MORE SQL OPERATORS, WORKING WITH STRINGS AND SUB-QUERIES

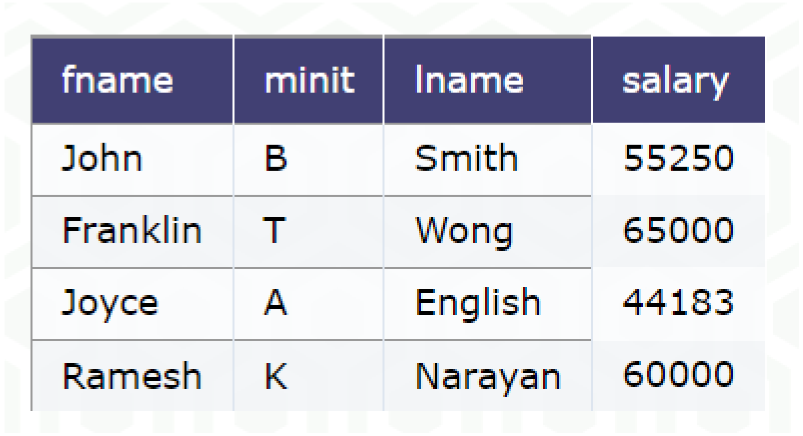
CT230

Database Systems 1

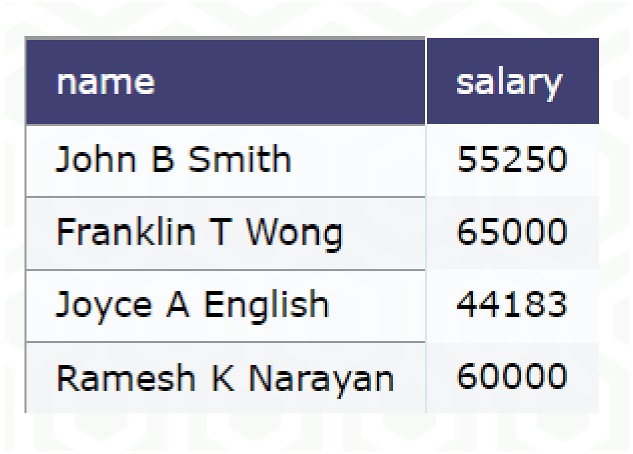
CONCATENATING STRINGS AND ORDERING OUTPUT

Although we want to store atomic attributes as much as possible we may not want to display string values in a way different to how they are stored

For example, for query #10. Retrieve the salary and name of all employees working in department 5, compare the outputs:



fname	minit	lname	salary
John	B	Smith	55250
Franklin	T	Wong	65000
Joyce	A	English	44183
Ramesh	K	Narayan	60000



name	salary
John B Smith	55250
Franklin T Wong	65000
Joyce A English	44183
Ramesh K Narayan	60000

KEYWORDS TO MODIFY OUTPUT …

AS

… Used to rename any output in SELECT

… can also be used to re-name (alias) tables in FROM

CONCAT

… concatenate strings

… similar usage to other programming languages

CAST

… CAST(*expression* AS *datatype*(*length*))

ORDER BY

… last clause in SQL to order output results

ORDERING THE OUTPUT WITH ORDER BY

Syntax:

```
ORDER BY <attribute list>
```

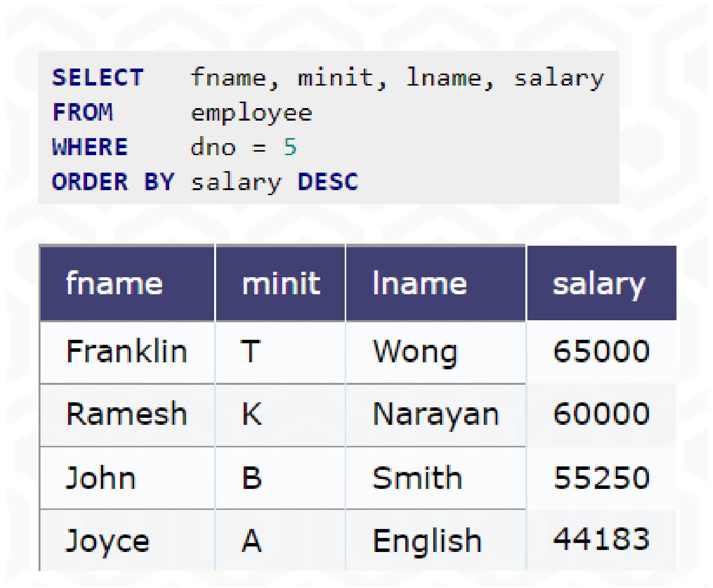
Allows the results of a query to be ordered by values of one or more attributes

Either ascending (ASC) or descending (DESC).

The default order is ascending.

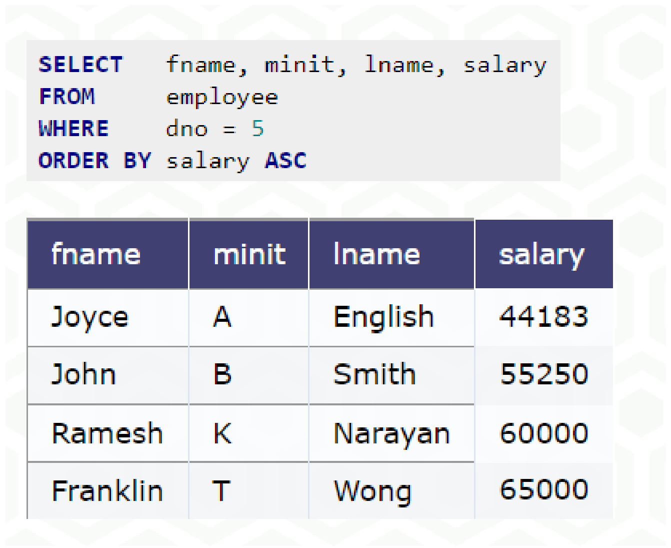
** Must be last clause of the SELECT statement.

Note: ORDER is a reserved keyword!



```
SELECT   fname, minit, lname, salary
FROM     employee
WHERE    dno = 5
ORDER BY salary DESC
```

fname	minit	lname	salary
Franklin	T	Wong	65000
Ramesh	K	Narayan	60000
John	B	Smith	55250
Joyce	A	English	44183



```
SELECT   fname, minit, lname, salary
FROM     employee
WHERE    dno = 5
ORDER BY salary ASC
```

fname	minit	lname	salary
Joyce	A	English	44183
John	B	Smith	55250
Ramesh	K	Narayan	60000
Franklin	T	Wong	65000

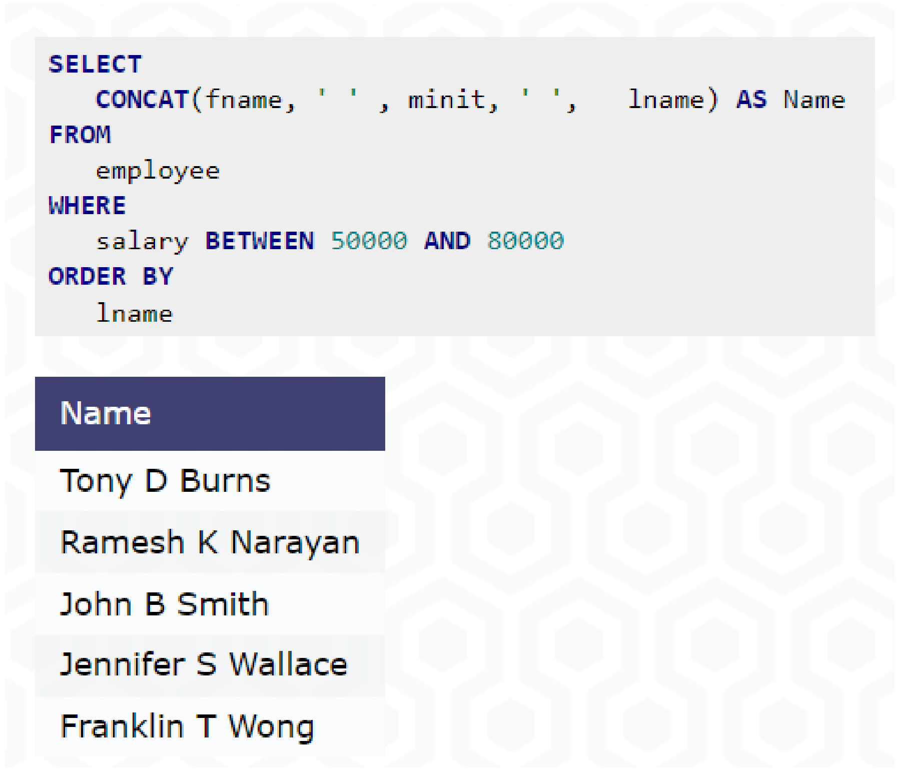
TIDYING UP SQL CODE … Example 11 again

EXAMPLE 11: Retrieve names of all employees whose salary is between 50000 and 80000

```
SELECT
    fname,
    minit,
    lname
FROM
    employee
WHERE
    salary BETWEEN 50000 AND 80000;
```

TIDYING UP OUTPUT… #11 again

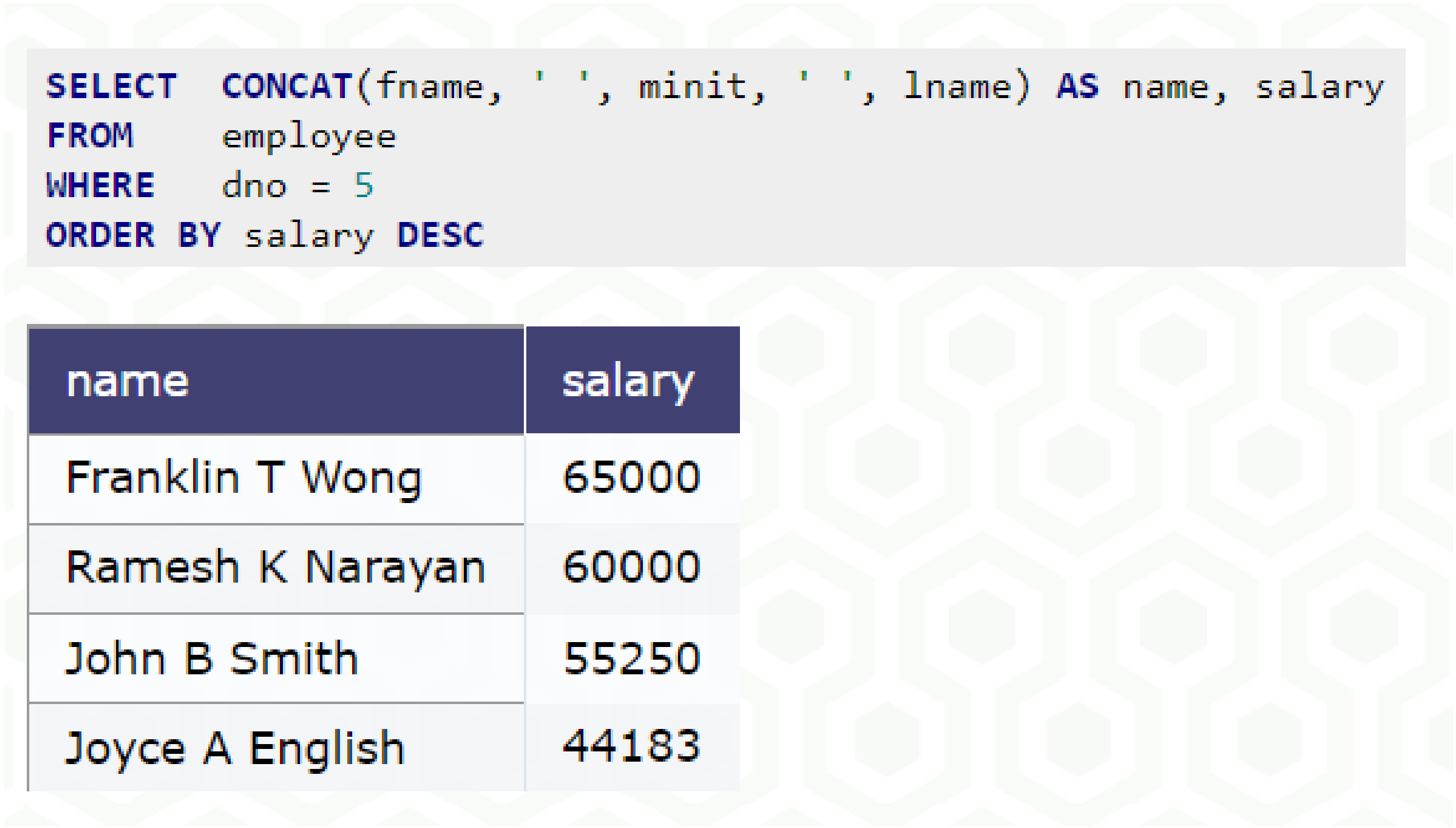
```
SELECT
   CONCAT(fname, ‘ ’, minit, ‘ ’,   lname) AS Name
FROM
   employee
WHERE
   salary BETWEEN 50000 AND 80000
ORDER BY
   lname;
```



```
SELECT
   CONCAT(fname, ' ' , minit, ' ',   lname) AS Name
FROM
   employee
WHERE
   salary BETWEEN 50000 AND 80000
ORDER BY
   lname
```

Name
Tony D Burns
Ramesh K Narayan
John B Smith
Jennifer S Wallace
Franklin T Wong

EXAMPLE 12: Produce a list of salaries for all staff, produced in descending order of salary (outputting ssn, names and salary)



```
SELECT   CONCAT(fname, ' ', minit, ' ', lname) AS name, salary
FROM     employee
WHERE    dno = 5
ORDER BY salary DESC
```

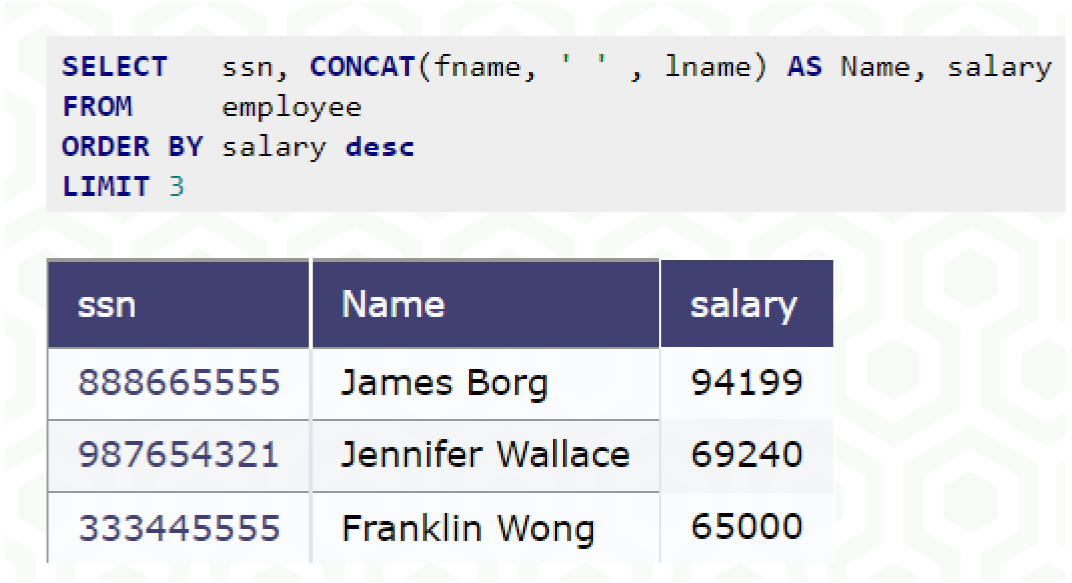
name	salary
Franklin T Wong	65000
Ramesh K Narayan	60000
John B Smith	55250
Joyce A English	44183

TOP AND LIMIT (EXAMPLE 13)

SELECT TOP N clause is used to specify the number of tuples/rows (N) to return but it is not supported by mySQL. Instead mySQL supports a LIMIT N clause which has the same functionality. The LIMIT clause is listed at the end of the query.

Example 13: List the employees with the top 3 salaries

```
SELECT
    ssn, CONCAT(fname, ‘ ‘ , lname) AS Name , salary
FROM
    employee
ORDER BY
    salary DESC
LIMIT 3;
```



```
SELECT   ssn, CONCAT(fname, ' ' , lname) AS Name, salary
FROM     employee
ORDER BY salary desc
LIMIT 3
```

ssn	Name	salary
888665555	James Borg	94199
987654321	Jennifer Wallace	69240
333445555	Franklin Wong	65000

NOTE: SINGLE AND DOUBLE QUOTES

MySQL usually allows single and double quotes to be used interchangeably.

Generally, single quotes should be used for strings (varchar(), text, etc.)

HOW TO DEAL WITH APOSTROPHES IN STRINGS ….

We must be careful because an opening quote could be accidently closed by an apostrophe.

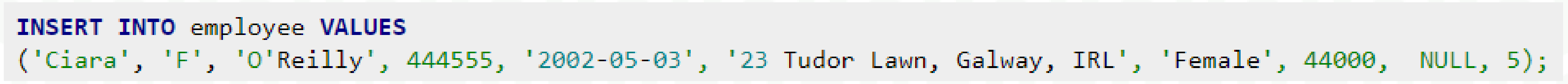
To overcome this, if there is an apostrophe in a string it should be replaced by two apostrophes side-by-side (general rule for all special characters – have two of the character) or \

e.g., Find the salary for the employee with surname O'Grady

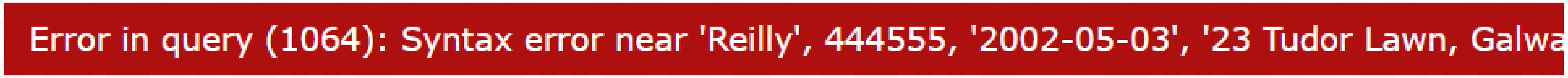
```
SELECT salary
FROM employee
WHERE lname = 'O''Grady';
```

N.B. Must also take care of this when inserting string data using INSERT INTO

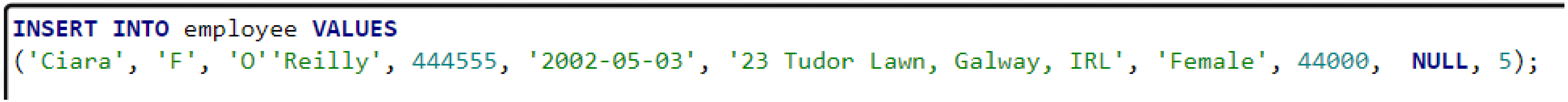
Example from company database:



```
INSERT INTO employee VALUES
('Ciara', 'F', 'O'Reilly', 444555, '2002-05-03', '23 Tudor Lawn, Galway, IRL', 'Female', 44000,  NULL, 5);
```



```
Error in query (1064): Syntax error near 'Reilly', 444555, '2002-05-03', '23 Tudor Lawn, Galwa
```



```
INSERT INTO employee VALUES
('Ciara', 'F', 'O''Reilly', 444555, '2002-05-03', '23 Tudor Lawn, Galway, IRL', 'Female', 44000,  NULL, 5);
```

EXAMPLE 14: Using the operator Is Null retrieve names of all employees who **Do Not** have a supervisor (superssn IS NULL)

IS NULL : allow an explicit search for NULL

SELECT

FROM

WHERE

WORKING WITH STRINGS AND PATTERN MATCHING

SQL is case insensitive (apart from table names as mentioned if on linux server)

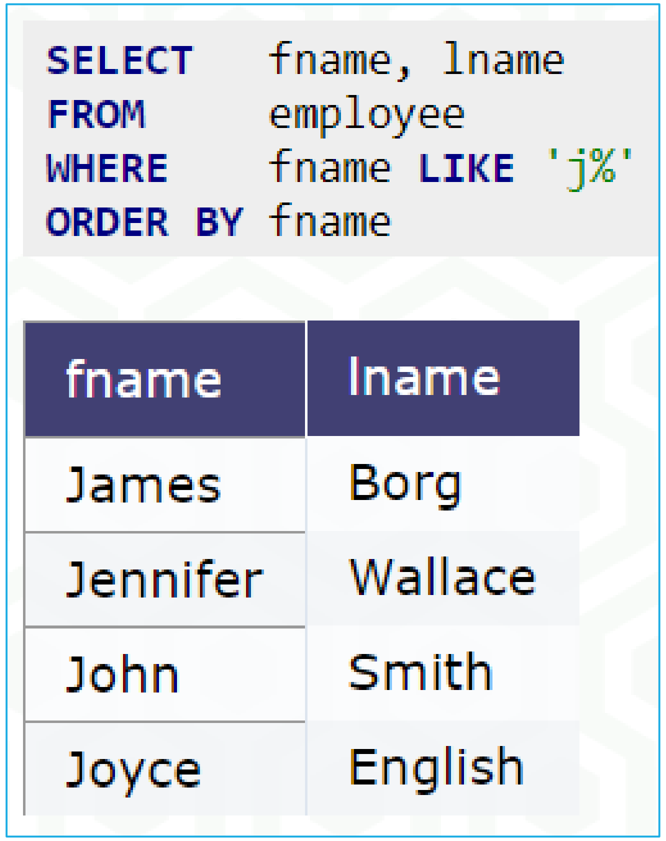
Case insensitivity also applies to string searching

However, *often* when working with strings we do not look for an exact match (i.e. an exact match using “=“)

To support partial matching often use pattern matching characters and `LIKE` with wildcard characters % and _

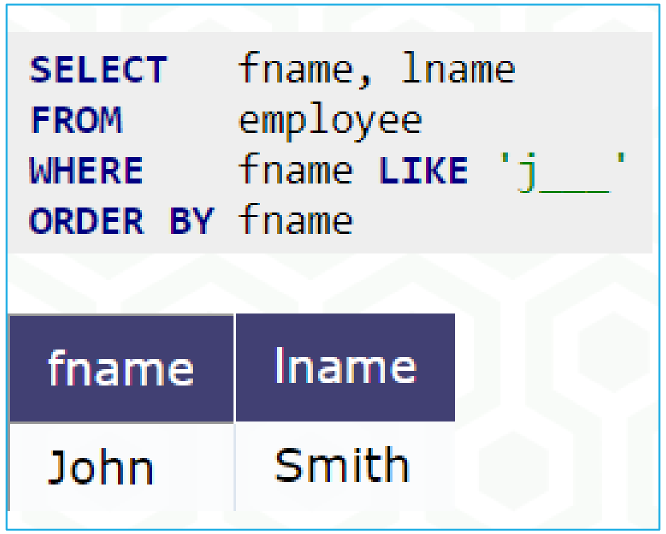
Symbol	Description	Example (fname)
%	Represents 0 or more characters	j% finds John, Joyce, James, Jennifer
_	Represents a single character	j___ finds John only

EXAMPLES (#15) … what is the difference?



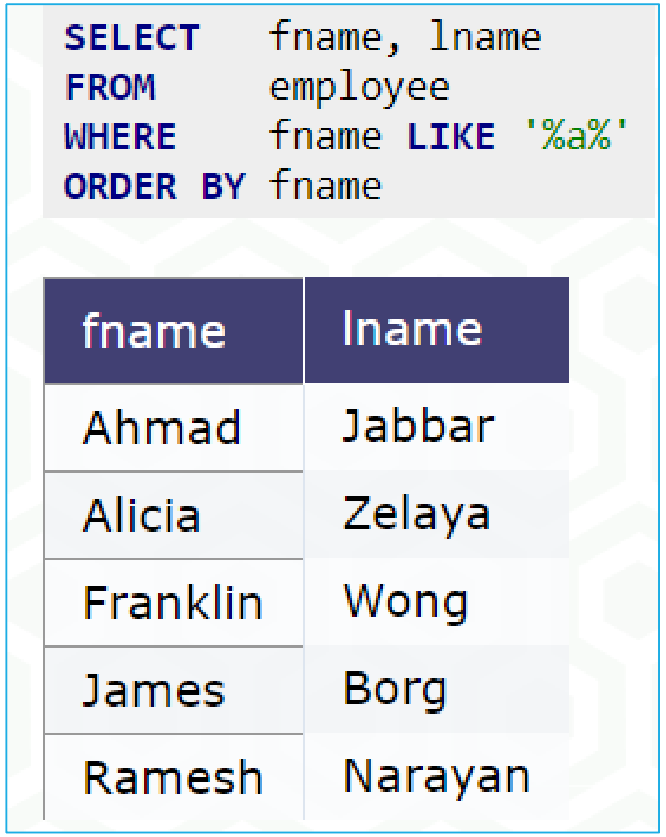
```
SELECT    fname, lname
FROM      employee
WHERE     fname LIKE 'j%'
ORDER BY  fname
```

fname	lname
James	Borg
Jennifer	Wallace
John	Smith
Joyce	English



```
SELECT    fname, lname
FROM      employee
WHERE     fname LIKE 'j___'
ORDER BY  fname
```

fname	lname
John	Smith



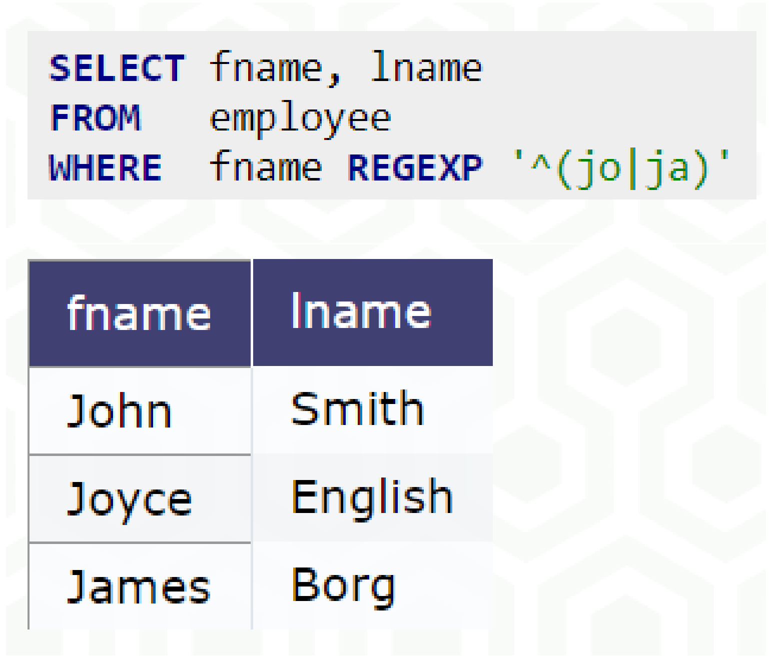
```
SELECT    fname, lname
FROM      employee
WHERE     fname LIKE '%a%'
ORDER BY  fname
```

fname	lname
Ahmad	Jabbar
Alicia	Zelaya
Franklin	Wong
James	Borg
Ramesh	Narayan

CAN USE REGEXP FOR MORE COMPLICATED STRING MATCHING

Symbol	Description
^	Matches position at the **beginning** of the searched string
$	Matches position at the **end** of the searched string
[ ]	Matches any character inside the square brackets
[^]	Matches any character **not** inside the square brackets
*	Matches preceding character 0 or more times
+	Matches preceding character 1 or more times
|	Or
{n}	Matches preceding character n number of times

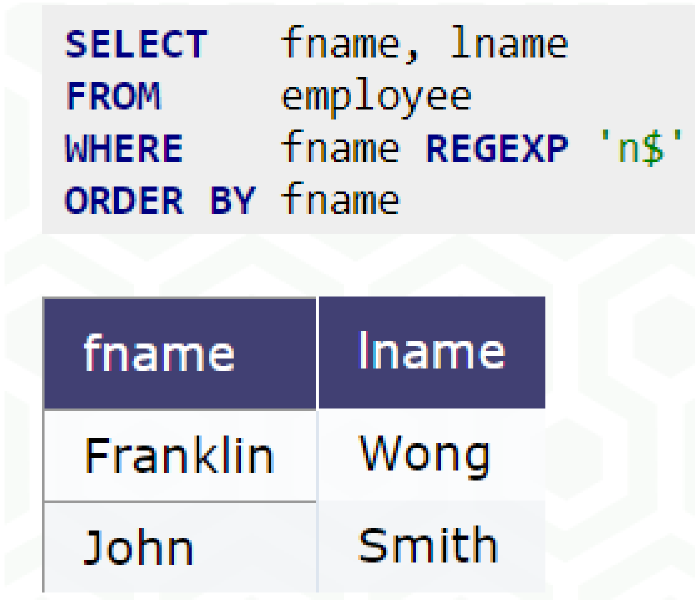
EXAMPLE 16a: Find the names of employees whose first names begin with *jo* or *ja*



```
SELECT fname, lname
FROM   employee
WHERE  fname REGEXP '^(jo|ja)'
```

fname	lname
John	Smith
Joyce	English
James	Borg

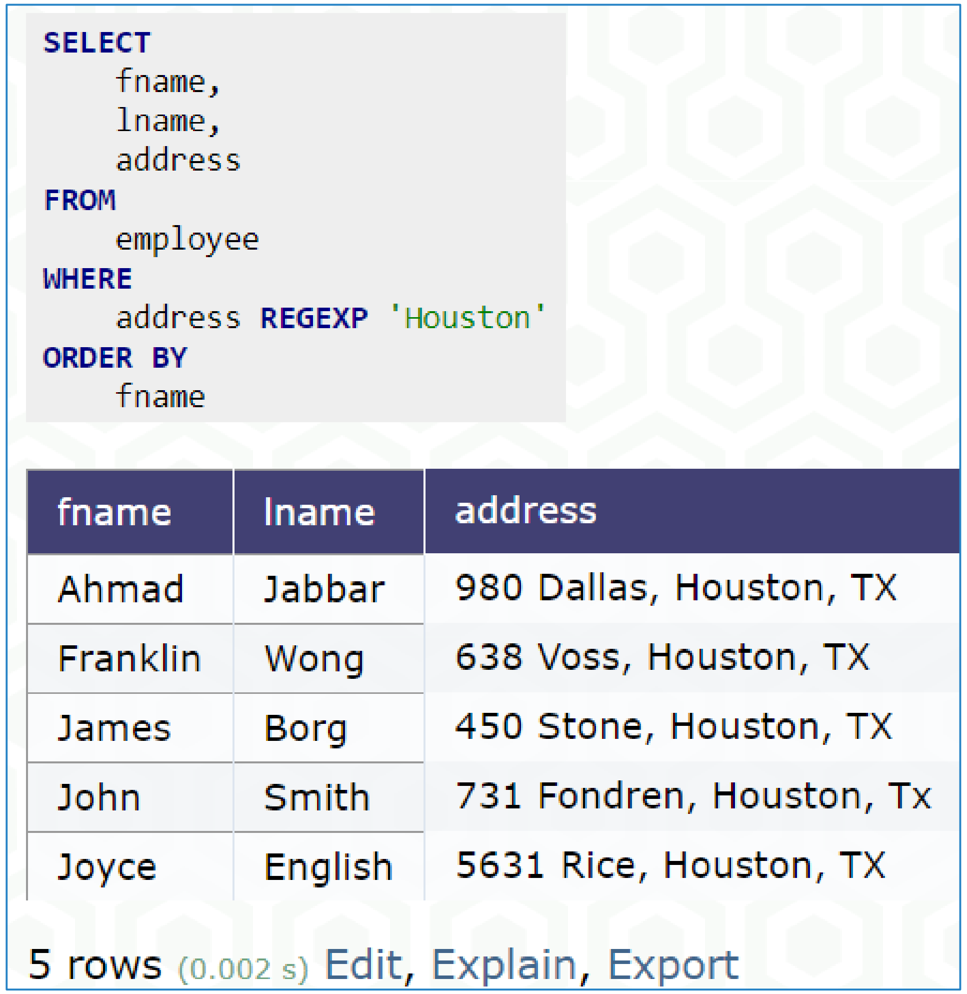
EXAMPLE 16b: Find the names of employees whose first names end with *n*



```
SELECT   fname, lname
FROM     employee
WHERE    fname REGEXP 'n$'
ORDER BY fname
```

fname	lname
Franklin	Wong
John	Smith

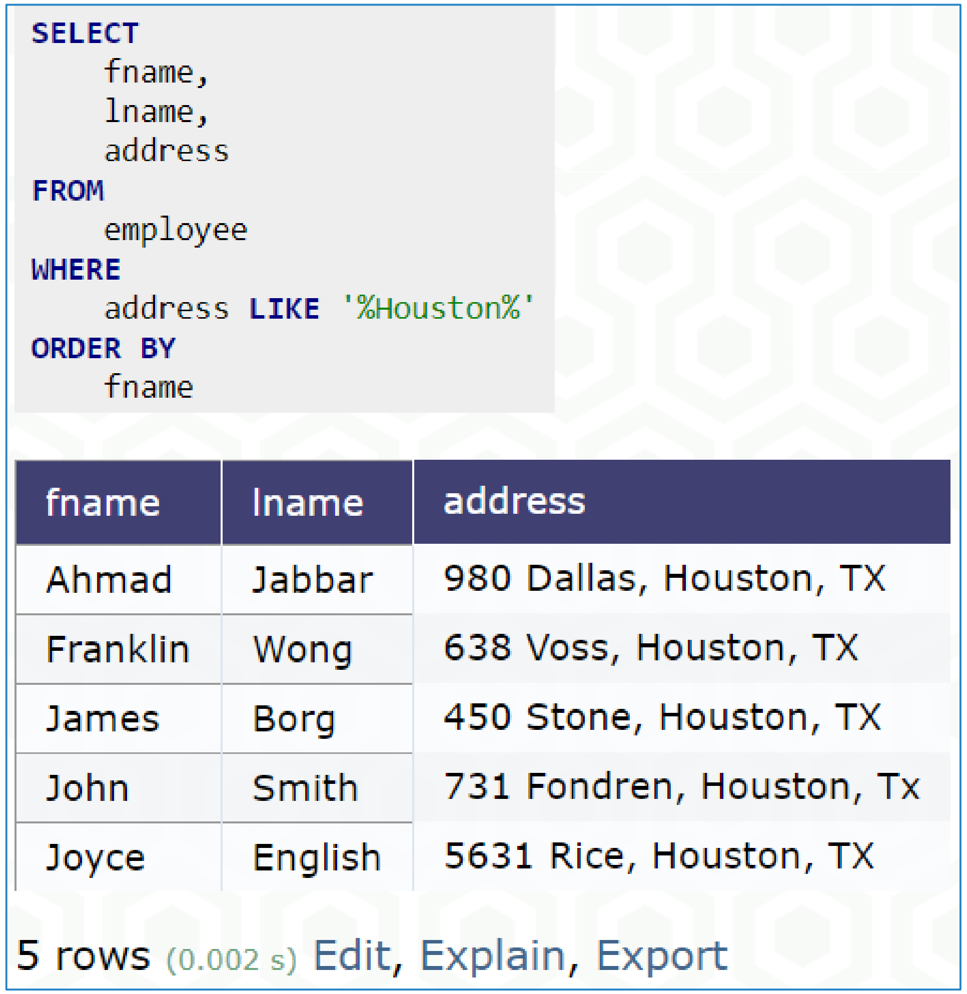
EXAMPLE 17: Find employees (name and address) who live in Houston



```
SELECT
    fname,
    lname,
    address
FROM
    employee
WHERE
    address REGEXP 'Houston'
ORDER BY
    fname
```

fname	lname	address
Ahmad	Jabbar	980 Dallas, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
James	Borg	450 Stone, Houston, TX
John	Smith	731 Fondren, Houston, Tx
Joyce	English	5631 Rice, Houston, TX

5 rows (0.002 s) Edit, Explain, Export



```
SELECT
    fname,
    lname,
    address
FROM
    employee
WHERE
    address LIKE '%Houston%'
ORDER BY
    fname
```

fname	lname	address
Ahmad	Jabbar	980 Dallas, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
James	Borg	450 Stone, Houston, TX
John	Smith	731 Fondren, Houston, Tx
Joyce	English	5631 Rice, Houston, TX

5 rows (0.002 s) Edit, Explain, Export

EXAMPLE 18:

Version 1: List the details (name and birth date) of the children of the employee with SSN 333445555

Version 2: List the details (name and birth date) of the children of Franklin T Wong

What is the difference?

For version 2, we need two tables and we need to explicitly link the two tables as part of the query (that is the `employee` and `dependent` tables) in order to meet this request or to use a sub-query

HOW TO ACCESS DATA ACROSS MULTIPLE TABLES?

3 potential approaches*:

- Joins
- Subqueries
- Union queries

* not all suitable for all problems

SUBQUERIES

- A subquery is a query within another query
  - Also called a ***nested*** query
- The subquery *usually* returns data that will be used in the main query
- Data returned from the subquery may be a set of values or a single value
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements

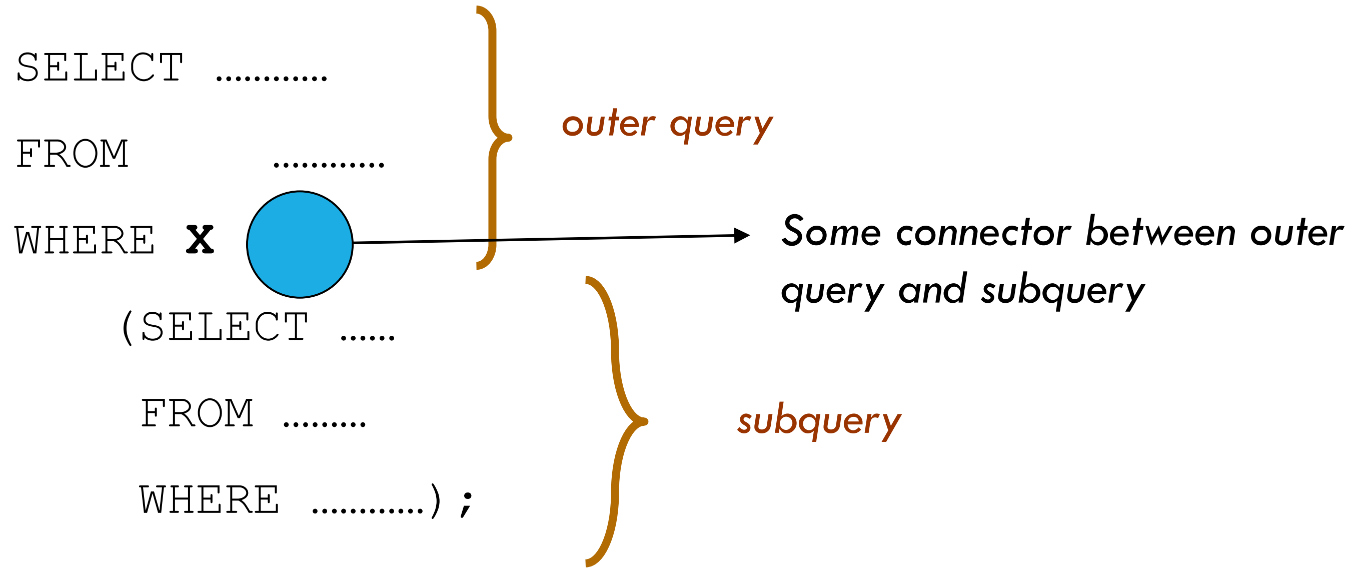
When to use a sub-query?

- Needed when an existing value from the database needs to be retrieved and used as part of the query solution.
- Needed when an aggregate function needs to be performed and used as part of a query solution.
- Can (sometimes) replace a join of tables (where appropriate).

Subqueries in SELECT

Subqueries can be used as part of the **WHERE** and **HAVING** clauses of an outer SELECT

SUBQUERY SAMPLE FORMAT:



```
SELECT ............
FROM         ............
WHERE X
    (SELECT ......
     FROM .........
     WHERE ............);
```

Nested SELECT statement is called a *subquery*
SELECT statement which contains subquery is called an *outer query*

CONNECTING OUTER AND INNER QUERIES (1 OF 2)

If subquery returns only **one value** then can use operators such as:

=, !=, >, >=, <, <=

If subquery *could* return **more than one value (i.e., a list of values)** then need connectors such as:

IN, ANY, ALL to check through the values from the subquery.

CONNECTING OUTER AND INNER QUERIES (2 OF 2)

The keyword NOT can also be used where appropriate (often with `IN`, e.g., `NOT IN`)

In addition can have a more general condition using:

Exists: True if there exists at least one value in the result from a subquery

Not Exists: True if there is nothing in the result form a subquery (i.e. it is empty).

CONNECTORS: ANY, ALL

Used with basic mathematical operators: =, != , >, <, >=, <=

For example,

```
=ALL

>ANY
```

- ALL: the condition is true if the comparison is true for every (ALL) values returned by the subquery.
- ANY: the condition is true if the comparison is true for at least one (ANY) value returned by the subquery.

CONNECTOR: IN

Checks for equality.

Can be used for a list of values or a single value.

Does not require any additional mathematical operator.

The IN condition is true if the comparison is true for at least one value returned by the subquery, i.e. "a value is IN the subquery".

Returning to EXAMPLE 18:

Version 2: List the details (name and birth date) of the children of Franklin T Wong?

Using a sub-query:

- The sub-query should query the employee table to find the ssn of the employee Franklin T Wong.
- The outer query can then use the ssn returned by the subquery to check if the ssn exists (as an essn) in the dependent table. If/when a match is found return the name and birth date of the children.

EXAMPLE 18 *ctd.*

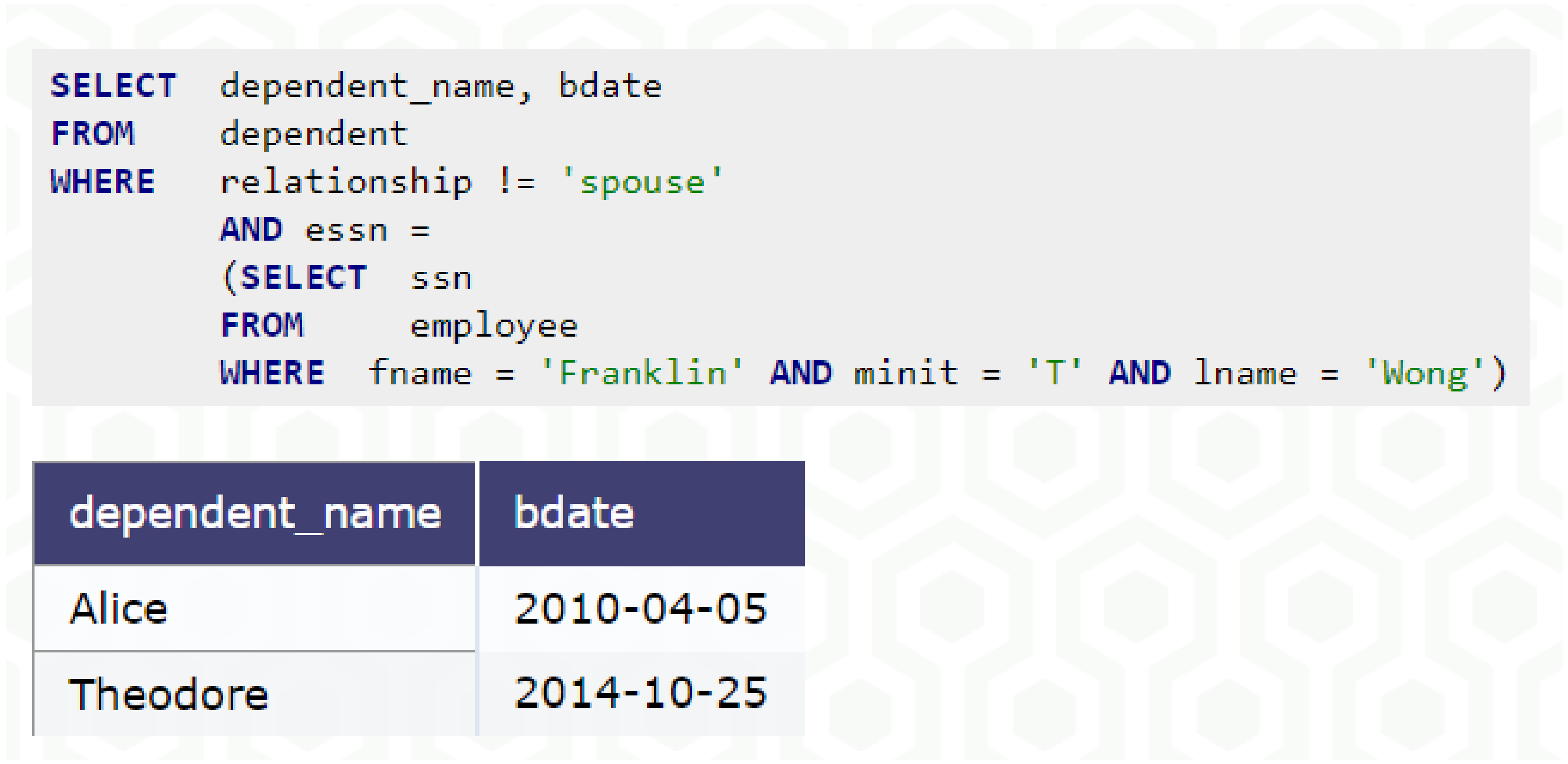
- "The sub-query should query the employee table to find the ssn of the employee Franklin T Wong"

```
SELECT  ssn
FROM     employee
WHERE  fname = 'Franklin' AND minit = 'T' AND lname = 'Wong';
```

- The outer query can then use the ssn returned by the subquery to check if the ssn exists (as an essn) in the dependent table. If/when a match is found return the name and birth date of the children (not spouse).

```
SELECT  dependent_name, bdate
FROM     dependent
WHERE   relationship != ‘spouse’ AND essn =
```

PUTTING THIS TOGETHER ….



```
SELECT  dependent_name, bdate
FROM    dependent
WHERE   relationship != 'spouse'
        AND essn =
        (SELECT  ssn
        FROM     employee
        WHERE  fname = 'Franklin' AND minit = 'T' AND lname = 'Wong')
```

dependent_name	bdate
Alice	2010-04-05
Theodore	2014-10-25

TRY …. EXAMPLE 19: Using a subquery method, list the staff (names) who work in department named ‘headquarters’

EXAMPLE 20: Using subqueries, list the names of all employees who are in the same department as employee John B Smith

Steps:

1. Use a subquery to get John B Smith’s department (a single number)

2. Use outer query to find who else is in that department number

* Be careful not to return “John B Smith” in the answer – i.e. he is in his own department!

You try ….

#21 Retrieve the name and salary of all employees who work on a project for greater than 20 hours.

#22 Retrieve the names of employees who have no dependents (Hint: using NOT IN to connect the queries).

SUMMARY

- Working with strings is an important part of SQL coding.
- Writing code that is easy to read – and that produces easy-to-read output is also very important.
- We can nest queries so that we can access data across multiple tables (Sub-queries). It is very important to use the correct connector between outer and inner queries (often there is more than one suitable option).