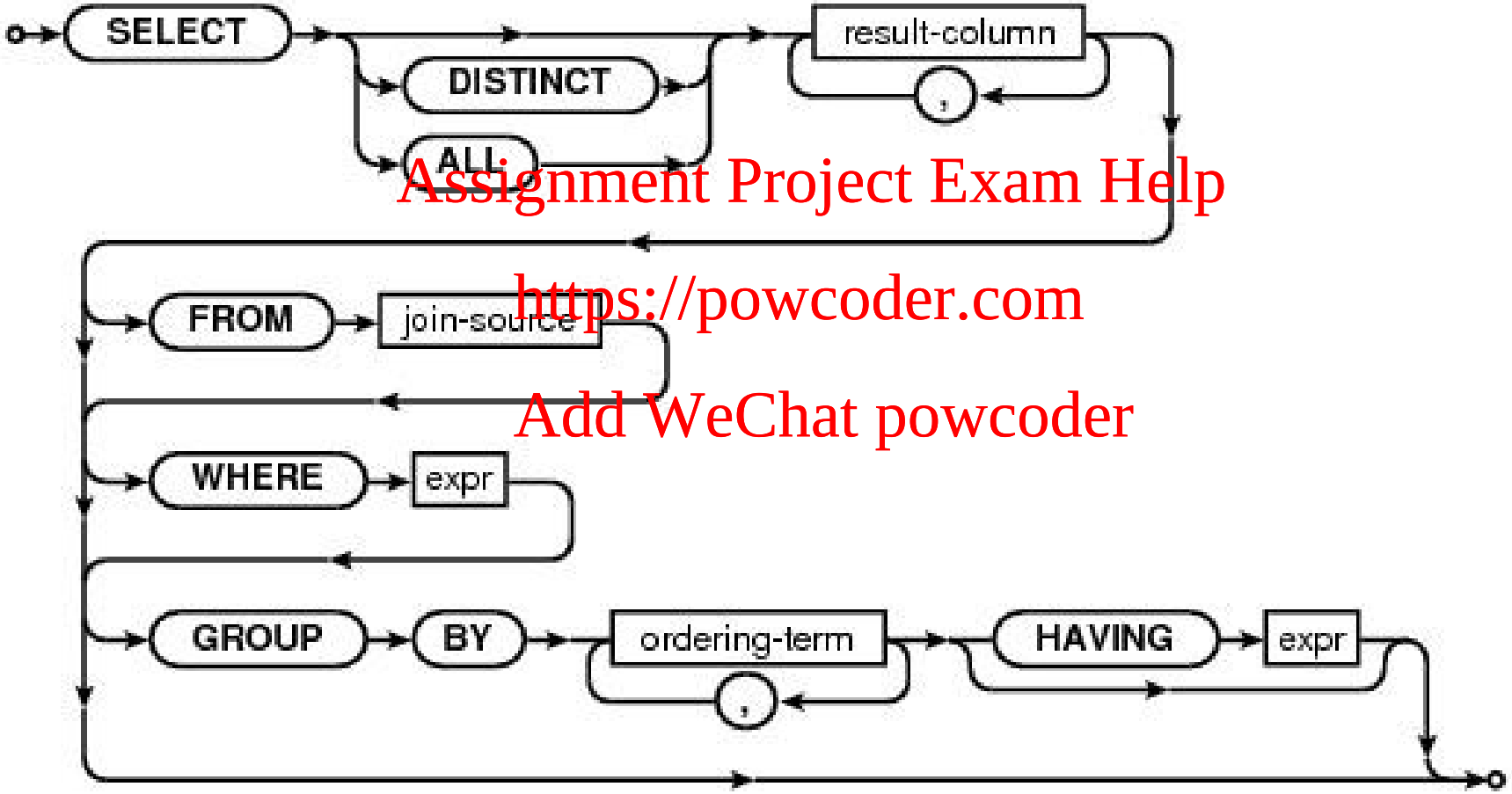
```
SELECT TOP 3 FirstName, Surname
FROM Employees
ORDER BY Surname ASC
```

Results | Messages

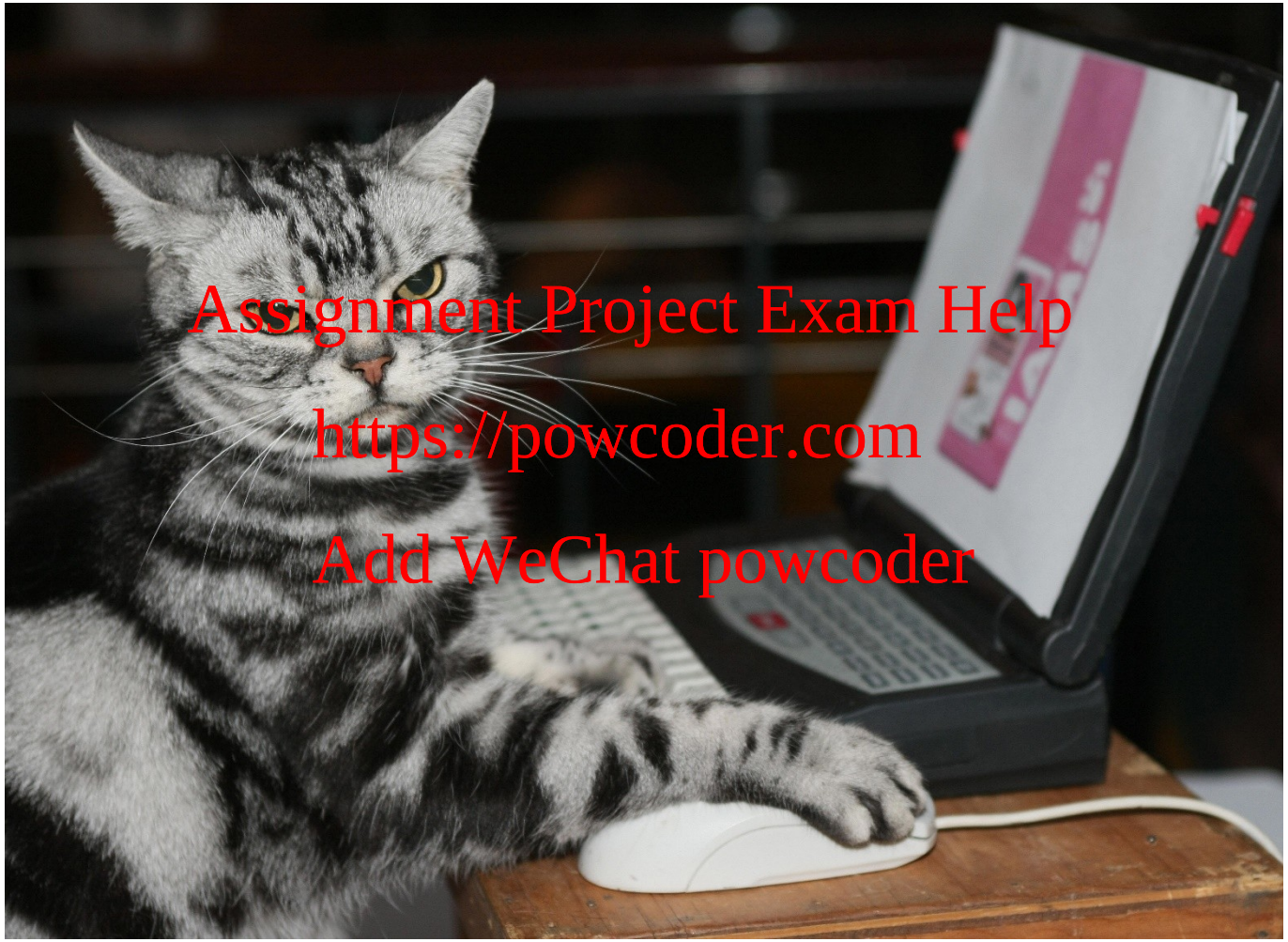
	FirstName	Surname
1	Pauline	Bradshaw
2	Charles	Brown
3	Mary	Brown

SQL – SELECT Syntax

- The SQL SELECT Query is the most often used SQL statement



OK Sounds a Bit Complicated



How to write a SQL database query

- You have all the data in the database.
- Hopefully its systematic and normalized
- You just need to figure out how to ask the question to get what you need.
- A UML/Database Diagram WILL HELP.
  - Simple single table questions are straight forward
  - On multiple table questions you need JOIN the relations together with JOIN conditions
  - Follow the UML/Database Diagram relationship lines from FK → PK

- Apply filtering conditions
  - WHERE x = y AND z like ‘me’

Hint

- Work backwards!!

- Go to the column(s) that your after and figure out how to get the data you want!!
  - What columns do I need (**SELECT** x, y, z)
  - What rows am I interested in (**WHERE** a =? AND b = ?)
  - What Table(s) do these come from? (**FROM** xxx)

SQL Commands Overview: **WHERE**

Employees

RegNo	FirstName	Surname	Dept	Office	Salary	City
001	Mary	Brown	Administration	10	45	London
002	Charles	White	Production	20	36	Toulouse
003	Gus	Green	Administration	20	40	Oxford
004	Jackson	Neri	Distribution	16	45	Dover
005	Charles	Brown	Planning	14	80	London
006	Laurence	Chen	Planning	7	73	Worthing
007	Pauline	Bradshaw	Administration	75	40	Brighton
008	Alice	Jackson	Production	75	46	Toulouse

Departments

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Jose

SQL Commands Overview: **WHERE**

- Display names of employees who are in production department earning more than 40 thousand.

```
SELECT FirstName, Surname
FROM employees
WHERE dept = 'Production' AND salary > 40;
```


Results | Messages

	FirstName	Surname
1	Alice	Jackson

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

SQL Commands Overview: **WHERE**

- Find all the information of the employees named Brown:

```
SELECT *
FROM Employees
WHERE Surname = 'Brown'
```


Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Charles	Brown	Planning	14	80	London
2	Mary	Brown	Administration	10	45	London

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

SQL Commands Overview: **WHERE**

- Find the first names and surnames of the employees who work in office number 20 of the Administration department:

```
SELECT FirstName, Surname
FROM Employees
WHERE Office = '20' AND
      Dept = 'Administration'
```


Results | Messages

	FirstName	Surname
1	Gus	Green

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

SQL Commands Overview: **WHERE**

- Find the first names and surnames of the employees who work in either the Administration or the Production department:

```
SELECT FirstName, Surname
FROM Employees
WHERE Dept = 'Administration' OR
      Dept = 'Production'
```


Results | Messages

	FirstName	Surname
1	Alice	Jackson
2	Charles	White
3	Gus	Green
4	Mary	Brown
5	Pauline	Bradshaw

Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

SQL Commands Overview: **WHERE**

- If no brackets are used then 'AND' is evaluated before 'OR'
  - Find the first names of the employees named Brown who work in the Administration department or the Production department:

```
SELECT FirstName
FROM Employees
WHERE Surname = 'Brown' AND
    (Dept = 'Administration' OR
     Dept = 'Production')
```


Results

	FirstName
1	Mary

If no brackets are used:

Results

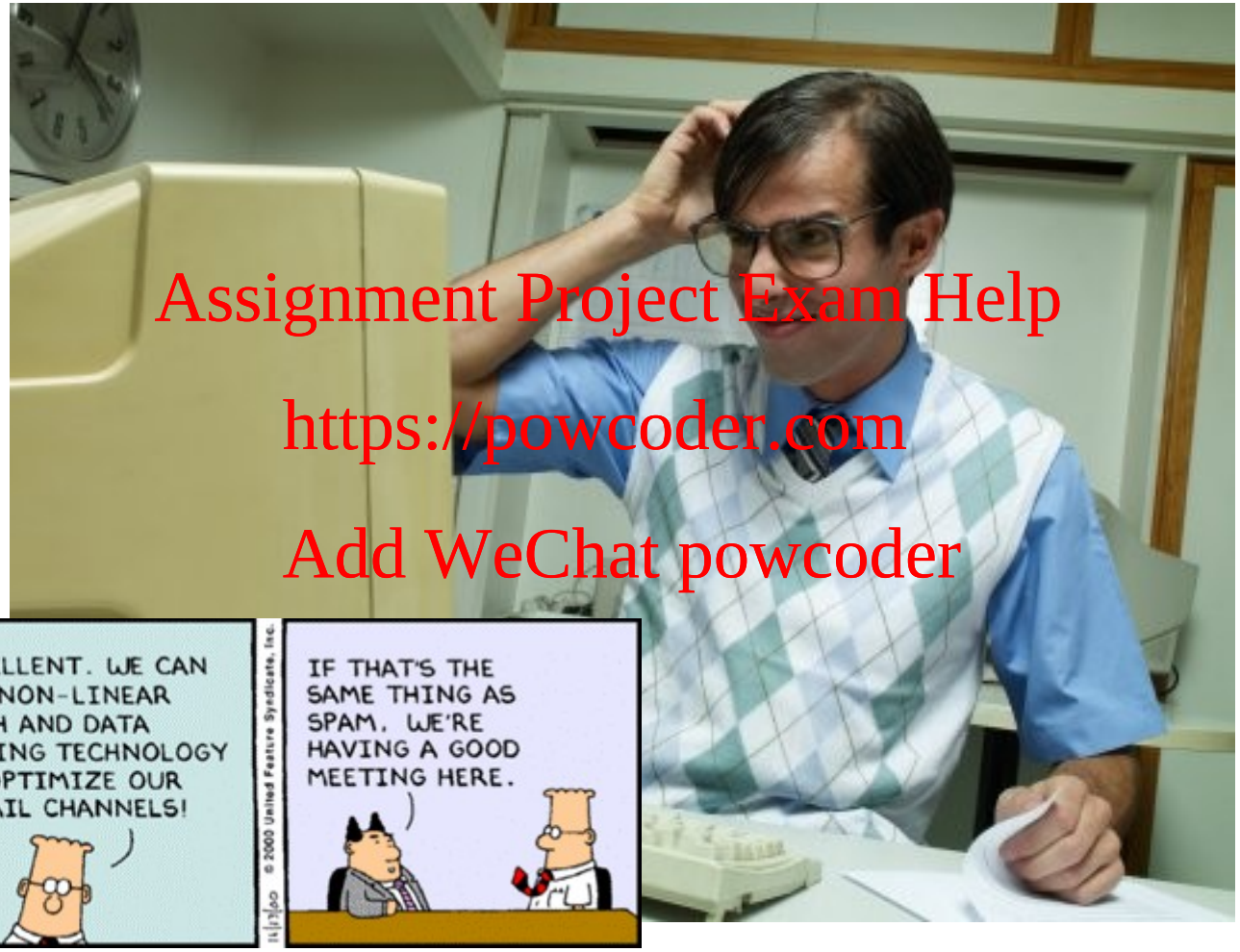
	FirstName
1	Alice
2	Charles
3	Mary

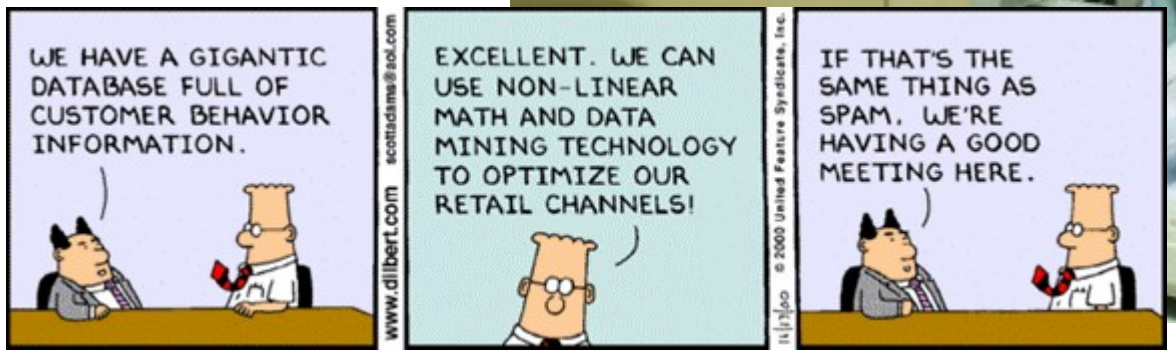
Table Schemas:

Employees (RegNo, FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

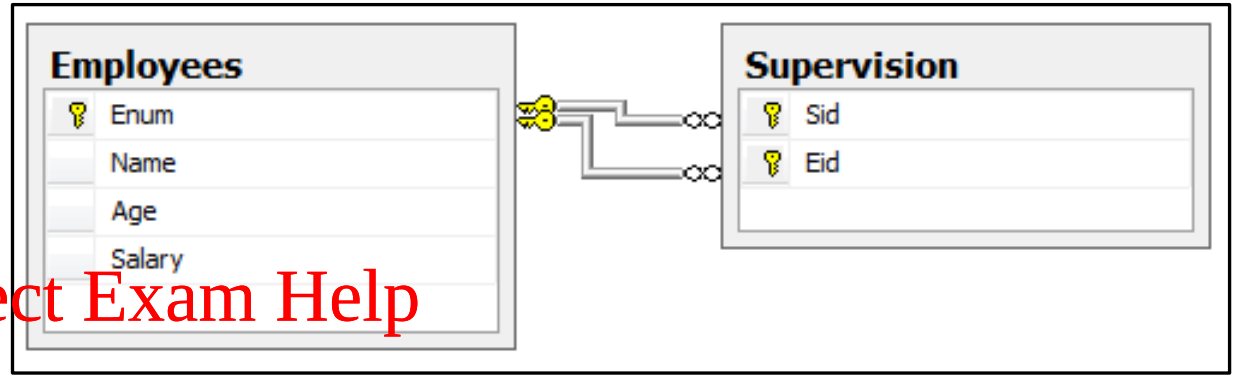
Database Setup





SQL Query – SELECT (Database Setup)

```
CREATE TABLE Employees(
        Enum char(3),
        Name varchar(100) NOT NULL,
        Age int NULL,
        Salary int NULL,
CONSTRAINT employeePk PRIMARY KEY (Enum)
);
```




```
CREATE TABLE Supervision(
        Sid char(3),
        Eid char(3),
        CONSTRAINT supervisorFk FOREIGN KEY (Sid) REFERENCES Employees(Enum),
        CONSTRAINT employeeFk FOREIGN KEY (Eid) REFERENCES Employees(Enum),
        CONSTRAINT supervisorFk PRIMARY KEY (Sid, Eid)
);
```

SQL Query – SELECT (Database Setup)

Employees

Enum	Name	Age	Salary
101	Mary Smith	34	40
103	Mary Bianchi	23	35
104	Luigi Neri	38	61
105	Nico Bini	44	38
210	Marco Celli	49	60
231	Siro Bisi	50	60
252	Nico Bini	44	70
301	Steve Smith	34	70
375	Mary Smith	50	65

Supervision

Sid	Eid
210	101
210	103
210	104
231	105
301	210
301	231
375	252

Logical Schema

Employees(<u>Enum</u>, Name, Age, Salary)

PK: Enum

Supervision(<u>Sid, Eid</u>)

PK: (Sid, Eid)

FK: Sid ~> Employees(Enum)

FK: Eid ~> Employees(Enum)

SQL Query – SELECT (Database Setup)

```
INSERT INTO Employees VALUES
('101', 'Mary Smith',      34,    40),
('103', 'Mary Bianchi',    23,    35),
('104', 'Luigi Neri',      38,    61),
('105', 'Nico Bini',       44,    38),
('210', 'Marco Celli',     49,    60),
('231', 'Siro Bisi',       50,    60),
('252', 'Nico Bini',       44,    70),
('301', 'Steve Smith',     34,    70),
('375', 'Mary Smith',      50,    65);
```



```
INSERT INTO Supervision VALUES
(210,    101),
(210,    103),
(210,    104),
(231,    105),
(301,    210),
(301,    231),
(375,    252);
```

Example 1 – Basic SELECT

Relational Schema:
Employees(Enum, Name, Age, Salary)
Supervision(Sid, Eid)

- Think: Asking a single table for an answer.
- Find the number, the name and the age of each employee earning more than 40 thousand
  - Put required attributes in the **SELECT** clause:
    - the query asks for only 3 columns, so
    - **SELECT** Enum, Name, Age
  - Put required tables to get the attributes in the **FROM** clause:
    - It needs only one table to answer:

      SELECT Enum, Name, Age
      FROM Employees
  - Put filters in the **WHERE** clause:
    - **WHERE** salary > 40, so:

When Query is executed:

Enum	Name	Age
104	Luigi Neri	38
210	Marco Celli	49
231	Siro Bisi	50
252	Nico Bini	44
301	Steve Smith	34
375	Mary Smith	50

```
SELECT Enum, Name, Age
        FROM Employees
        WHERE salary > 40
```

Example database – stored in a DBMS

Employees

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	75	46	Toulouse

Departments

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Jose

Example database – stored in a DBMS

```
CREATE TABLE Departments(
 DeptName varchar(50)  PRIMARY KEY ,
 Address varchar(50) NULL,
 City varchar(50) NULL
);
```

```
INSERT INTO Departments VALUES
('Administration', 'Bond Street', 'London'),
('Distribution', 'Pond Road', 'Brighton'),
('Planning','Bond Street', 'London'),
('Production', 'Rue Victor Hugo', 'Toulouse'),
('Research', 'Sunset Street', 'San Jose');
```


```
CREATE TABLE dbo.Employees(
FirstName varchar(50) NOT NULL,
Surname varchar(50) NOT NULL,
Dept varchar(50) REFERENCES  Departments(DeptName),
Office varchar(50) NULL,
Salary int NULL,
City varchar(50) NULL,
PRIMARY KEY (FirstName, Surname)
);
```



```
INSERT INTO Employees VALUES
('Mary', 'Brown', 'Administration', '10', 45, 'London'),
('Charles', 'White', 'Production', '20', 36, 'Toulouse'),
('Gus', 'Green', 'Administration', '20', 40, 'Oxford'),
('Jackson', 'Neri', 'Distribution', '16', 45, 'Dover'),
('Charles', 'Brown', 'Planning', '14', 80,          'London'),
('Laurence', 'Chen', 'Planning', '7', 73, 'Worthing'),
('Pauline', 'Bradshaw', 'Administration', '75', 40, 'Brighton'),
('Alice', 'Jackson', 'Production', '75', 46, 'Toulouse');
```

Simple Query – SELECT *

- Select all columns of all tuples of a table
  - **SELECT** * **FROM** tableName

- Example
  - Find the salaries of ALL employees:


```
SELECT * FROM Employees
```

Results | Messages

FirstName	Surname	Dept	Office	Salary	City
Alice	Jackson	Production	75	46	Toulouse
Charles	Brown	Planning	14	80	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Laurence	Chen	Planning	7	73	Worthing
Mary	Brown	Administration	10	45	London
Pauline	Bradshaw	Administration	75	40	Brighton

Simple Query – conjunction ("and") Condition

- Find the first names AND surnames of the employees who work in office number 20 of the Administration department:

```
SELECT FirstName, Surname
FROM Employees
WHERE Office = '20' AND
Dept = 'Administration'
```


Execution output:

FirstName	Surname
Gus	Green

Table Schemas:

Employees (FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

Condition – disjunction (“or”)

- Find the first names and surnames of the employees who work in either the Administration or the Production department:

```
SELECT FirstName, Surname
FROM Employees
WHERE Dept = 'Administration' OR
      Dept = 'Production'
```


Execution output:

FirstName	Surname
Mary	Brown
Charles	White
Gus	Green
Pauline	Bradshaw
Alice	Jackson

Table Schemas:

Employees (FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (DeptName, Address, City)

Condition – Logic Operator Precedence

- Find the first names of the employees named Brown who work in the Administration department or the Production department:
  - If not using brackets, 'AND' is evaluated before 'OR' (Left -> Right)

```
SELECT FirstName
FROM Employees
WHERE Surname = 'Brown' AND
    (
        Dept = 'Administration' OR
        Dept = 'Production'
    )
```


What is the difference if the condition is the following?

```
Surname = 'Brown' AND
Dept = 'Administration'
OR Dept = 'Production'
```

Execution output:

FirstName
Mary

Table Schemas:

Employees (<u>FirstName, Surname,</u> Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Departments (<u>DeptName</u>, Address, City)

Condition – Pattern Matching: **LIKE** operator

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


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

Condition – Pattern Matching: **LIKE** operator

- Equality (=) is often too strong
  - ‘London’ <> ‘London, UK’ <> ‘Richmond, London’
  - Need a way to match sub-strings
- Find the employees with surnames that have ‘r’ as the second letter and end in ‘n’:


```
SELECT *
FROM Employees
WHERE Surname LIKE '_r%n'
```

In patterns,
- ‘_’ means any single character and
- ‘%’ means any string of characters

FirstName	Surname	Dept	Office	Salary	City
Mary	**Brown**	Administration	10	45	London
Gus	**Green**	Administration	20	40	Oxford
Charles	**Brown**	Planning	14	80	London

Condition – Predicates for NULL values

- Null values may mean that:
  - a value is not applicable
  - a value is applicable but unknown
  - it is unknown whether a value is applicable or not

- SQL-89 used a two-valued logic
  - a comparison with NULL returned FALSE

- SQL-2 uses a three-valued logic
  - a comparison with NULL returns UNKNOWN

- To test for null values, you must use
  - Attribute IS NULL
  - Attribute IS NOT NULL

  - Do NOT USER Attribute='NULL' or Attribute=NULL