Relational databases

The Topic: What?

- An introduction to relational databases
  - Interacting with a database
  - Creating tables
  - Using Python to Insert, Update, and Delete
  - Using Python to read from a database
  - Joins
  - Aggregates
  - Exception Handling

Applications: Why?

- To be able to break down a complex problem into more manageable sub-tasks.
- To create code that can be re-used, that is simple, and easily maintained.

Expectations: Who?

- Learners are expected to have covered tuples, lists, dictionaries, and sets in Python previously.

Persistent data

The Art of saving data to non-volatile storage

Data

Files

Database

Pickle

Shelve

Interacting with a database

Trainer demonstration

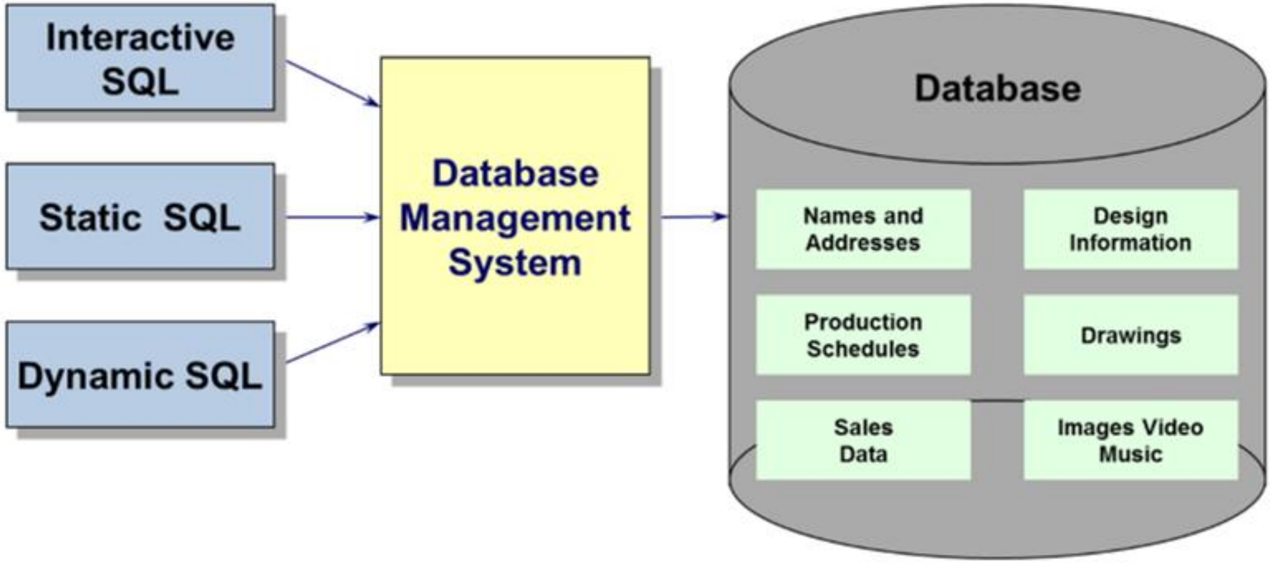
demo_sqlite.py

SQL

What is SQL?

Structured Query Language (SQL) is:

- Specifically designed to communicate with databases
- Industry-wide standard used by most database systems
- Various ways SQL can be used to access the data

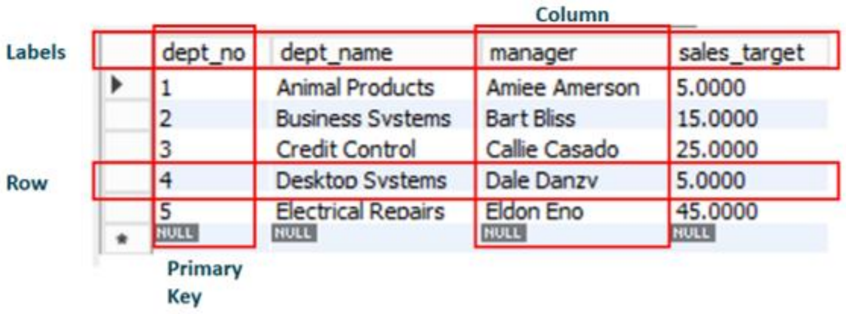


Database tables

Database Table

A database is simply a means to store data in a structured manner

- Composed of one more tables
- Data organised into rows and columns
  - Labels identify the data in each column
  - Each row of data also known as a record
  - Data not stored in any specific order – Sort as needed when
  - Primary Key used to uniquely identify each record
- Easy to reference and maintain



dept_no	dept_name	manager	sales_target
1	Animal Products	Amiee Amerson	5.0000
2	Business Systems	Bart Bliss	15.0000
3	Credit Control	Callie Casado	25.0000
4	Desktop Systems	Dale Danzy	5.0000
5	Electrical Repairs	Eldon Eno	45.0000
NULL	NULL	NULL	NULL

SQL: insert, update, delete

The commands within SQL which are involved in creating databases and tables is known as Data Definition Language (DDL). While the commands that are used to maintain the data within it are part of Data Manipulation Language (DML).

DDL: CREATE / ALTER / DROP

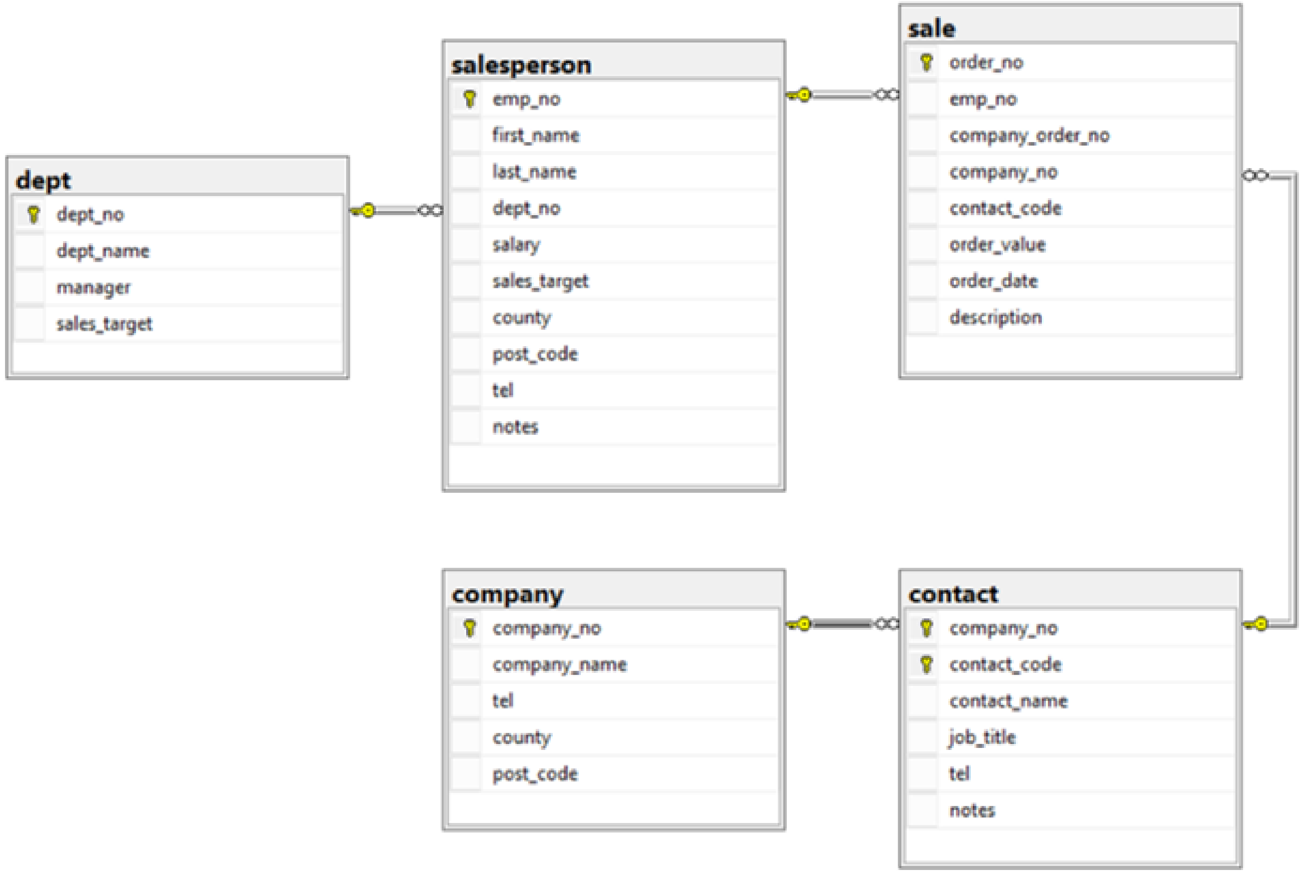
- **TABLE**
- **VIEW**
- **INDEX**

DML:

- **INSERT**
- **UPDATE**
- **DELETE**

Using DDL and dml

Trainer demonstration



dept
- dept_no
- dept_name
- manager
- sales_target

salesperson
- emp_no
- first_name
- last_name
- dept_no
- salary
- sales_target
- county
- post_code
- tel
- notes

sale
- order_no
- emp_no
- company_order_no
- company_no
- contact_code
- order_value
- order_date
- description

company
- company_no
- company_name
- tel
- county
- post_code

contact
- company_no
- contact_code
- contact_name
- job_title
- tel
- notes

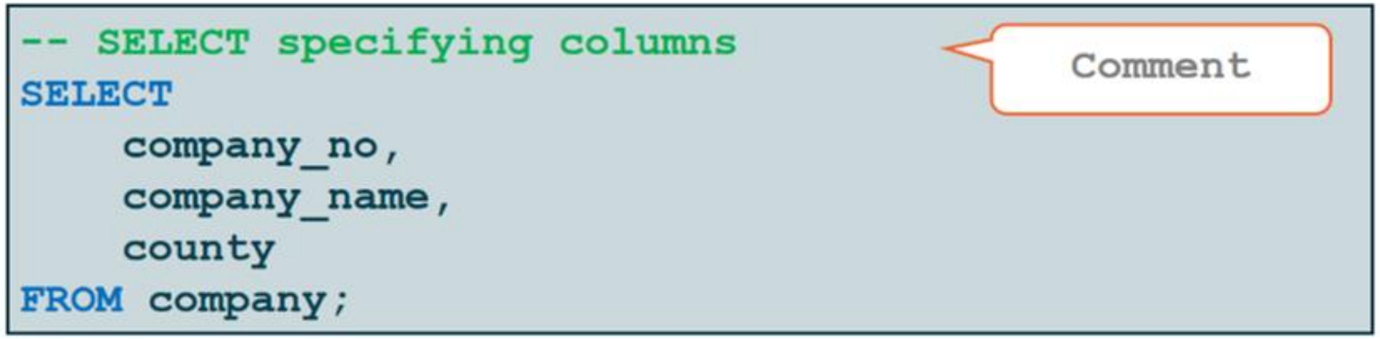
- qastore

Note: Section 1.2 shows the checks that can be done to make sure the database has been set up as expected, using SELECT (this is not part of DDL/DML).

SQL: Select

The commands within SQL which are involved in querying and displaying the data are known as Data Query Language (DQL)

SELECT – Specifying Columns

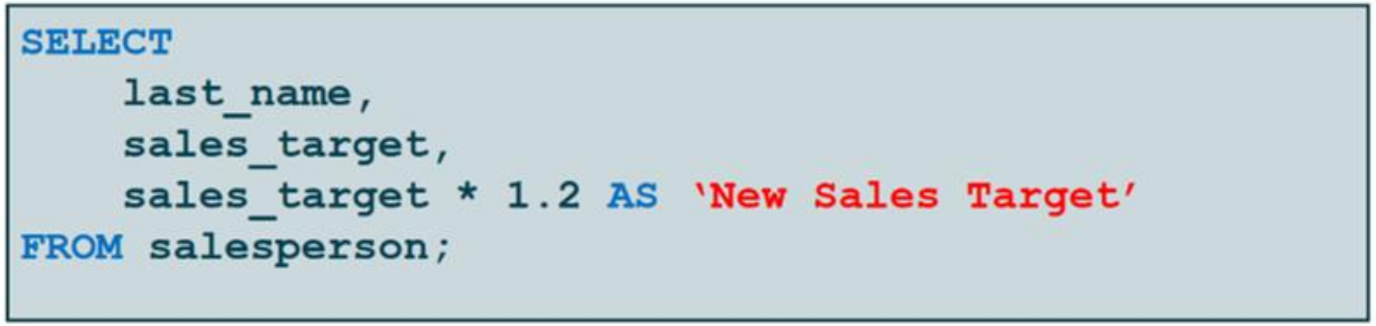


```
-- SELECT specifying columns
SELECT
    company_no,
    company_name,
    county
FROM company;
```

- What happens if you change the column order?
- What happens if you repeat a column?
- What happens if you misspell a column?
- What happens if you use a column from another table?

SQL: Select

SELECT – Calculated / Virtual Column and Alias

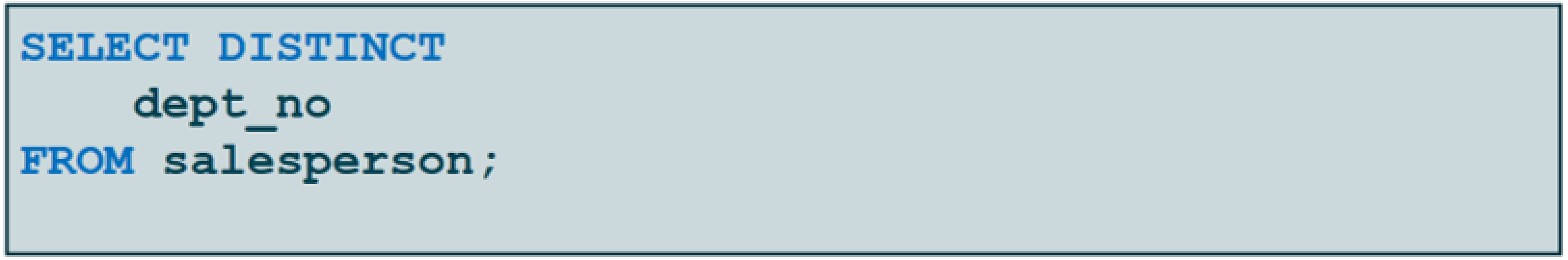


```
SELECT
    last_name,
    sales_target,
    sales_target * 1.2 AS 'New Sales Target'
FROM salesperson;
```

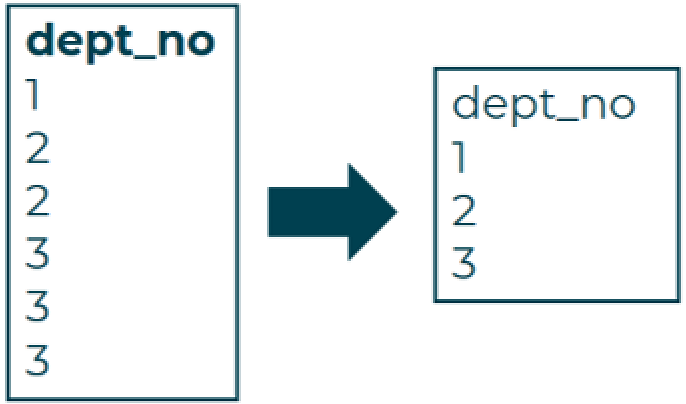
- What happens if you miss out the ‘AS’?
- What happens if you miss out the quotes?
- What if the alias is just a single word?
- Has sales_target been changed on the table?

SQL: Select

SELECT - Distinct



```
SELECT DISTINCT
       dept_no
FROM salesperson;
```

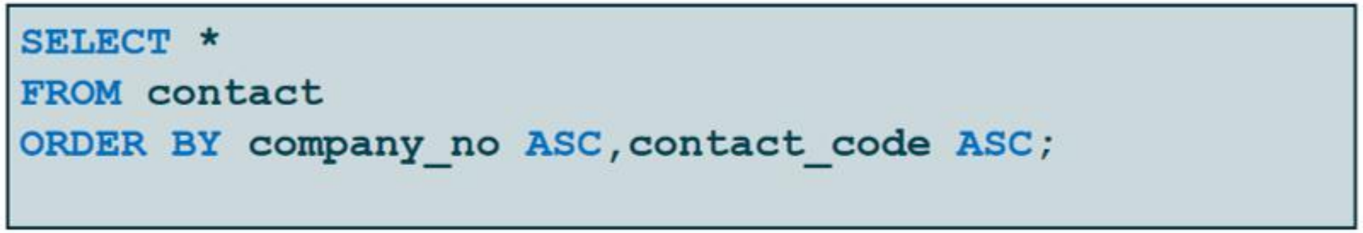


dept_no
1
2
2
3
3
3

dept_no
1
2
3

SQL: Sorting

SELECT with ORDER BY Ascending

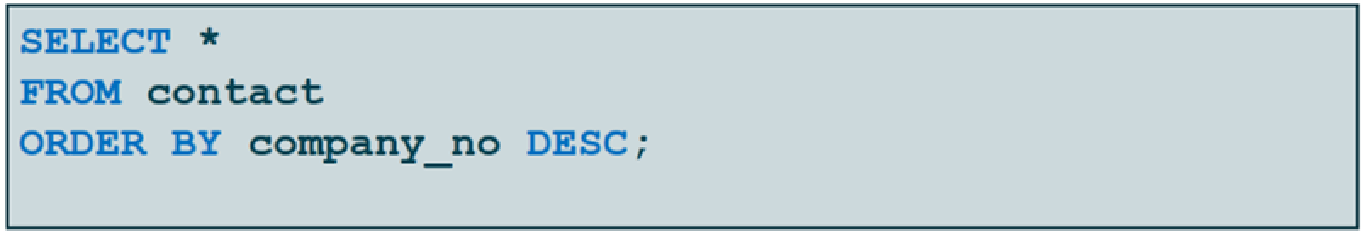


```
SELECT *
FROM contact
ORDER BY company_no ASC,contact_code ASC;
```

- What happens if you omit the ASC?
- What happens is you replace company_no by 1?
- What happens is you replace contact_code by 2?
- What happens if you swop the ORDER BY columns?

SQL: Sorting

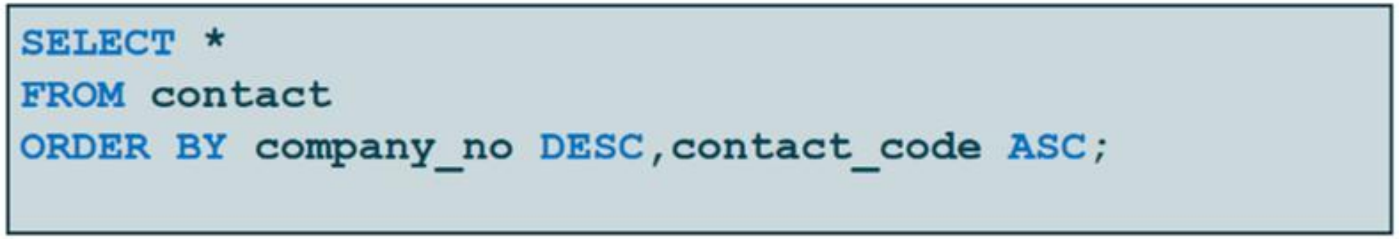
SELECT with ORDER BY Descending



```
SELECT *
FROM contact
ORDER BY company_no DESC;
```

- What happens if you omit the DESC?
- What happens if you swop the ORDER BY columns?

SELECT with ORDER BY Combination

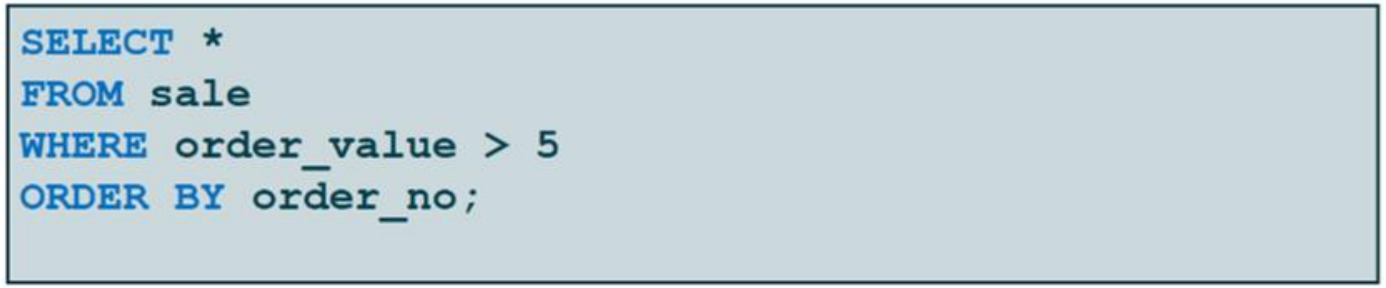


```
SELECT *
FROM contact
ORDER BY company_no DESC,contact_code ASC;
```

- What happens if you swop the ASC and DESC?
- What happens if you swop the ORDER BY columns?
- Think of an example when this might this be required?

SQL: Where

SELECT with WHERE using Relational Operator



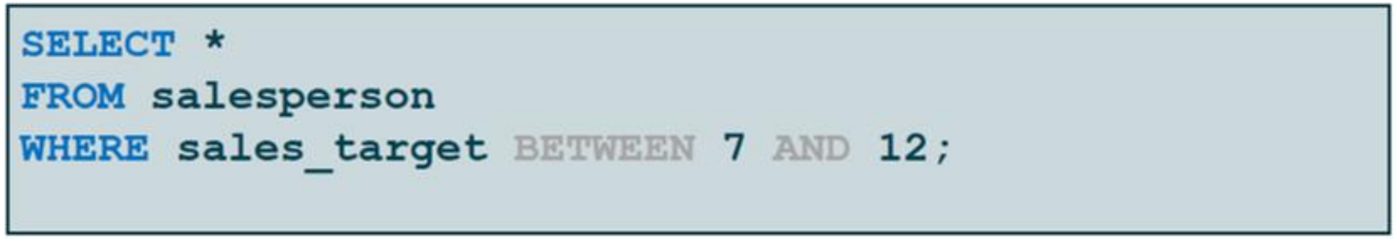
```
SELECT *
FROM sale
WHERE order_value > 5
ORDER BY order_no;
```

You can modify the condition with other basic operators

- <   – Less than
- >= – Greater than or equal to
- <= – Less than or equal to
- =   – Equal to
- <> – Not equal to

SQL: Where

SELECT with WHERE using BETWEEN

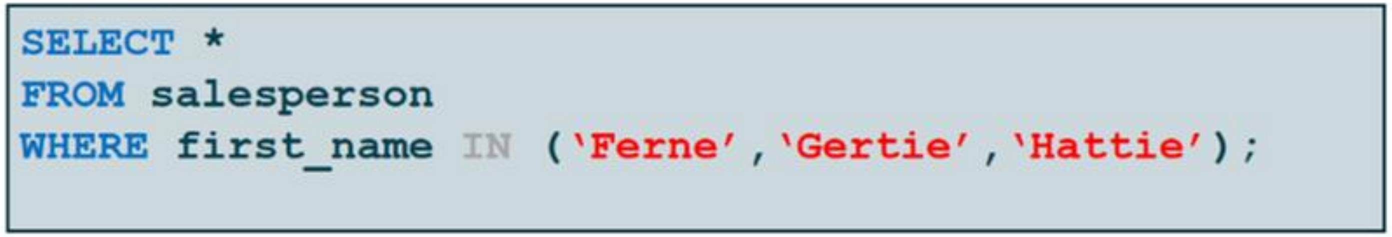


```
SELECT *
FROM salesperson
WHERE sales_target BETWEEN 7 AND 12;
```

- Are the values inclusive?
- Modify the condition with other columns and values
- What happens if you specify a higher starting value?
- What happens if you write: NOT BETWEEN 7 AND 12

SQL: Where

SELECT with WHERE using IN

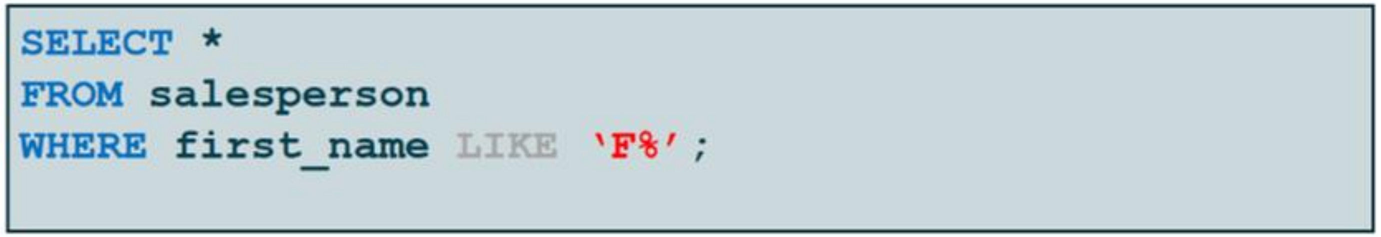


```
SELECT *
FROM salesperson
WHERE first_name IN (‘Ferne’,‘Gertie’,‘Hattie’);
```

- Why is the IN useful?
- What happens if you swop IN with NOT IN?
- Modify the condition with other columns and values

SQL: Where

SELECT with WHERE using LIKE



```
SELECT *
FROM salesperson
WHERE first_name LIKE 'F%';
```

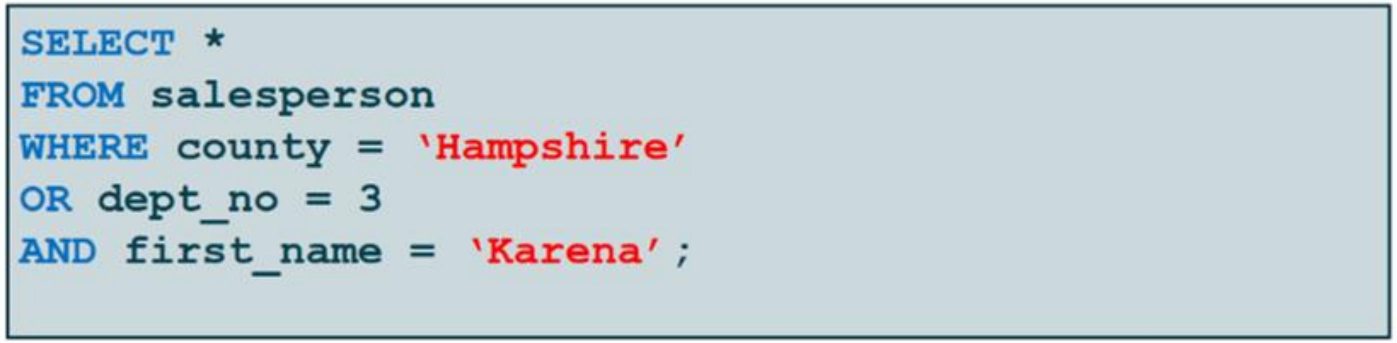
- **'%E'** – Ends in an 'E'
- **'%R%'** – 'R' somewhere in the middle

- **'_E%R%N_'** – **Any character** in first / last position,
  'E' in second position,
  'N' in second last position and
  'R' somewhere in the middle

- Use '=' if you know the full matching value not LIKE

SQL: Where

SELECT with WHERE Multiple AND / OR



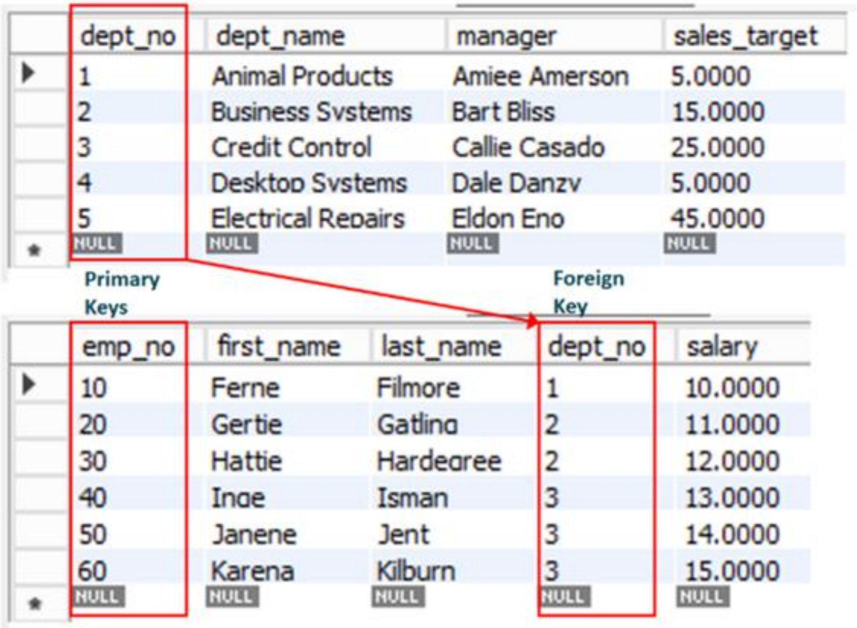
```sql
SELECT *
FROM salesperson
WHERE county = ‘Hampshire’
OR dept_no = 3
AND first_name = ‘Karena’;
```

- Does the AND or OR take precedence?
- Modify with brackets to alter the logic and results

Related tables

Multiple Tables

- **Tables related through Primary / Foreign Key relationship**



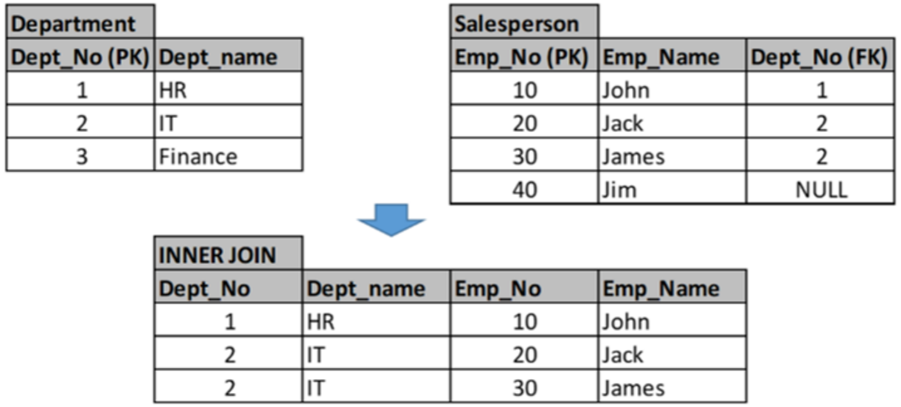
dept_no	dept_name	manager	sales_target
1	Animal Products	Amiee Amerson	5.0000
2	Business Svstems	Bart Bliss	15.0000
3	Credit Control	Callie Casado	25.0000
4	Desktop Svstems	Dale Danzv	5.0000
5	Electrical Repairs	Eldon Eno	45.0000
NULL	NULL	NULL	NULL

emp_no	first_name	last_name	dept_no	salary
10	Ferne	Filmore	1	10.0000
20	Gertie	Gatlina	2	11.0000
30	Hattie	Hardearee	2	12.0000
40	Inae	Isman	3	13.0000
50	Janene	Jent	3	14.0000
60	Karena	Kilburn	3	15.0000
NULL	NULL	NULL	NULL	NULL

Joining tables

Tables in a database are usually related (linked) to one another through data that they have in common (keys). In order to retrieve information that is held in separate tables it is more efficient to JOIN the tables together first than link them in the where clause.

E.g., If we want to know the Employee Names with the names of the departments they are in we could piece this information together first and then SELECT what we want from the joined table:



Department	
Dept_No (PK)	Dept_name
1	HR
2	IT
3	Finance

Salesperson		
Emp_No (PK)	Emp_Name	Dept_No (FK)
10	John	1
20	Jack	2
30	James	2
40	Jim	NULL

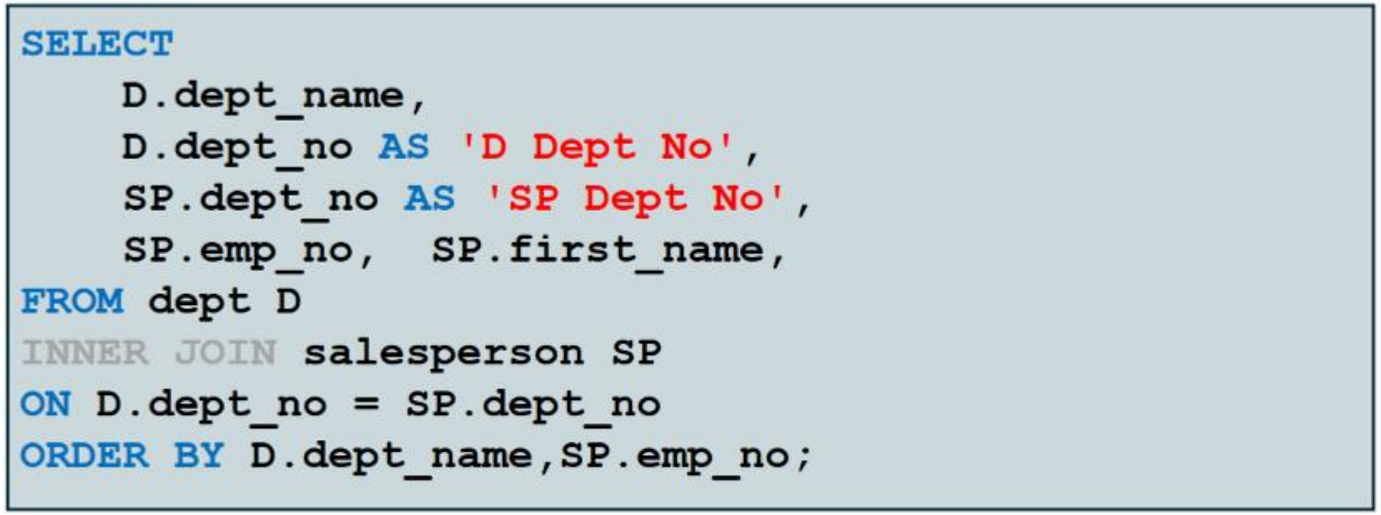
INNER JOIN			
Dept_No	Dept_name	Emp_No	Emp_Name
1	HR	10	John
2	IT	20	Jack
2	IT	30	James

There are two main ways to join tables:

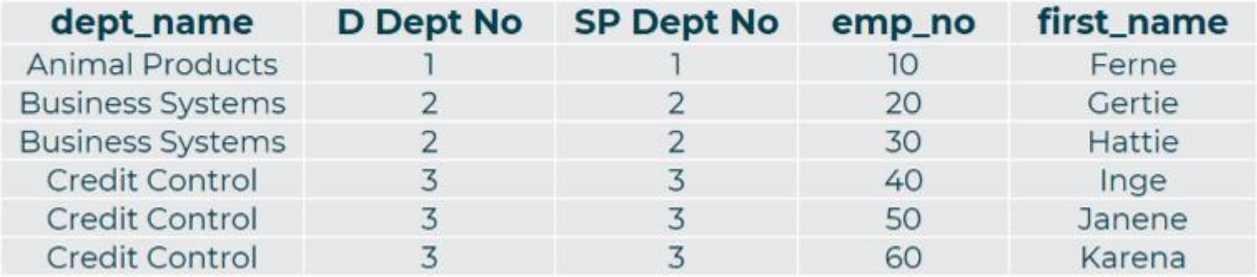
- INNER JOIN
- OUTER JOIN

Joining tables

JOIN of Department with Salesperson



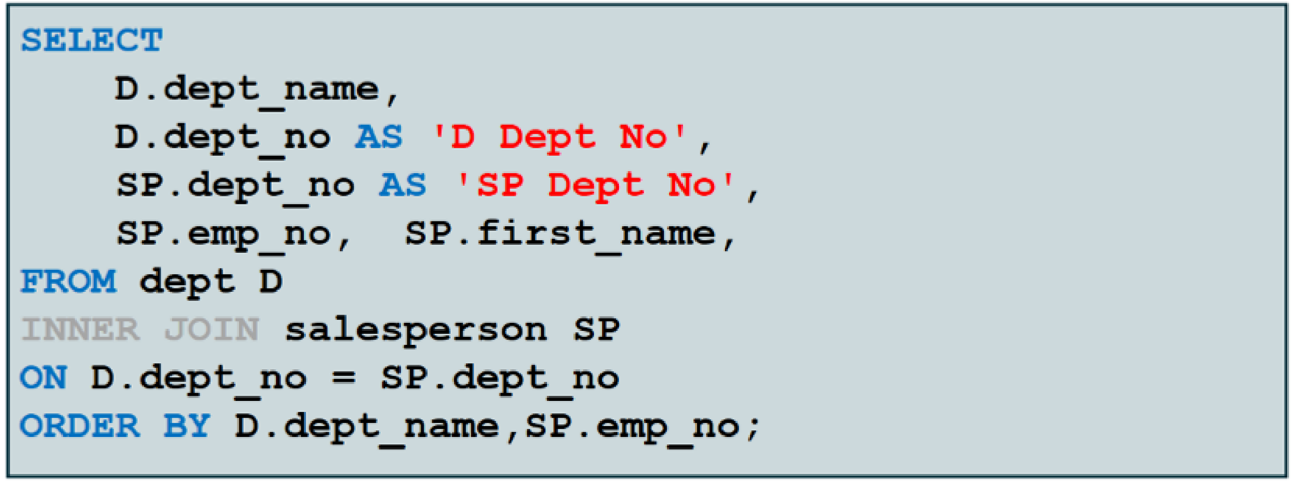
```
SELECT
    D.dept_name,
    D.dept_no AS 'D Dept No',
    SP.dept_no AS 'SP Dept No',
    SP.emp_no,  SP.first_name,
FROM dept D
INNER JOIN salesperson SP
ON D.dept_no = SP.dept_no
ORDER BY D.dept_name,SP.emp_no;
```



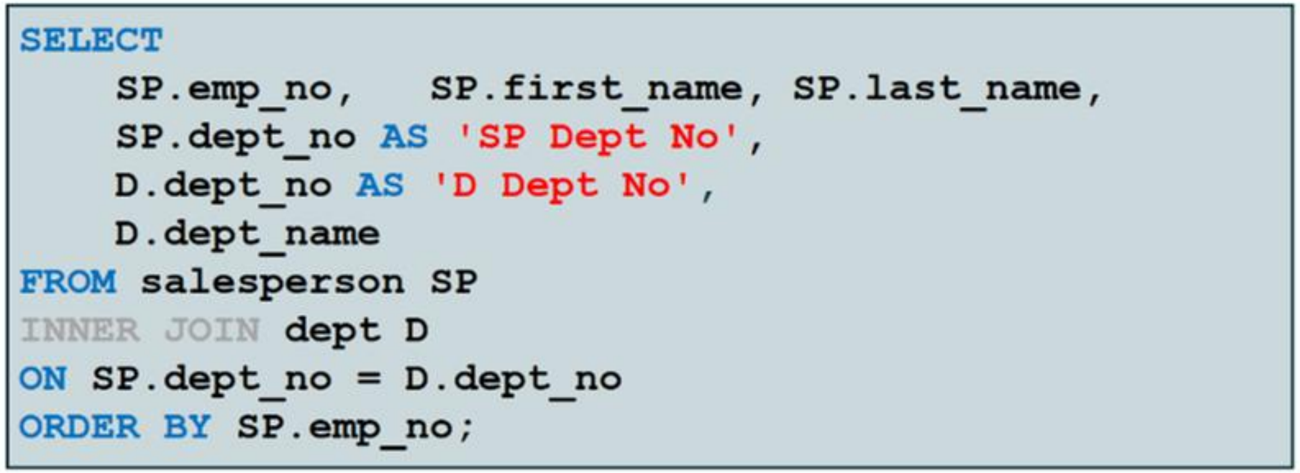
dept_name	D Dept No	SP Dept No	emp_no	first_name
Animal Products	1	1	10	Ferne
Business Systems	2	2	20	Gertie
Business Systems	2	2	30	Hattie
Credit Control	3	3	40	Inge
Credit Control	3	3	50	Janene
Credit Control	3	3	60	Karena

Joining tables

It doesn't matter which table is specified first. These retrieve the same rows (just with the columns in a different order):



```
SELECT
    D.dept_name,
    D.dept_no AS 'D Dept No',
    SP.dept_no AS 'SP Dept No',
    SP.emp_no,  SP.first_name,
FROM dept D
INNER JOIN salesperson SP
ON D.dept_no = SP.dept_no
ORDER BY D.dept_name,SP.emp_no;
```

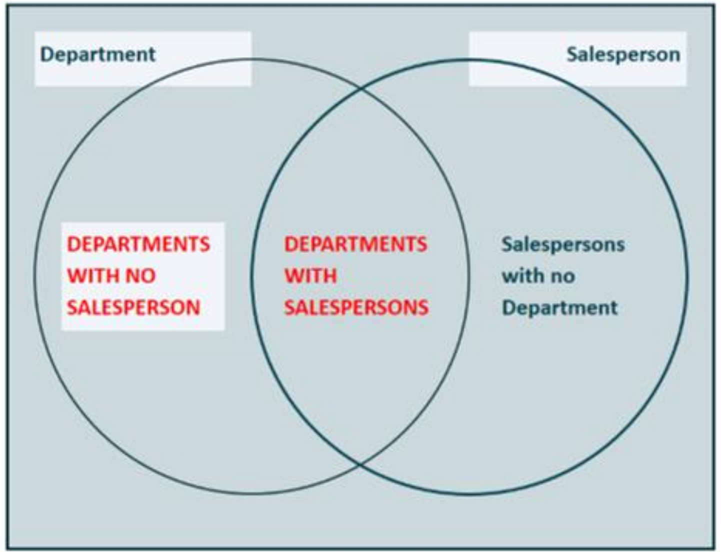


```
SELECT
    SP.emp_no,   SP.first_name, SP.last_name,
    SP.dept_no AS 'SP Dept No',
    D.dept_no AS 'D Dept No',
    D.dept_name
FROM salesperson SP
INNER JOIN dept D
ON SP.dept_no = D.dept_no
ORDER BY SP.emp_no;
```

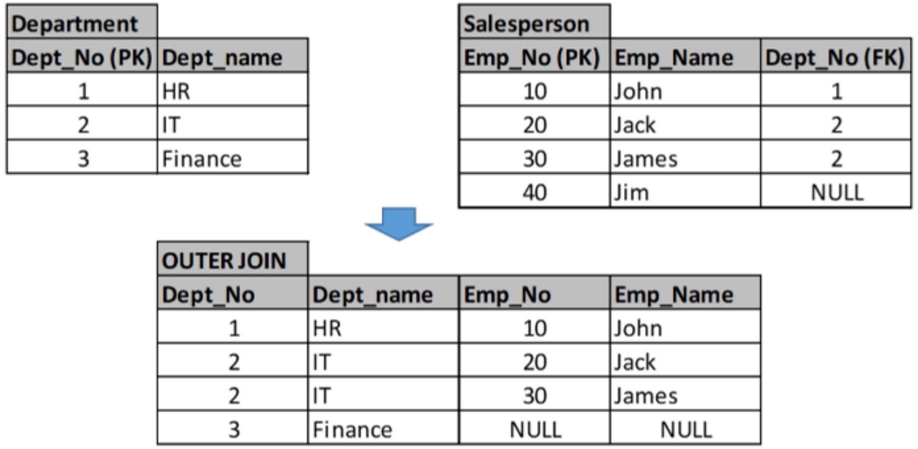
Joining tables

OUTER JOIN

- *List the department and salesperson details for all departments including those that do not have a salesperson assigned to it*



Department
Salesperson
DEPARTMENTS WITH NO SALESPERSON
DEPARTMENTS WITH SALESPERSONS
Salespersons with no Department



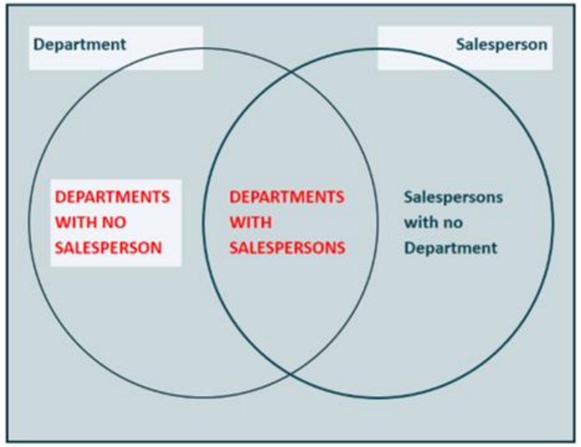
Department	
Dept_No (PK)	Dept_name
1	HR
2	IT
3	Finance

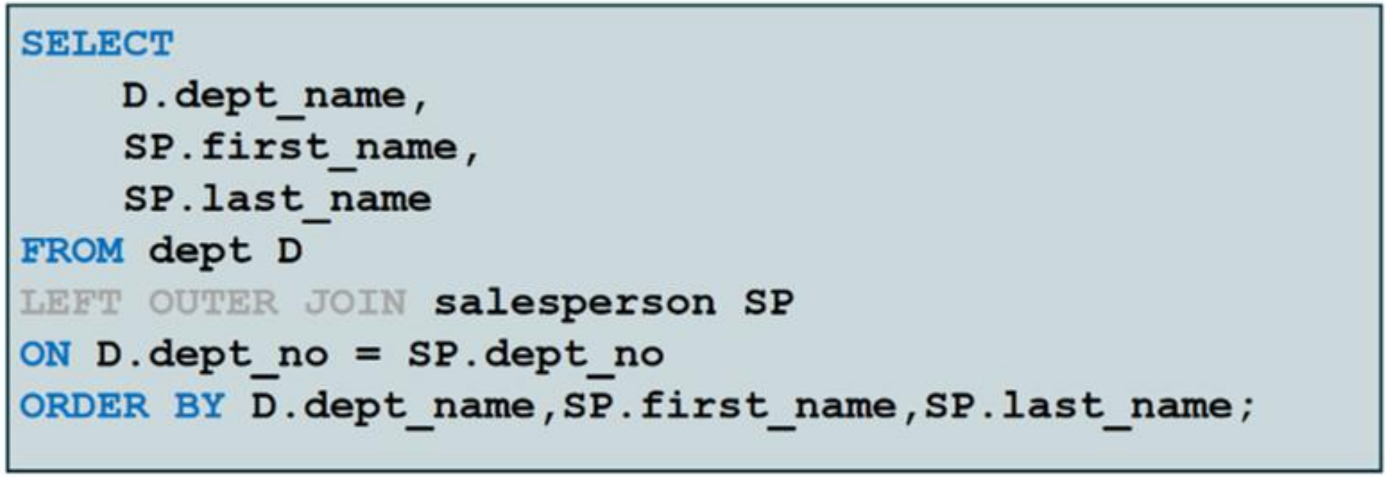
Salesperson		
Emp_No (PK)	Emp_Name	Dept_No (FK)
10	John	1
20	Jack	2
30	James	2
40	Jim	NULL

OUTER JOIN			
Dept_No	Dept_name	Emp_No	Emp_Name
1	HR	10	John
2	IT	20	Jack
2	IT	30	James
3	Finance	NULL	NULL

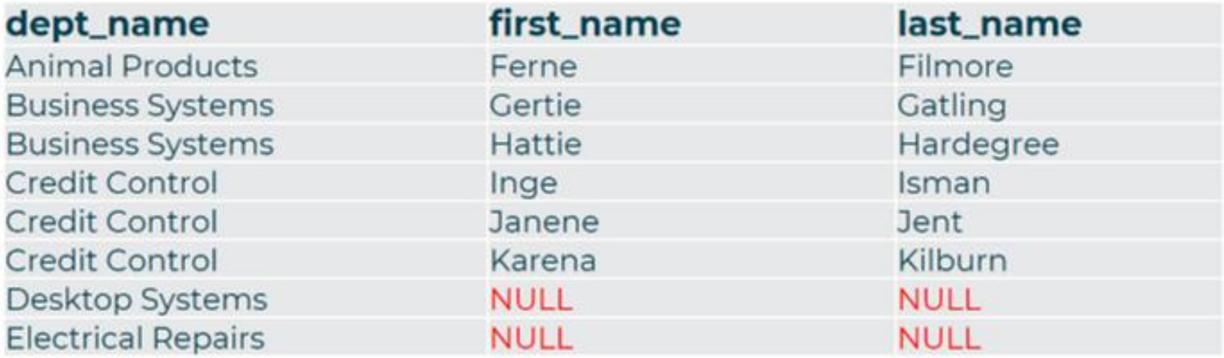
Joining tables

OUTER JOIN of all Departments


Department
Salesperson
DEPARTMENTS WITH NO SALESPERSON
DEPARTMENTS WITH SALESPERSONS
Salespersons with no Department



```
SELECT
    D.dept_name,
    SP.first_name,
    SP.last_name
FROM dept D
LEFT OUTER JOIN salesperson SP
ON D.dept_no = SP.dept_no
ORDER BY D.dept_name,SP.first_name,SP.last_name;
```



dept_name	first_name	last_name
Animal Products	Ferne	Filmore
Business Systems	Gertie	Gatling
Business Systems	Hattie	Hardegree
Credit Control	Inge	Isman
Credit Control	Janene	Jent
Credit Control	Karena	Kilburn
Desktop Systems	NULL	NULL
Electrical Repairs	NULL	NULL

SQL: Aggregates

We may sometimes need to summarise data using aggregates.

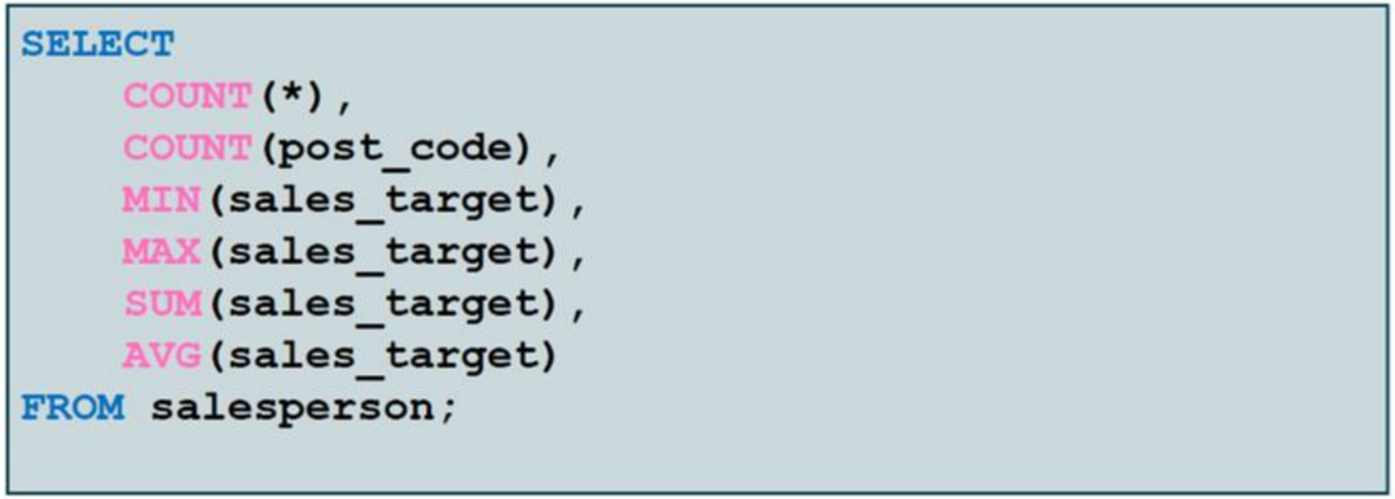
These might include finding totals (summing) or averaging for example.

This could be done by retrieving the data into a collection in Python and performing the calculations there, but that is less efficient –

There are built-in commands that do this in SQL.

SQL: Aggregates

COUNT / MIN / MAX / SUM / AVG

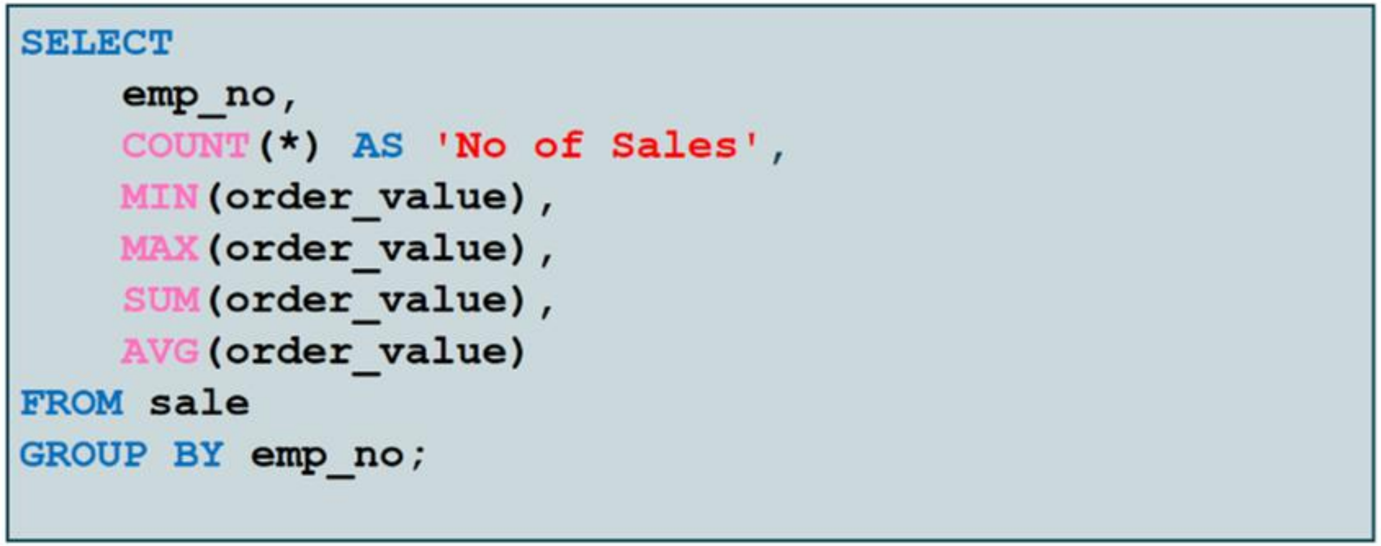


```
SELECT
      COUNT(*),
      COUNT(post_code),
      MIN(sales_target),
      MAX(sales_target),
      SUM(sales_target),
      AVG(sales_target)
FROM salesperson;
```

- **What does the SQL do?**
  - What is the difference between COUNT(*) and COUNT(post_code)?
  - What happens if there is a NULL value in the other Aggregate Functions?

SQL: Aggregates

Subtotal of Single Column

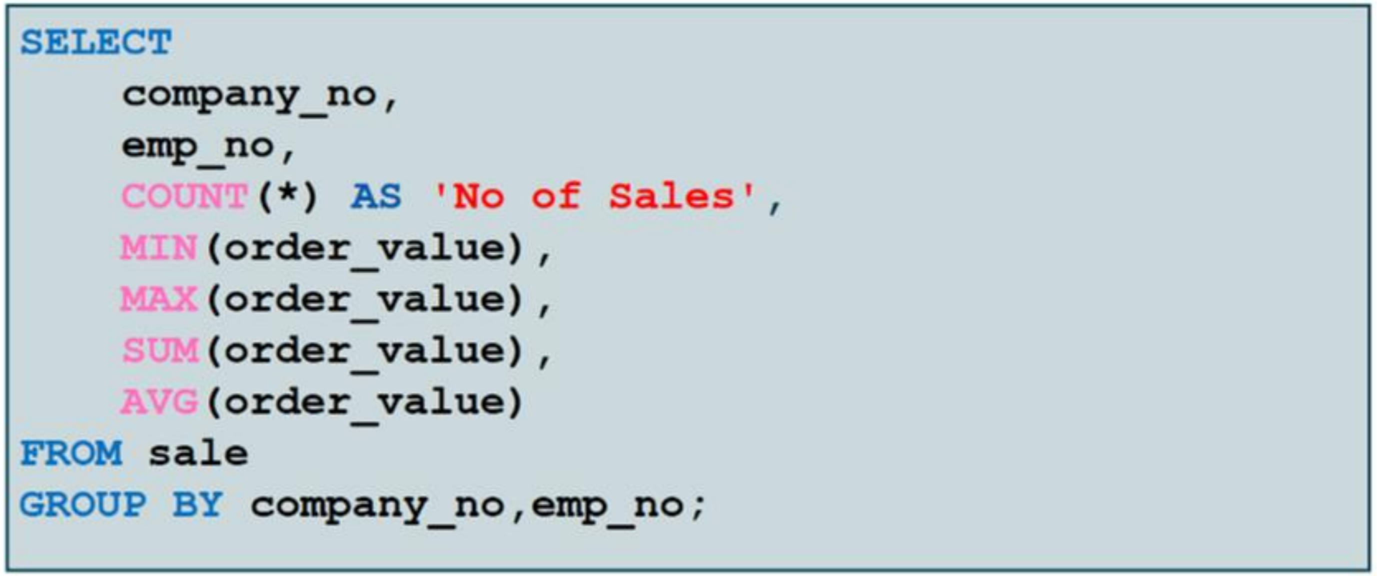


```
SELECT
    emp_no,
    COUNT(*) AS 'No of Sales',
    MIN(order_value),
    MAX(order_value),
    SUM(order_value),
    AVG(order_value)
FROM sale
GROUP BY emp_no;
```

- **What does the SQL do?**
  - Note the field(s) in the GROUP BY need to be specified in the SELECT

SQL: Aggregates

Subtotal of Multiple Columns

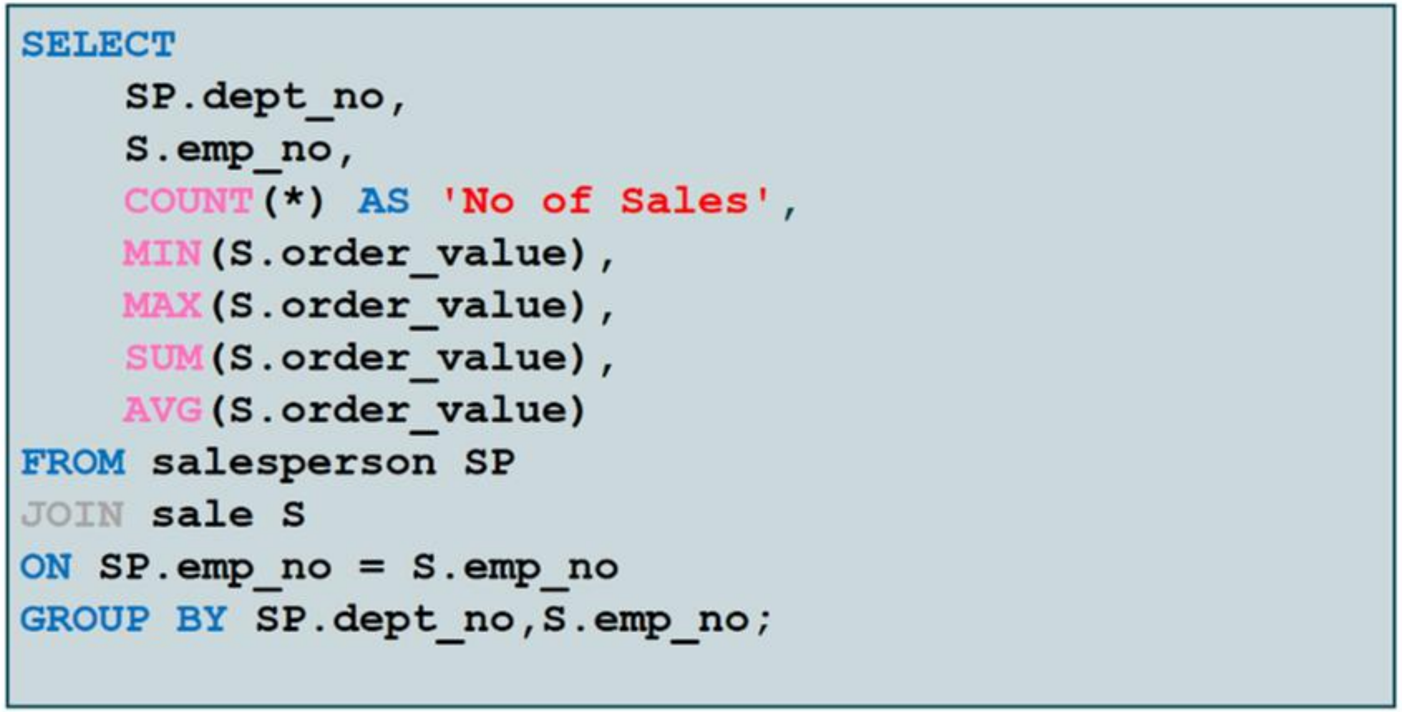


```
SELECT
     company_no,
     emp_no,
     COUNT(*) AS 'No of Sales',
     MIN(order_value),
     MAX(order_value),
     SUM(order_value),
     AVG(order_value)
FROM sale
GROUP BY company_no,emp_no;
```

- What does the SQL do?

SQL: Aggregates

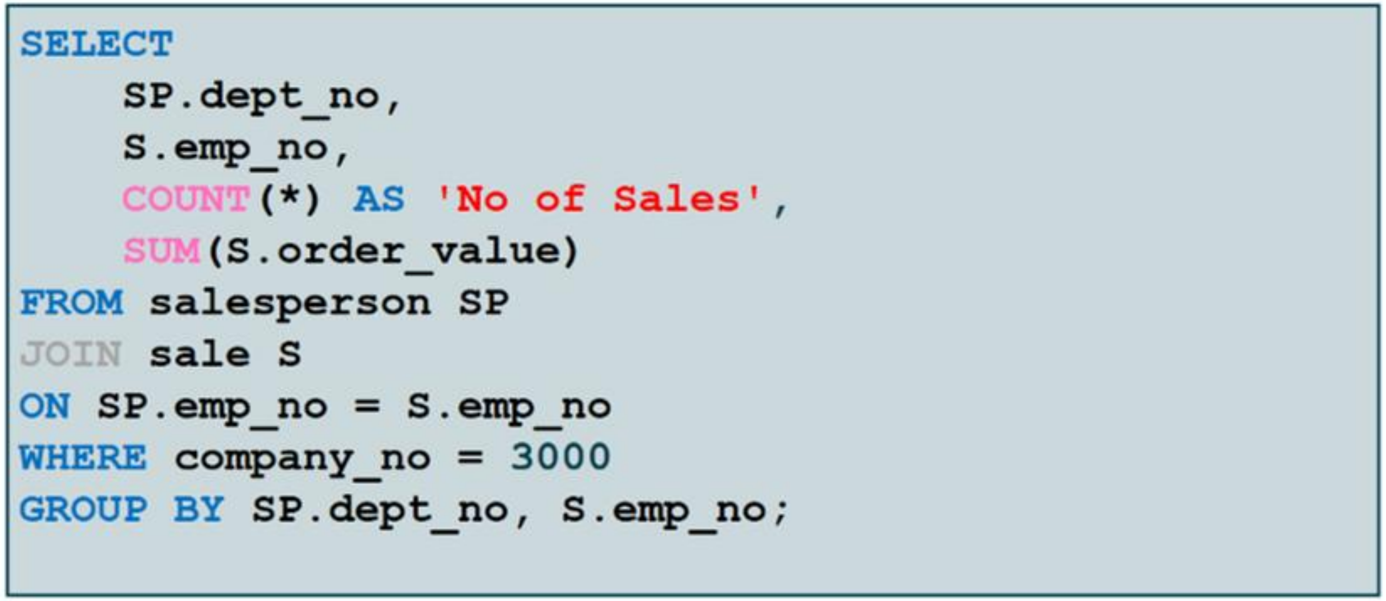
Subtotal of Multiple Tables and Columns



```
SELECT
     SP.dept_no,
     S.emp_no,
     COUNT(*) AS 'No of Sales',
     MIN(S.order_value),
     MAX(S.order_value),
     SUM(S.order_value),
     AVG(S.order_value)
FROM salesperson SP
JOIN sale S
ON SP.emp_no = S.emp_no
GROUP BY SP.dept_no,S.emp_no;
```

SQL: Aggregates

Subtotal Filter Input

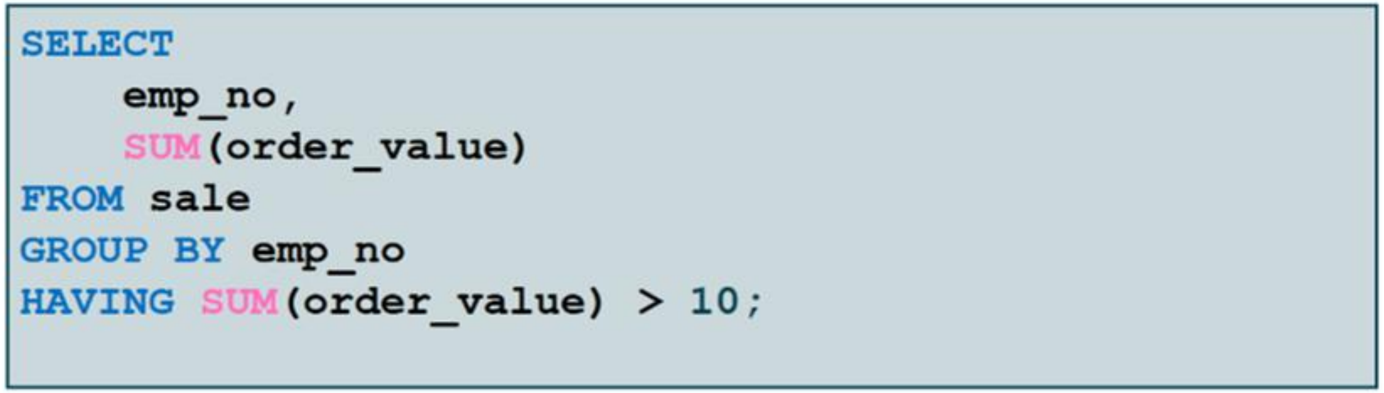


```
SELECT
    SP.dept_no,
    S.emp_no,
    COUNT(*) AS 'No of Sales',
    SUM(S.order_value)
FROM salesperson SP
JOIN sale S
ON SP.emp_no = S.emp_no
WHERE company_no = 3000
GROUP BY SP.dept_no, S.emp_no;
```

- **What does the SQL do?**
  - Modify to use other WHERE conditions

SQL: Aggregates

Filter Results

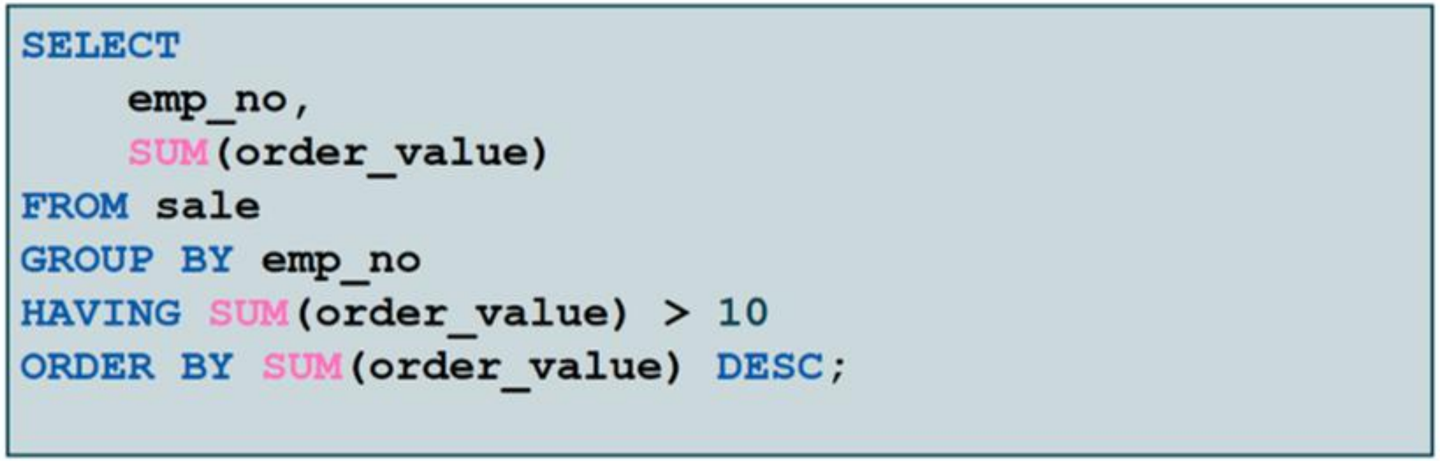


```
SELECT
    emp_no,
    SUM(order_value)
FROM sale
GROUP BY emp_no
HAVING SUM(order_value) > 10;
```

- **What does the SQL do?**
  - Modify to use other Aggregate Functions

SQL: Aggregates

Sorting Results



```
SELECT
    emp_no,
    SUM(order_value)
FROM sale
GROUP BY emp_no
HAVING SUM(order_value) > 10
ORDER BY SUM(order_value) DESC;
```

- **What does the SQL do?**
    - Modify to use other Aggregate Functions

Exception handling

What is Exception Handling?

- **Every operation involving user data entry may result in a crash**
  - Every I/O operation, files databases… may crash
- **Good programs are bullet proof**
  - Bad ones dump error message and die!
- **It may not be possible to make safe every route through an app**
  - But you must try
- **Catch exceptions in functions or let them go up the calls stack**
- **When an exceptions is handled it's cleared as if no error occurred**

EXCEPTION HANDLING EXAMPLE

```
import pyodbc

def showCompany():
    connectionString = r'DRIVER={ODBC Driver 13 for SQL Server};
        SERVER=.\SQLExpress;DATABASE=qastore;Trusted_Connection=yes'
    try:
        conn = pyodbc.connect(connectionString)
        cur = conn.cursor()
        result = cur.execute('SELECT * FROM companys').fetchall()
        conn.close()
        return result
    except:
        return None
#---------------------- main -----------------
rows = showCompany()
if rows != None:
    for row in rows:
        print(row)
else:
    print('Error reading data.')
```

EXCEPTION HANDLING IN FUNCTION

```
import pyodbc

def showCompany():
    connectionString = r'DRIVER={ODBC Driver 13 for SQL Server};
                SERVER=.\SQLExpress;DATABASE=qastore;Trusted_Connection=yes'
    try:
        conn = pyodbc.connect(connectionString)
        cur = conn.cursor()
        result = cur.execute('SELECT * FROM company').fetchall()
        conn.close()
        return result
    except Exception as ex:
        print("Error: ", ex)
        return None
#---------------------- main -----------------
rows = showCompany()
if rows != None:
    for row in rows:
        print(row)
```

EXCEPTION HANDLING IN MAIN

```
import pyodbc

def getQAStoreConnection():
    connectionString = r'DRIVER={ODBC Driver 13 for SQL Server};
                                 SERVER=.\SQLExpress;DATABASE=qastore;Trusted_Connection=yes'
    conn = pyodbc.connect(connectionString)
    cur = conn.cursor()
    return conn

def getQAStoreRows(sql):
    try:
        conn = getQAStoreConnection()
        cur = conn.cursor()
        result = cur.execute(sql).fetchall()
        conn.close()
        return result
    except:
        return None
#---------------------- main -----------------
try:
    rows = getQAStoreRows('SELECT * FROM company')
    if rows != None:
        for row in rows:
            print(row)
except:
    print('Error reading data.')
```

LEARNING CHECK

5-10 MINS

Quiz!

1. What command in SQL makes a database?

2. What command allows you to put a row of data into a table?

3. What happens if you use * after the SELECT command?

4. What command allows you to sort the data that you retrieve?

5. How many different ways to filter using WHERE can you think of?

SOLUTIONS

Relational Databases Quiz

Quiz!

1. What command in SQL makes a database?

CREATE

2. What command allows you to put a row of data into a table?

INSERT

3. What happens if you use * after the SELECT command?

All columns are retrieved.

4. What command allows you to sort the data that you retrieve?

ORDER BY

5. How many different ways to filter using WHERE can you think of?

These have been covered in this section:

Arithmetical operators: <. <=, >, >=, =, <>

BETWEEN .. AND ..

IN ..

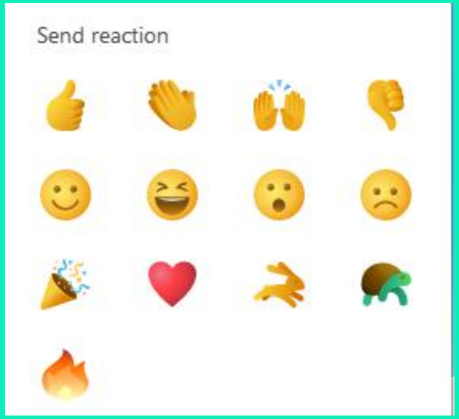
LIKE ..

Logical operators: AND, OR

Labs

Please refer to the lab instructions.

END OF SECTION



An introduction to relational databases

- Interacting with a database
- Creating tables
- Using Python to Insert, Update, and Delete
- Using Python to read from a database
- Joins
- Aggregates
- Exception Handling

- To be able to break down a complex problem into more manageable sub-tasks.
- To create code that can be re-used, that is simple, and easily maintained.

REMINDER: TAKE A BREAK!

10.30 - 10.40
11.40 – 11.50
12.50 – 13.30
14.30 – 14.40
15.40 – 15.50

Your Body on Walking

Ridiculously simple, astonishingly powerful, scientifically proven by study after study: Sneaking in a few minutes a day can transform your health, body, and mind. Why are you still sitting?

BRAIN: Just 2 hours of walking a week can reduce your risk of stroke by 30%.

MEMORY: 40 minutes 3 times a week protects the brain region associated with planning and memory.

MOOD: 30 minutes a day can reduce symptoms of depression by 36%.

HEALTH: Logging 3,500 steps a day lowers your risk of diabetes by 29%.

LONGEVITY: 75 minutes a week of brisk walking can add almost 2 years to your life.

HEART: 30 to 60 minutes most days of the week drastically lowers your risk of heart disease.

BONES: 4 hours a week can reduce the risk of hip fractures by up to 43%.

WEIGHT: A daily 1-hour walk can cut your risk of obesity in half.