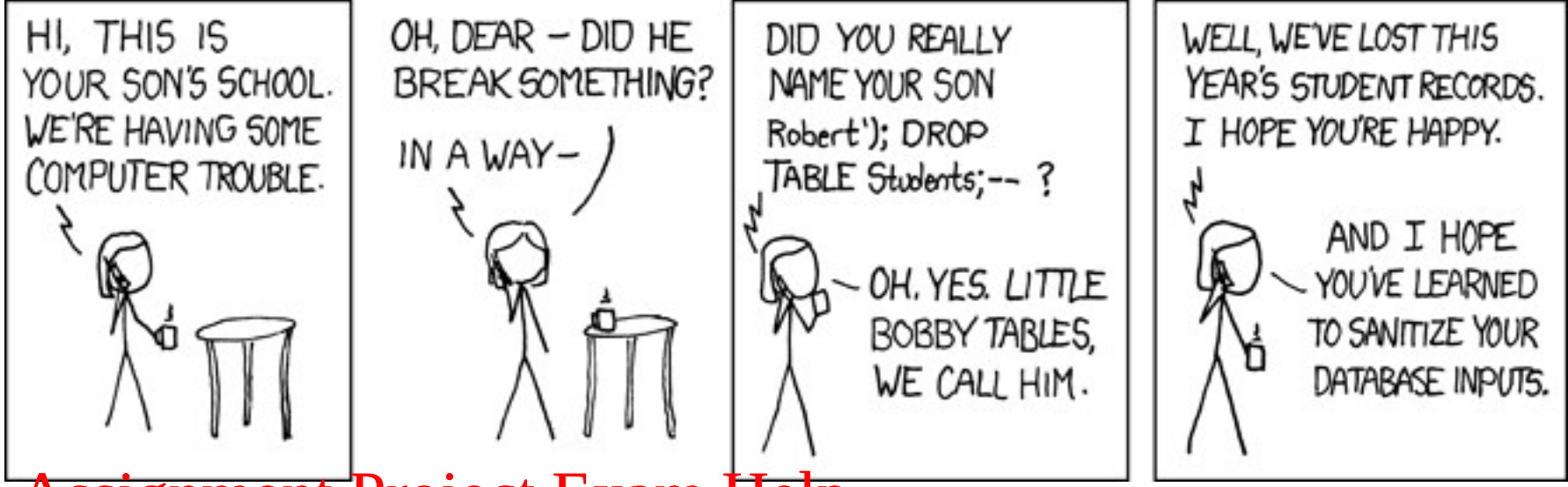


Database Fundamentals

SQL JOINS

SQL – 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 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)

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

- SQL Queries return a new relations that are composed of the attributes used in the query
- Relations can be combined using the key word **JOIN** to create a new relation
  - This relation can have a combination of the attributes from the tables used in the query


```
SELECT * FROM TableA JOIN TableB ON StudID = Stud
```


TableA

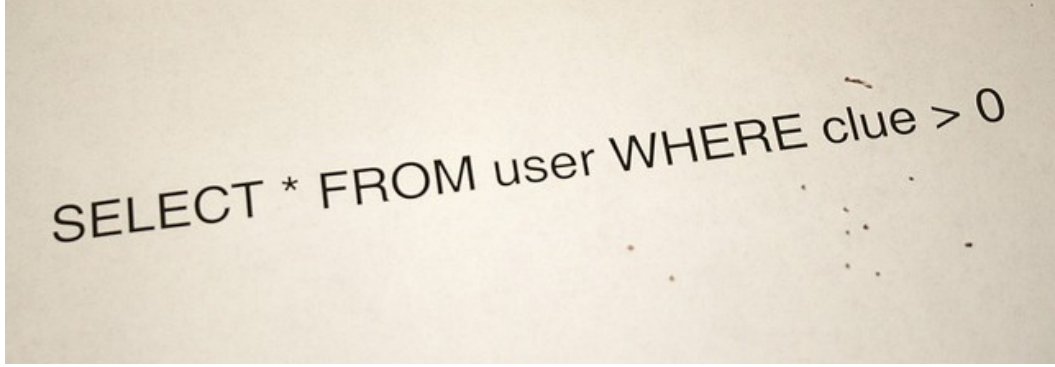
StudID	Name
001	Pete
002	Jane

TableB

Stud	Book
001	Databases
002	Programming

Results

StudID	Stud	Name	Book
001	001	Pete	Databases
002	002	Jane	Programming



SQL – Table JOINS

TURNING DATA INTO INFORMATION

SQL – Sample Database

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

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

SQL – Sample Database

```
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');
```

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

SQL – FROM Table Variables

- Used to improve readability of the query or rename a relation
  - Useful where the relation is used more than once to distinguish between different instances of the relation

```
SELECT Employee.firstname, Department.City
FROM Departments JOIN Employees
ON Departments.DeptName = Employees.Dept
WHERE Surname='Brown'
```

Explicitly stating the table attribute being used


```
SELECT e.firstname, d.City
FROM Departments AS d JOIN Employees AS e
ON d.DeptName = e.Dept
WHERE Surname='Brown'
```

```
SELECT e.firstname, d.City
FROM Departments d JOIN Employees e
ON d.DeptName = e.Dept
WHERE Surname='Brown'
```

Explicitly stating the table attribute being used with abbreviations

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

SQL Query – Cartesian product

- This operator is basic for evaluating many other operators
- It combines two tables and gives every possible combinations of the tuples of the two tables
  - The Cartesian product of r and s following is given table z

r

A	B	C
1	2	3
1	2	4

s

D	E
a	b
a	c

z

A	B	C	D	E
1	2	3	a	b
1	2	3	a	c
1	2	4	a	b
1	2	4	a	c

- SQL:

```
SELECT * FROM r, s
SELECT * FROM r, s WHERE 1 = 1
SELECT * FROM r JOIN s ON 1 = 1
SELECT * FROM r INNER JOIN s ON 1 = 1
```

Meaningful criteria in the WHERE clauses turn the output of a Cartesian Product into something **Useful**!

SQL Query – **JOIN** Query

- The meaning of the query:
  - Build the Cartesian product of all tables in the **FROM** clause
  - Consider only the **rows** that satisfy the conditions in the **WHERE** clause
  - Evaluate the **attribute list** for each of the rows returned
  - The whole point is to recombine data distributed across multiple tables to give a more "complete" picture
    - A normalised database stores data across many different tables (space efficient and to avoid anomalies) that must be recombined when retrieving information

SQL Query – **JOIN** Query

- Example:
  - Display a table that contains all the information of every employee and all the information of the employee's department.
    - The requested information is stored in two tables: **Departments** and **Employee**
    - Employees are related to departments via the Department name they work in
    - The two tables need to be 'combined' to form a new wider result table
  - Joining tables involves combining rows of data from each table based on a join condition
    - A Join condition is a comparison between a set of one or more columns that have contain the same or related information.
    - In the application, the Dept of Employees have the same meaning as DeptName of Departments. So the condition is Departments.DeptName = Employees.Dept
    - DeptName and Dept are called the join attributes.

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

SQL Query – **JOIN** Query Mechanism

- In a normalised database design data is stored across multiple relations and connected by foreign key relationships
- To re-create the information captured in the data relations must be merged together based on conditions that specify which columns in one relation match the same data in tuples of the other
  - This ordinarily follows matching table2 FK → table1 PK attributes

In our sample database design:
Departments.DeptName = Employees.Dept
DeptName and Dept are called **the join attributes**

```
SELECT * FROM Employees
JOIN Departments
ON Employees.Dept = Departments.DeptName
```

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

SQL Query – **JOIN** Query Mechanism

- The SQL query – this type of join is also called an **INNER JOIN**.

```
SELECT * FROM Departments, Employees WHERE Employees.Dept = Departments.DeptName
SELECT * FROM Departments JOIN Employees ON Employees.Dept = Departments.DeptName
SELECT * FROM Departments INNER JOIN Employees ON Employees.Dept = Departments.DeptName
```


The query is evaluated following the algorithm below:

```
for each employee x in Employees
        for each department y in Departments
                if x.Dept == y.Deptname then
                        output x + y tuple
                end if
        end for y
end for x
```


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

- If an employee tuple does not match any department tuple on the join attributes, the employee tuple is said **dangling**.
- **Dangling** tuples are dropped from the output

SQL Query – **JOIN** Query Mechanism

- Joins between tables can also make use renaming to improve readability of longer/large queries:

```
SELECT * FROM Employees
JOIN Departments
ON Employees.Dept = Departments.DeptName
```



```
SELECT * FROM Employees AS e
JOIN Departments AS d
ON e.Dept = d.DeptName
```

```
SELECT * FROM Employees e
JOIN Departments d
ON e.Dept = d.DeptName
```

SQL Query – **JOIN** Query output

- The output of the SQL query on previous slide is (we assume that City in Departments is renamed to Dcity):

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

How to decide the number of columns and the column names? How to decide the number of tuples in the output?

What happened to the Research department? What is the rule behind?

Why some department tuples are duplicated? What is the rule behind?

why did we need the assumption about City?

SQL Query – **JOIN** Query with selection and projection

- For each employee with Surname Brown, list the firstname and the address of his/her department
  - The information required are in two tables, name is in Employees and address of the department is in Departments.
  - The join condition is (same as a previous example) DeptName=Dept

- The SQL query

```
SELECT firstname, address
FROM Departments JOIN Employees
ON DeptName=Dept
WHERE Surname = 'Brown'
```

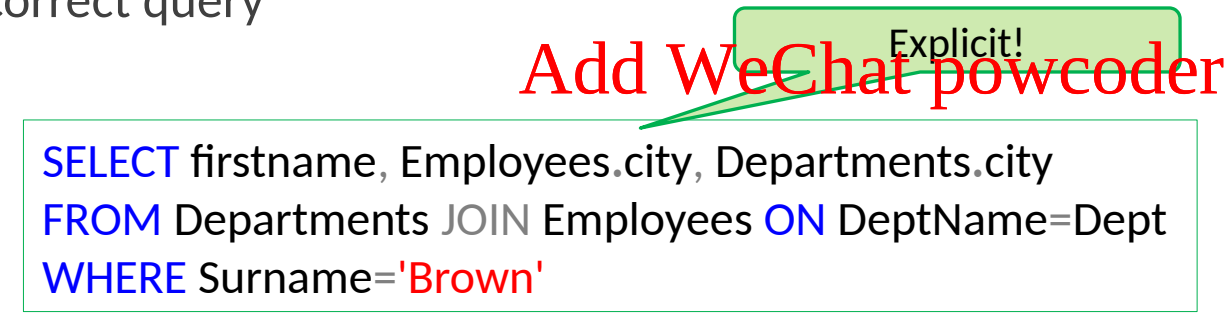

- Compare this with

```
SELECT firstname, address
FROM Departments JOIN Employees
ON DeptName=Dept
```

SQL Query – **JOIN** Query duplicate column names

- For each employee with Surname Brown, list his/her firstname, the city where he/she lives and the city in which he/she works
- Incorrect SQL query

- The correct query



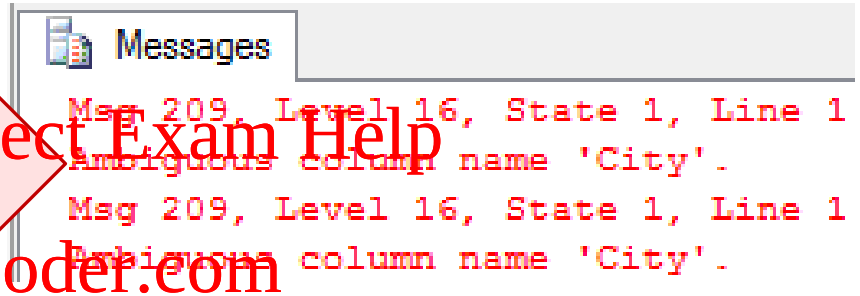
- Use table name to distinguish duplicate column names

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

SQL Query – **JOIN** Query duplicate column names

- What if the output has ambiguous attribute names?
- Example:

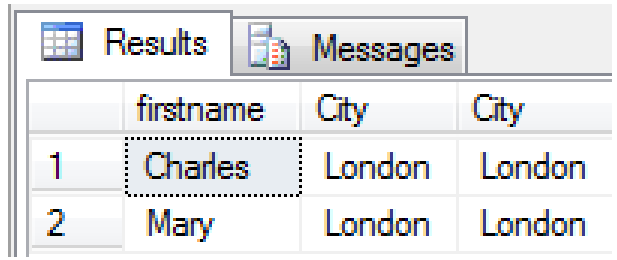
```
SELECT firstname, City, City
FROM Departments JOIN Employees
ON DeptName = Dept
WHERE Surname='Brown'
```




```
SELECT firstname, Departments.City,
Employees.City
FROM Departments JOIN Employees
ON DeptName = Dept
WHERE Surname='Brown'
```

OR

```
SELECT firstname, d.City, e.City
FROM Departments d JOIN Employees e
ON d.DeptName = e.Dept
WHERE Surname='Brown'
```



	firstname	City	City
1	Charles	London	London
2	Mary	London	London

SQL Query – **JOIN** Query table alias

- Use table alias to simplify a query.
  - The following queries are equivalent:

```
SELECT firstname, Employees.city, Departments.city
FROM Departments JOIN Employees ON DeptName=Dept
WHERE Surname='Brown'
```

```
SELECT firstname, E.city, D.city
FROM Departments AS D JOIN Employees AS E ON DeptName=Dept
WHERE Surname='Brown'
```

```
SELECT firstname, E.city, D.city
FROM Departments D JOIN Employees E ON DeptName=Dept
WHERE Surname='Brown'
```


Note that “firstname” is unambiguous hence it does not need to be qualified with the table from which it comes from

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

JOIN Query – a different but a more common way

- Find the names of the employees and the cities in which they work:
  - The following two queries are equivalent

```
SELECT FirstName, Surname, D.City
FROM Employees E, Departments D
WHERE E.Dept = D.DeptName;
```



```
SELECT FirstName, Surname, D.City
FROM Employees JOIN Departments D
ON Dept = DeptName;
```

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

SQL Query – **JOIN** Query Multiple Matches

- Find the head of department for each employee

Employees

Employee	Dept
Smith	sales
Black	production
White	production

Departments

Dept	Head
sales	Mori
production	Brown
production	Pink

JOINS on attributes that are not unique may result in multiple matching records

```
SELECT * FROM Departments D JOIN Employees E
ON D.Dept = E.Dept
```


Results | Messages

	Dept	Head	Employee	Dept
1	production	Brown	Black	production
2	production	Pink	Black	production
3	sales	Mori	Smith	sales
4	production	Brown	White	production
5	production	Pink	White	production

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

SQL Query – **JOIN** Query Dangling Tuples

- Find the head of department for each employee

Employees

Employee	Dept
Smith	sales
Black	production
White	production

Departments

Dept	Head
production	Brown
purchasing	Pink


```
SELECT * FROM Departments D JOIN Employees E
ON D.Dept = E.Dept
```

Results | Messages

	Dept	Head	Employee	Dept
1	production	Brown	Black	production
2	production	Brown	White	production

Dangling tuples do not end up in the final result set of INNER JOINs

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

SQL Query – **JOIN** Query Dangling Tuples

- Find the head of department for each employee

Employees

Employee	Dept
Smith	sales
Black	production
White	production

Departments

Dept	Head
marketing	Brown
purchasing	Pink


```
SELECT * FROM Departments D JOIN Employees E
ON D.Dept = E.Dept
```

Results | Messages

Dept	Head	Employee	Dept

In a join, all tuples may become **dangling** tuples and the JOIN outputs no tuples

Departments(DeptName, Head)
Employees(Employee, Dept)
FK(Dept) -> Departments(Dept)

JOIN Query – Multiple attributes in join condition

Cars

Reg	Dept	Owner	...
5694FR	75	Latour Hortense	
6544XY	75	Cordon Edouard	
6544XY	47	Mimault Bernard	
7122HT	75	Cordon Edouard	

Offences

Code	Date	Officer	Dept	Reg
143256	25/10/1992	567	75	5694FR
987554	26/10/1992	456	75	5694FR
987557	26/10/1992	456	75	6544XY
630876	15/10/1992	456	47	6544XY


```
SELECT Code, Date, Officer, O.Dept, O.Reg, Owner
FROM Offences O JOIN Cars C
ON O.Dept = C.Dept AND O.Reg = C.Reg
```

Code	Date	Officer	Dept	Reg	Owner	...
143256	25/10/1992	567	75	5694FR	Latour Hortense	
987554	26/10/1992	456	75	5694FR	Latour Hortense	
987557	26/10/1992	456	75	6544XY	Cordon Edouard	
630876	15/10/1992	456	47	6544XY	Mimault Bernard	

Outer Joins

JOIN Query – OUTER JOIN

- An **OUTER JOIN** is a variant of the join that keeps the dangling tuples in the result
  - Achieved by padding missing values with NULL where the tuples do not match in both relations
- Three variants
  - **LEFT OUTER JOIN**
    - only dangling tuples of the left of the join statement are padded with NULL values
    - LEFT JOIN | LEFT OUTER JOIN
  - **RIGHT OUTER JOIN**
    - only dangling tuples of the right of the join statement are padded with NULL values
    - RIGHT JOIN | RIGHT OUTER JOIN
  - **FULL OUTER JOIN**
    - dangling tuples of both sides of the join statement are padded with NULL values
    - FULL JOIN | FULL OUTER JOIN

JOIN Query – OUTER JOIN

- **INNER JOIN**
  - only matching tuples between tables are retained
  - JOIN | INNER JOIN

Table1

A	B
A1	B1
A2	B2
A4	B4

```
SELECT t1.a, t1.b, t2.c FROM
Table1 t1 INNER JOIN Table2 t2
ON  t1.a = t2.a
```

Table2

A	C
A1	C1
A2	C2
A3	C3

=

A	B	C
A1	B1	C1
A2	B2	C2

JOIN Query – OUTER JOIN

- **LEFT OUTER JOIN**
  - only dangling tuples of the **left** of the join statement are padded with NULL values
  - LEFT JOIN | LEFT OUTER JOIN

Table1

A	B
A1	B1
A2	B2
A4	B4

```
SELECT t1.a, t1.b, t2.c FROM
Table1 t1 LEFT OUTER JOIN Table2 t2
ON  t1.a = t2.a
```

Table2

A	C
A1	C1
A2	C2
A3	C3

=

A	B	C
A1	B1	C1
A2	B2	C2
A4	B4	NULL

JOIN Query – OUTER JOIN

- **RIGHT OUTER JOIN**
  - only dangling tuples of the **right** of the join statement are padded with NULL values
  - RIGHT JOIN | RIGHT OUTER JOIN

Table1

A	B
A1	B1
A2	B2
A4	B4

```
SELECT t1.a, t1.b, t2.c FROM
Table1 t1 RIGHT OUTER JOIN Table2 t2
ON  t1.a = t2.a
```

Table2

A	C
A1	C1
A2	C2
A3	C3

=

A	B	C
A1	B1	C1
A2	B2	C2
NULL	NULL	C3

Would = A3 if SELECT t2.a, t1.b, t2.c

JOIN Query – OUTER JOIN

- **FULL OUTER JOIN**
  - dangling tuples of **both sides** of the join statement are padded with NULL values
  - FULL JOIN | FULL OUTER JOIN

Table1

A	B
A1	B1
A2	B2
A4	B4

```
SELECT t1.a, t1.b, t2.c FROM
Table1 t1 FULL OUTER JOIN Table2 t2
ON  t1.a = t2.a
```

Table2

A	C
A1	C1
A2	C2
A3	C3

A	B	C
A1	B1	C1
A2	B2	C2
NULL	NULL	C3
A4	B4	NULL

Would = A3 if SELECT t2.a, t1.b, t2.c

JOIN Query – OUTER JOIN examples

Employees

Employee	Dept
Smith	sales
Black	production
White	production

Departments

Dept	Head
production	Mori
purchasing	Brown


```
SELECT * FROM
Employees E RIGHT OUTER JOIN Department D
ON D.Dept = E.Dept
```

Employee	Dept	Head
Black	production	Mori
White	production	Mori
NULL	purchasing	Brown

```
SELECT * FROM
Employees E LEFT OUTER JOIN Department D
ON D.Dept = E.Dept
```

Employee	Dept	Head
Smith	sales	NULL
Black	production	Mori
White	production	Mori

```
SELECT * FROM
Employees E FULL OUTER JOIN Department D
ON D.Dept = E.Dept
```

Employee	Dept	Head
Smith	sales	**NULL**
Black	production	Mori
White	production	Mori
NULL	purchasing	Brown

Departments(DeptName, Head)
Employees(Employee, Dept)
FK(Dept) -> Departments(Dept)

More outer Join Examples based on

Drivers

FirstName	Surname	DriverID
Mary	Brown	VR2030030Y
Charles	White	PZ1012436B
Marco	Neri	AP4544442R

Automobiles

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

```
CREATE TABLE Automobiles(
CarRegNo varchar(50) PRIMARY KEY,
Make varchar(50),
Model varchar(50),
DriverID varchar(50)
);
CREATE TABLE Drivers(
DriverID varchar(50) PRIMARY KEY,
FirstName varchar(50),
Surname varchar(50)
);
```



```
INSERT INTO Drivers (FirstName, Surname, DriverID)
VALUES
('Mary', 'Brown', 'VR2030030Y'),
('Charles', 'White', 'PZ1012436B'),
('Marco', 'Neri', 'AP4544442R');
INSERT INTO Automobiles(CarRegNo, Make, Model, DriverID)
VALUES
('ABC123','BMW','323','VR2030030Y'),
('DEF456','BMW','Z3','VR2030030Y'),
('GHI789','Lancia','Delta','PZ10124436B'),
('BBB421','BMW','316','MI2020030U');
```

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

SQL JOINS – LEFT OUTER JOINS

- LEFT OUTER JOIN
  - Retains all dangling tuples from the table on the left of the join statement

Employees

Employee	Dept
Smith	sales
Black	production
White	production

Departments

Dept	Head
production	Brown
purchasing	Pink


```
SELECT * FROM Employees E
LEFT OUTER JOIN Departments D
ON D.Dept = E.Dept
```

Results | Messages

	Employee	Dept	Dept	Head
1	Black	production	production	Brown
2	Smith	sales	NULL	NULL
3	White	production	production	Brown

Departments(DeptName, Head)
Employees(Employee, Dept)
FK(Dept) -> Departments(Dept)

JOIN Query – LEFT OUTER JOIN

- LEFT OUTER JOIN
  - Find the drivers with their cars, including the drivers without cars:

```
SELECT FirstName, Surname, D.DriverID,
CarRegNo, Make, Model
FROM Drivers D LEFT OUTER JOIN Automobiles A
ON D.DriverID = A.DriverID
```

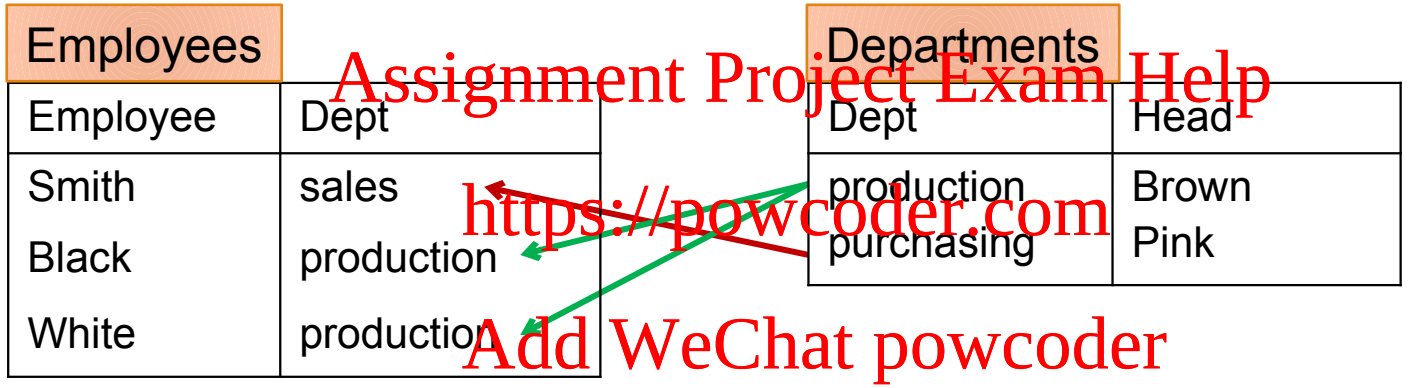

Results | Messages

	FirstName	Surname	DriverID	CarRegNo	Make	Model
1	Marco	Neri	AP4544442R	NULL	NULL	NULL
2	Charles	White	PZ10124436B	NULL	NULL	NULL
3	Mary	Brown	VR2030030Y	ABC123	BMW	323
4	Mary	Brown	VR2030030Y	DEF456	BMW	Z3

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

SQL JOINS – RIGHT OUTER JOINS

- RIGHT OUTER JOIN
  - Retains all dangling tuples from the table on the right of the join statement



Employees

Employee	Dept
Smith	sales
Black	production
White	production

Departments

Dept	Head
production	Brown
purchasing	Pink


```
SELECT * FROM Employees E
RIGHT OUTER JOIN Departments D
ON D.Dept = E.Dept
```

Results | Messages

	Employee	Dept	Dept	Head
1	Black	production	production	Brown
2	White	production	production	Brown
3	NULL	NULL	purchasing	Pink

Departments(DeptName, Head)
Employees(Employee, Dept)
FK(Dept) -> Departments(Dept)

SQL JOINS – RIGHT OUTER JOINS

- RIGHT OUTER JOIN
  - Find the details all cars and their drivers including cars without drivers:

```
SELECT FirstName, Surname, D.DriverID,
CarRegNo, Make, Model
FROM Drivers D RIGHT OUTER JOIN Automobiles A
ON D.DriverID = A.DriverID
```


Results | Messages

	FirstName	Surname	DriverID	CarRegNo	Make	Model
1	Mary	Brown	VR2030030Y	ABC123	BMW	323
2	NULL	NULL	NULL	BBB421	BMW	316
3	Mary	Brown	VR2030030Y	DEF456	BMW	Z3
4	NULL	NULL	NULL	GHI789	Lancia	Delta

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

SQL JOINS – FULL OUTER JOINS

- FULL OUTER JOIN
  - Retains all dangling tuples from both tables either side of the join statement

Employees

Employee	Dept
Smith	sales
Black	production
White	production

Departments

Dept	Head
production	Brown
purchasing	Pink


```
SELECT * FROM Employees E
FULL OUTER JOIN Departments D
ON D.Dept = E.Dept
```

Results | Messages

	Employee	Dept	Dept	Head
1	Black	production	production	Brown
2	Smith	sales	NULL	NULL
3	White	production	production	Brown
4	NULL	NULL	purchasing	Pink

Departments(DeptName, Head)
Employees(Employee, Dept)
FK(Dept) -> Departments(Dept)

SQL JOINS – FULL OUTER JOINS

- FULL OUTER JOIN
  - Find the drivers with their cars, including drivers without cars, and cars without drivers:

```
SELECT FirstName, Surname, D.DriverID, CarRegNo, Make, Model
FROM Drivers D FULL OUTER JOIN Automobiles A
ON D.DriverID = A.DriverID
```


Results | Messages

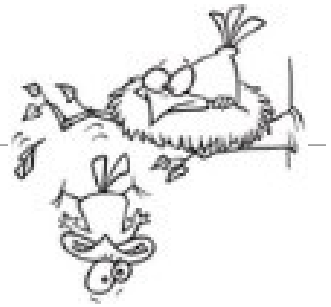
	FirstName	Surname	DriverID	CarRegNo	Make	Model
1	Marco	Neri	AP4544442R	NULL	NULL	NULL
2	Charles	White	PZ1012436B	NULL	NULL	NULL
3	Mary	Brown	VR2030030Y	ABC123	BMW	323
4	Mary	Brown	VR2030030Y	DEF456	BMW	Z3
5	NULL	NULL	NULL	BBB421	BMW	316
6	NULL	NULL	NULL	GHI789	Lancia	Delta

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

40 Previous Slides Can Be Summarized As

http://sqlwithmanoj.com/tag/cross-join-2/

Nested Queries



Nested Queries

- WHERE clause conditions can also
  - Compare an attribute expression with the result of another SQL query
    - **IN** - true if the attribute value exists in the results returned by the sub query
    - **NOT IN** - true if the attribute value being compared does NOT exist in the results returned by the sub query

    - ScalarValue Operator < ANY | ALL > SelectSQL
    - **ANY** - true if at least one row returned by SelectSQL satisfies the comparison. This is **default** if not specified
    - **ALL** - true if all rows returned by SelectSQL satisfy the comparison

  - Use the existential quantifier on an SQL query
    - **EXISTS** - true if sub query returns a result
    - **NOT EXISTS**

- The query appearing in the where clause is called a nested query

SQL – Sub Queries (IN)

- Sub Queries are nested SELECT statements within the condition of another query
- Sub Queries are used to provide a list of values that can be tested/used in a condition statement


```
SELECT * FROM Employees WHERE Dept IN (
SELECT DeptName FROM Departments
WHERE City <> 'London'
);
```


= ANY by default

- Equivalent to

```
SELECT E.* FROM Employees E JOIN Departments D
ON E.Dept = D.DeptName
WHERE D.City <> 'London'
```

Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Alice	Jackson	Production	75	46	Toulouse
2	Charles	White	Production	20	36	Toulouse
3	Jackson	Neri	Distribution	16	45	Dover

Simple nested query example 1

- Find the employees who work in departments in London

```
SELECT FirstName, Surname
FROM Employees
WHERE Dept IN (
            SELECT DeptName
   FROM Departments
   WHERE City = 'London'
)
```


Results | Messages

	FirstName	Surname
1	Charles	Brown
2	Gus	Green
3	Laurence	Chen
4	Mary	Brown
5	Pauline	Bradshaw

- Equivalent to

```
SELECT e.FirstName, e.Surname
FROM Employees e INNER JOIN Departments d
ON e.Dept = d.DeptName
WHERE d.City = 'London'
```

.

Simple nested query example 2

- Find the employees in Planning with the same first name as someone in Production
  - with a nested query

```
SELECT FirstName, Surname FROM Employees
WHERE Dept = 'Planning'
AND FirstName IN (
  SELECT FirstName
  FROM Employees
  WHERE Dept = 'Production'
)
```


  - without a nested query

```
SELECT e1.FirstName, e1.Surname
FROM Employees e1 INNER JOIN Employees e2
ON e1.FirstName = e2.FirstName
WHERE e1.Dept = 'Planning'
AND e2.Dept = 'Production'
```

Nested query with negation

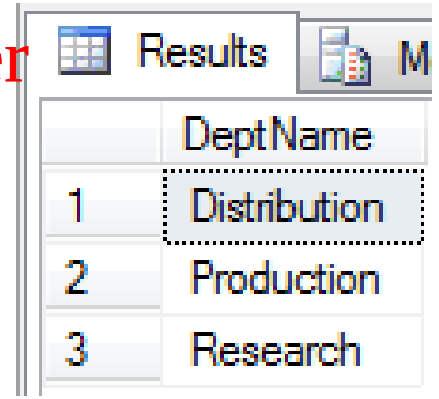
- Find the departments in which there is no-one named Brown

```
SELECT DeptName
FROM Departments
WHERE DeptName NOT IN (
        SELECT Dept
        FROM Employees
        WHERE Surname = 'Brown'
)
```

<> any

- Or using set difference

```
SELECT DeptName
FROM Departments
    EXCEPT
SELECT Dept AS DeptName
FROM Employees
WHERE Surname = 'Brown'
```



SQL – Sub Queries (NOT IN)

- Sub Queries are nested SELECT statements within the condition of another query
- Sub Queries are used to provide a list of values that can be tested/used in a condition statement


```
SELECT * FROM Employees WHERE Dept NOT IN (
SELECT DeptName FROM Departments
WHERE City <> 'London'
);
```


Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Charles	Brown	Planning	14	80	London
2	Gus	Green	Administration	20	40	Oxford
3	Laurence	Chen	Planning	7	73	Worthing
4	Mary	Brown	Administration	10	45	London
5	Pauline	Bradshaw	Administration	75	40	Brighton

```
SELECT E.* FROM Employees E JOIN Departments D
ON E.Dept = D.DeptName
WHERE D.City = 'London'
```

What does all this mean?

There are many equivalent ways to write the same query

Nested query using EXISTS

- A nested query can use variables of the external query ('transfer of bindings')
  - Semantics: the nested query is evaluated for each row of the external query

- Find students with the same name as another student


```
SELECT *
FROM Student s
WHERE EXISTS (
  SELECT *
  FROM Student s2
  WHERE s.Name = s2.Name
  AND s.StudentID <> s2.StudentID
)
```

SQL – Sub Queries (EXISTS)

- EXISTS
  - Tests if results are returned by a sub query rather than if a particular value exists in the sub query
    - EXISTS = sub query is not empty
    - NOT EXISTS = sub query is empty


```
SELECT * FROM Employees E
WHERE EXISTS (
SELECT * FROM Departments D
WHERE D.City = E.City
);
```

Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Alice	Jackson	Production	75	46	Toulouse
2	Charles	Brown	Planning	14	80	London
3	Charles	White	Production	20	36	Toulouse
4	Mary	Brown	Administration	10	45	London
5	Pauline	Bradshaw	Administration	75	40	Brighton

Departments

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

SQL – Sub Queries (NOT EXISTS)

- EXISTS
  - Tests if results are returned by a sub query rather than if a particular value exists in the sub query
    - EXISTS = sub query is not empty
    - NOT EXISTS = sub query is empty


```
SELECT * FROM Employees E
WHERE NOT EXISTS (
SELECT * FROM Departments D
WHERE D.City = E.City
);
```

Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Gus	Green	Administration	20	40	Oxford
2	Jackson	Neri	Distribution	16	45	Dover
3	Laurence	Chen	Planning	7	73	Worthing

Departments

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

SQL – Sub Queries (NOT EXISTS)

- EXISTS
  - Tests if results are returned by a sub query rather than if a particular value exists in the sub query
    - EXISTS = sub query is not empty
    - NOT EXISTS = sub query is empty


```
SELECT * FROM Employees E1
WHERE NOT EXISTS (
SELECT * FROM Employees E2
WHERE E2.Salary > E1.Salary
)
```

Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Charles	Brown	Planning	14	80	London

Nested queries using NOT EXISTS

- Find the names of all courses with no students enrolled

```
SELECT CourseName
FROM Course c
WHERE NOT EXISTS (
   SELECT *
   FROM Enrolment e
   WHERE c.CourseID = e.CourseID
  )
```

SQL – Sub Queries (ALL)

- Example: Find the employee with the highest salary

```
SELECT * FROM Employees E1
WHERE E1.Salary >= ALL (
SELECT E2.Salary FROM Employees E2
)
```


Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Charles	Brown	Planning	14	80	London

~~SELECT E1.* FROM Employees E1, Employees E2~~
~~WHERE E1.Salary >= E2.Salary;~~

	FirstName	Surname	Dept	Office	Dept	E1Salary		E2Salary
1	Charles	Brown	Planning	14	Planning	80	>	46
2	Laurence	Chen	Planning	7	Planning	73	>	46
3	Alice	Jackson	Production	75	Production	46	>	36
4	Charles	Brown	Planning	14	Planning	80	...	36
5	Gus	Green	Administration	20	Administration	40		36
6	Jackson	Neri	Distribution	16	Distribution	45		36
7	Laurence	Chen	Planning	7	Planning	73		36

SQL – Sub Queries in SELECT

- A Sub-SELECT query returns a value to be used as a result in the main query
  - This is different from the previous examples where the nested query is used to help generate/condition the results of the outer query
  - EXAMPLE: Find each employees salary as a % of the max salary for their given department


```
SELECT E.*, E.Salary / (
SELECT MAX(salary) FROM Employees E2
WHERE E2.Dept = E.Dept)* 100) AS PercentMax
FROM Employees E;
```


	FirstName	Surname	Dept	Office	Salary	City	PercentMax
1	Alice	Jackson	Production	75	46.00	Toulouse	100.00
2	Charles	Brown	Planning	14	80.00	London	100.00
3	Charles	White	Production	20	36.00	Toulouse	78.26
4	Gus	Green	Administration	20	40.00	Oxford	88.89
5	Jackson	Neri	Distribution	16	45.00	Dover	100.00
6	Laurence	Chen	Planning	7	73.00	Worthing	91.25
7	Mary	Brown	Administration	10	45.00	London	100.00
8	Pauline	Bradshaw	Administration	75	40.00	Brighton	88.89

```
SELECT E.*, CONVERT(Decimal(6,2),
E.Salary / (
SELECT MAX(salary) FROM Employees E2
WHERE E2.Dept = E.Dept)* 100) AS PercentMax
FROM Employees E;
```

SQL – Sub Queries in SELECT

- A Sub-SELECT query returns a value to be used as a result in the outer query
  - This is different from the previous examples where the nested query is used to help generate/condition the results of the outer query
- Note:
  - Because the SELECT query is part of another SELECT Clause it must only return ONE result – because it is being used to fill in a single result


```
SELECT E.*, CONVERT(Decimal(6,2),
E.Salary / (
SELECT (salary) FROM Employees E2
WHERE E2.Dept = E.Dept)* 100) AS PercentMax
FROM Employees E;
```

Results | Messages

```
Msg 512, Level 16, State 1, Line 1
Subquery returned more than 1 value.
This is not permitted when the subquery
follows =, !=, <, <= , >, >= or when the subquery
is used as an expression.
```

SQL – Sub Queries in FROM

- A Sub-FROM query returns a relation that can be used to help generate the results in the outer query
  - Example: Find all Employees that live in the same capital city:

```
SELECT E.* FROM Employees E
JOIN ( SELECT E2.* FROM Employees E2 ) AS MySubQuery
ON E.Dept = MySubQuery.Dept
WHERE E.FirstName <> MySubQuery.FirstName
AND E.Surname <> MySubQuery.Surname
AND E.City = MySubQuery.City;
```



```
SELECT * FROM (
  SELECT E1.* FROM Employees E1 JOIN Employees E2
  ON E1.Dept = E2.Dept
  WHERE E1.FirstName <> E2.FirstName
  AND E1.Surname <> E2.Surname
  AND E1.City = E2.City
) AS MySubQuery
```

Results | Messages

	FirstName	Surname	Dept	Office	Salary	City
1	Alice	Jackson	Production	75	46	Toulouse
2	Charles	White	Production	20	36	Toulouse

Comments on nested queries

- The use of nested queries may produce ‘less declarative’ queries, but they can be more readable
- Complex queries can become very difficult to understand
- The use of variables must respect visibility rules
  - a variable can only be used within the query where it is defined, or within a query that is nested in the query where it is defined