SQL

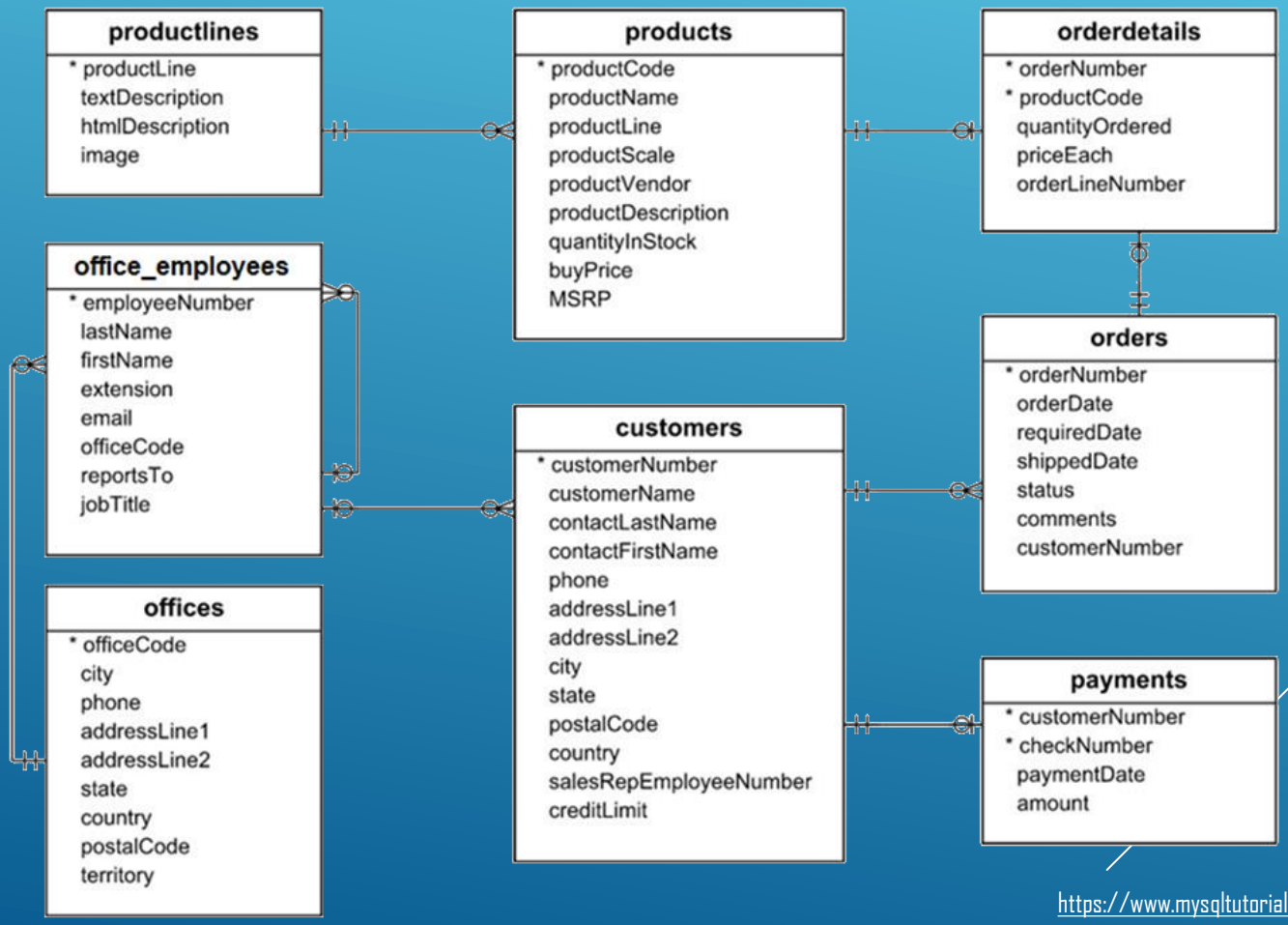
PART V (Common Table Expressions)

420-983-VA Introduction to Databases using Oracle Adrian Onet

SQL

Data Manipulation Language (DML)

Sample Models Schema. Describes an automotive models manufacturer and its sales.



https://www.mysqltutorial.org/mysql-sample-database.aspx

SQL

Data Manipulation Language (DML)

Common Table Expressions (CTE)

A common table expression is a named temporary result set that exists only within the execution scope of a single SELECT statement. Note that common table expressions together with the associated SELECT statement can be used as nested queries inside INSERT/UPDATE/DELETE.

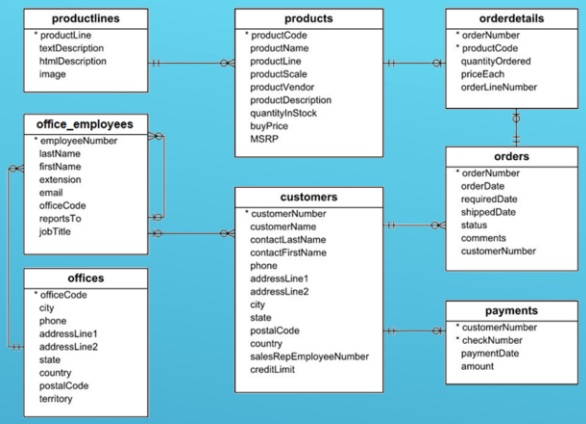
except

```
WITH cte_name_1 [(column_list_1)] AS (
    query_1
) [, cte_name_2 [(column_list_2)] AS (
    query_2
) …]
SELECT statement using cte_name;
```

SQL

Data Manipulation Language (DML)

Common Table Expressions (CTE)

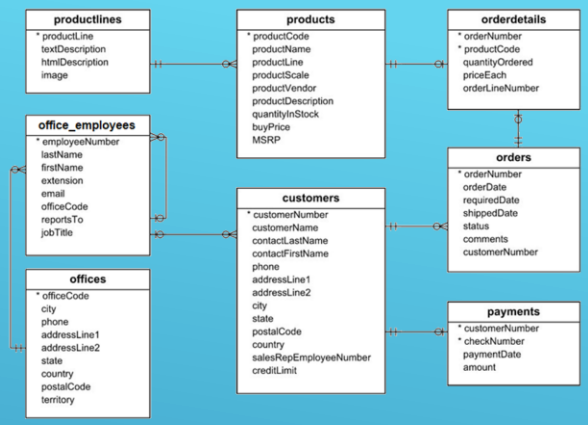


Return number of customers for each of the salesRepEmployee named (firstName) 'Leslie'. Schema details can be found here.

SQL

Data Manipulation Language (DML)

Common Table Expressions (CTE)



Return number of customers for each of the salesRepEmployee named (firstName) 'Leslie'. Schema details can be found here.

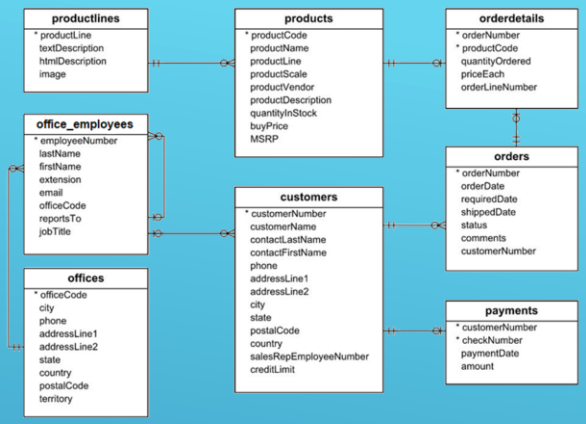
Temporary result used below.

```
WITH leslie_employee (employee_number) AS
(
    SELECT employeeNumber FROM office_employees WHERE firstName='Leslie'
)
SELECT salesRepEmployeeNumber, count(*) AS totalCustomers FROM customers
    WHERE salesRepEmployeeNumber IN (SELECT employee_number FROM leslie_employee)
    GROUP BY salesRepEmployeeNumber;
```

SQL

Data Manipulation Language (DML)

Common Table Expressions (CTE)



Return number of customers for each of the salesRepEmployee named (firstName) 'Leslie'. Schema details can be found here.

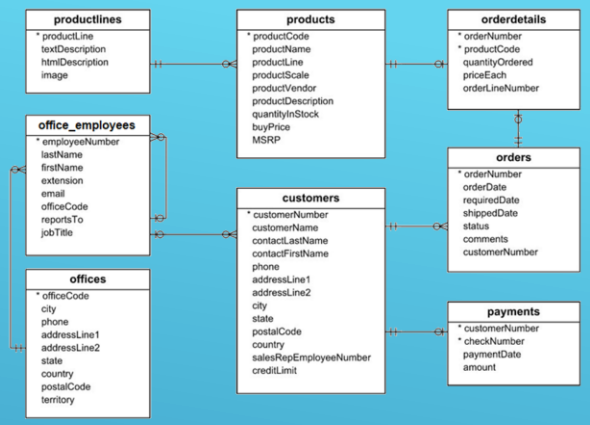
Temporary result used below.

```
WITH leslie_employee (employee_number) AS
(
    SELECT employeeNumber FROM office_employees WHERE firstName='Leslie'
)
SELECT salesRepEmployeeNumber, count(*) AS totalCustomers FROM customers
    WHERE salesRepEmployeeNumber IN (SELECT employee_number FROM leslie_employee)
    GROUP BY salesRepEmployeeNumber;
```

SQL

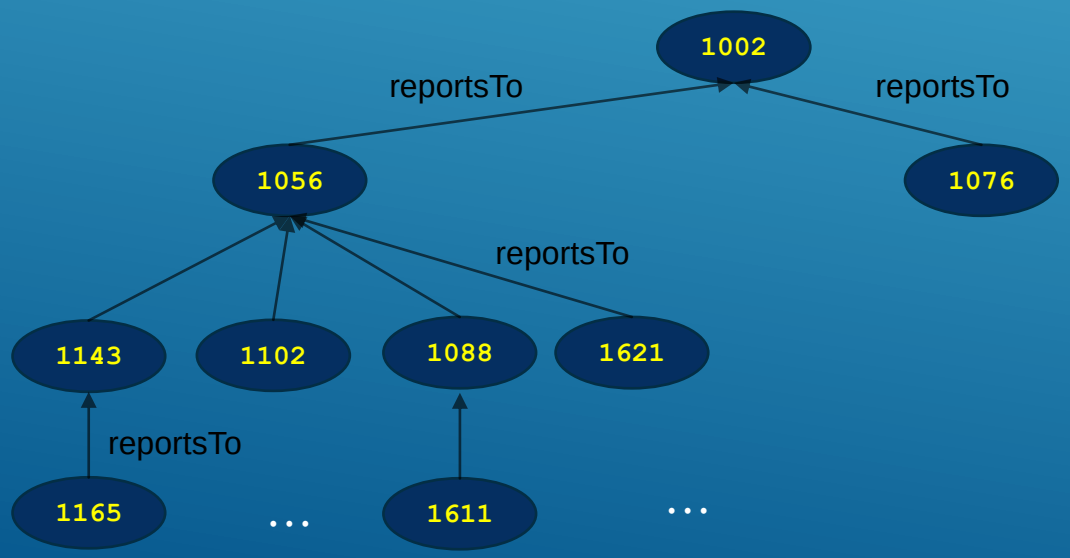
Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

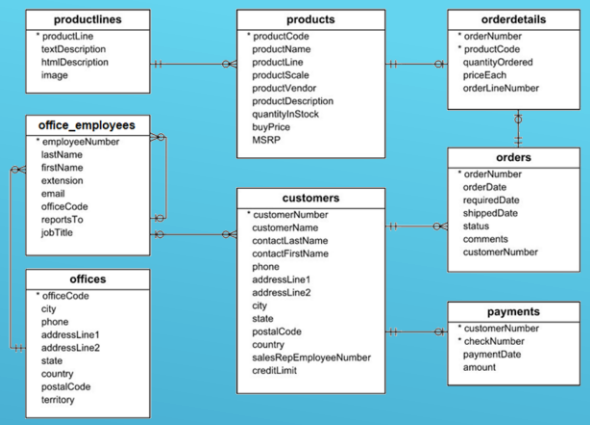
For example, consider this part of the managerial hierarchy in the office_employees table:



SQL

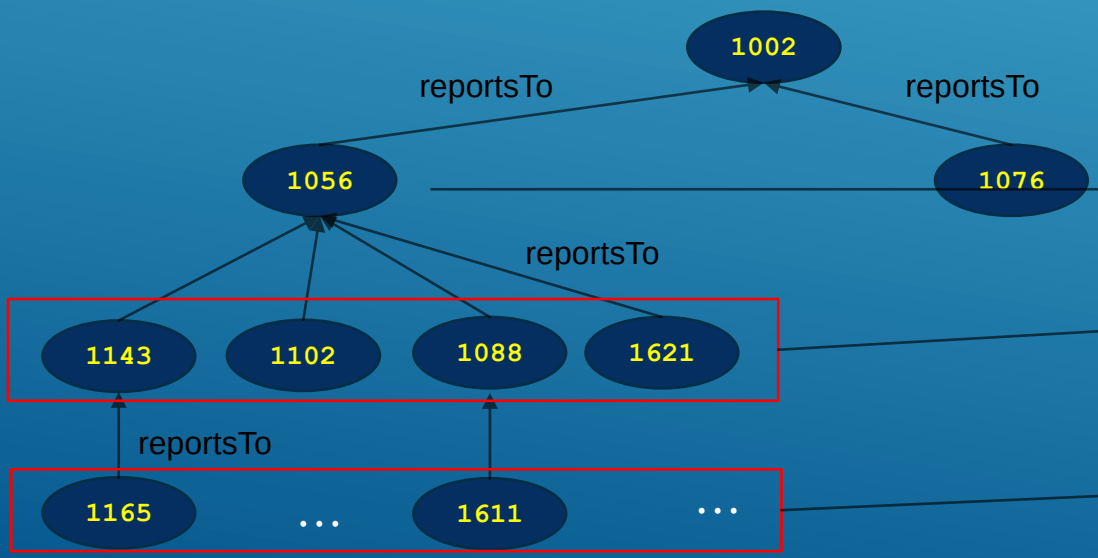
Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

For example, consider this part of the managerial hierarchy in the office_employees table:



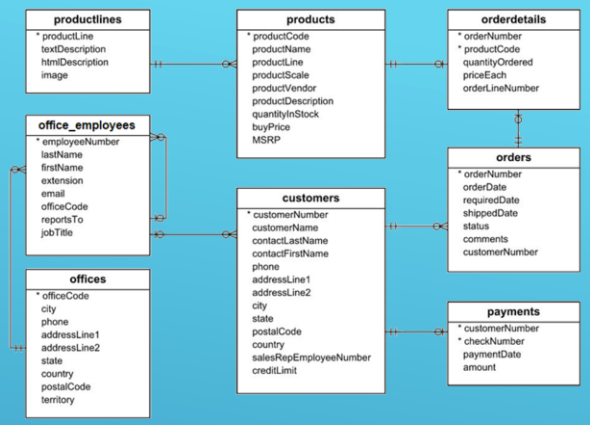
For employeeId 1056 the expected result will look like:

ENumber	Level
1056	0
1143	1
1102	1
1088	1
1621	1
1165	2
1611	2

SQL

Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

MySQL PostgreSQL Microsoft SQL Server

RECURSIVE not required.
Enforced to use **UNION ALL**.

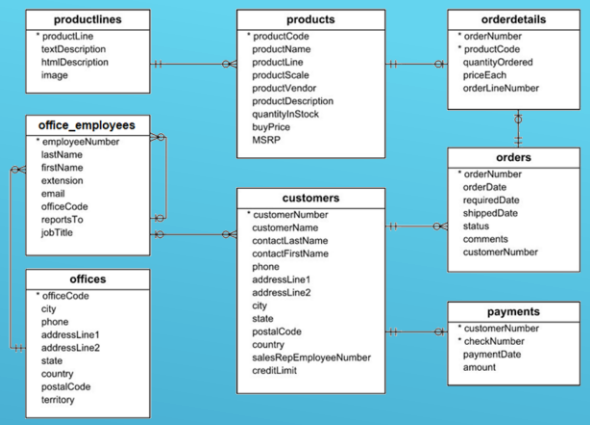
```
WITH RECURSIVE empHierarchy(employeeNumber, level)
AS
(
SELECT employeeNumber, 0 AS level FROM office_employees
    WHERE employeeNumber = 1056
UNION
SELECT E.employeeNumber, level + 1 AS level
  FROM empHierarchy H INNER JOIN office_employees E
                          ON E.reportsTo=H.employeeNumber
)
SELECT * FROM empHierarchy;
```

ENumber	Level
1056	0
1143	1
1102	1
1088	1
1621	1
1165	2
1611	2

SQL

Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

Anchor query, the starting point for the recursion.

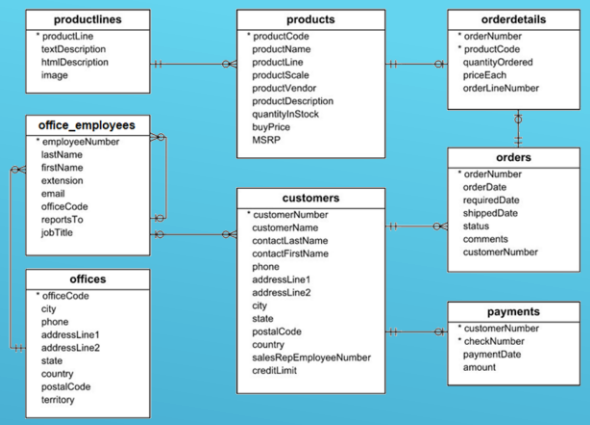
```
WITH RECURSIVE empHierarchy(employeeNumber, level)
AS
(
SELECT employeeNumber, 0 AS level FROM office_employees
    WHERE employeeNumber = 1056
UNION
SELECT E.employeeNumber, level + 1 AS level
  FROM empHierarchy H INNER JOIN office_employees E
                          ON E.reportsTo=H.employeeNumber
)
SELECT * FROM empHierarchy;
```

ENumber	Level
1056	0

SQL

Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

First iterative execution (level 1)

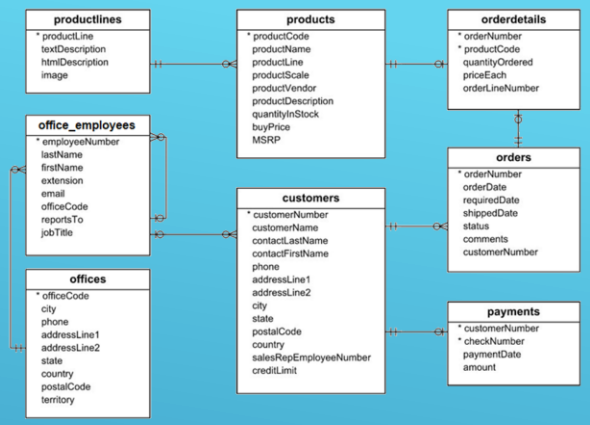
```
WITH RECURSIVE empHierarchy(employeeNumber, level)
AS
(
SELECT employeeNumber, 0 AS level FROM office_employees
    WHERE employeeNumber = 1056
UNION
SELECT E.employeeNumber, level + 1 AS level
  FROM empHierarchy H INNER JOIN office_employees E
                          ON E.reportsTo=H.employeeNumber
)
SELECT * FROM empHierarchy;
```

ENumber	Level
1056	0
1143	1
1102	1
1088	1
1621	1

SQL

Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

```
WITH RECURSIVE empHierarchy(employeeNumber, level)
AS
(
SELECT employeeNumber, 0 AS level FROM office_employees
    WHERE employeeNumber = 1056
UNION
SELECT E.employeeNumber, level + 1 AS level
  FROM empHierarchy H INNER JOIN office_employees E
                          ON E.reportsTo=H.employeeNumber
)
SELECT * FROM empHierarchy;
```

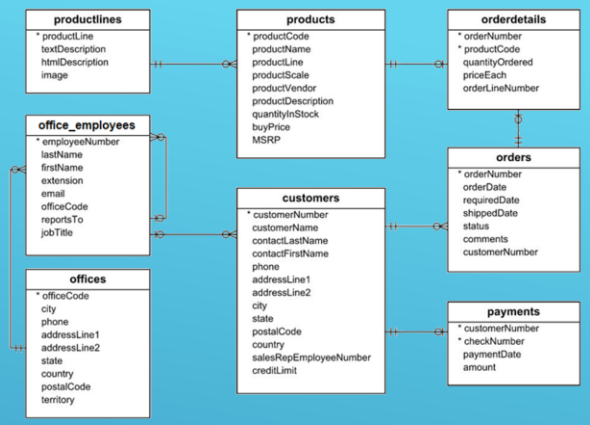
First iterative execution (level 2)

ENumber	Level
1056	0
1143	1
1102	1
1088	1
1621	1
1165	2
1611	2

SQL

Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

```
WITH RECURSIVE empHierarchy(employeeNumber, level)
AS
(
SELECT employeeNumber, 0 AS level FROM office_employees
    WHERE employeeNumber = 1056
UNION
SELECT E.employeeNumber, level + 1 AS level
  FROM empHierarchy H INNER JOIN office_employees E
                          ON E.reportsTo=H.employeeNumber
)
SELECT * FROM empHierarchy
OPTION (MAXRECURSION 10);
```

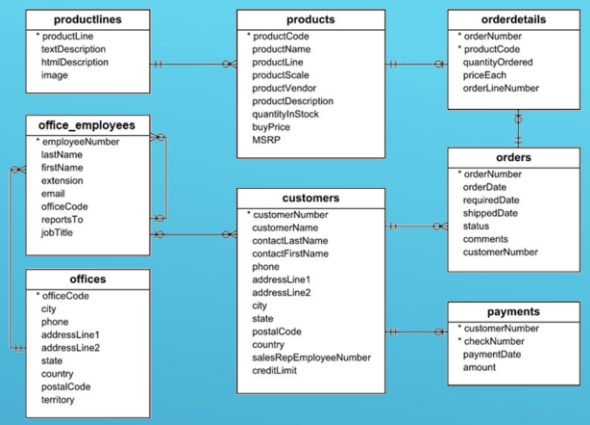
Setting max recursion level.

SQL Server

SQL

Data Manipulation Language (DML)

Hierarchical queries using CTE



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

Setting max recursion level. MySQL →

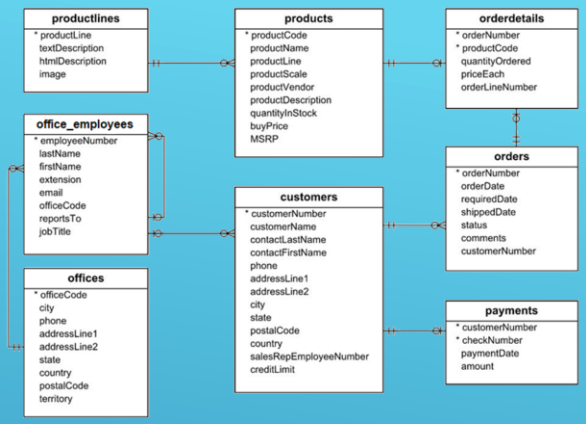
```
SET @@cte_max_recursion_depth=10;
```

```
WITH RECURSIVE empHierarchy(employeeNumber, level)
AS
(
SELECT employeeNumber, 0 AS level FROM office_employees
    WHERE employeeNumber = 1056
UNION
SELECT E.employeeNumber, level + 1 AS level
  FROM empHierarchy H INNER JOIN office_employees E
                          ON E.reportsTo=H.employeeNumber
)
SELECT * FROM empHierarchy;
```

SQL

Data Manipulation Language (DML)

Hierarchical queries using CONNECT BY



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

ORACLE IBM DB2

source

Anchor query, the starting point for the recursion.

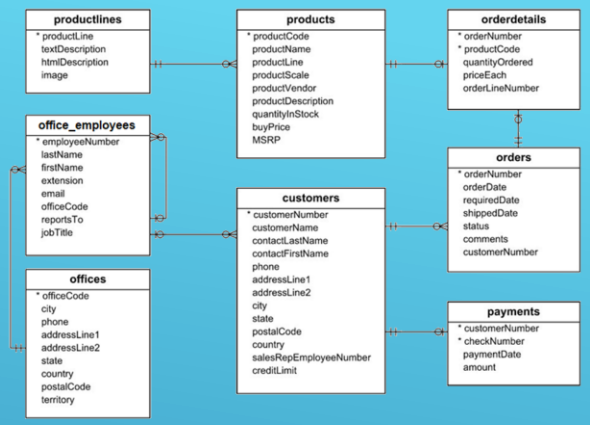
```
SELECT employeeNumber, LEVEL-1 FROM office_employees
START WITH employeeNumber=1056
CONNECT BY reportsto=PRIOR employeeNumber;
```

ENumber	Level
1056	0

SQL

Data Manipulation Language (DML)

Hierarchical queries using CONNECT BY



For a given manager find all the office_employees under him (based on the reportsTo link). Schema details can be found here.

ORACLE IBM DB2

source

Anchor query, the starting point for the recursion.

```
SELECT employeeNumber, LEVEL-1 FROM office_employees
START WITH employeeNumber=1056
CONNECT BY reportsto=PRIOR employeeNumber;
```

ENumber	Level
1056	0
1143	1
1102	1
1088	1
1621	1
1165	2
1611	2

First iteration. From the source take those records which has the reportsTo as employeeNumbers generated at previous step.

```
CONNECT BY PRIOR employeeNumber=reportsto;
```

SQL

Data Manipulation Language (DML)

Recursive queries – Duplicate Rows

Consider below players table with 2 columns: name as PK and repl the replication number. Return the list of player names replicated the number of times given by the repl value.

```
CREATE TABLE players(name VARCHAR(20) PRIMARY KEY, repl int NOT NULL);

INSERT INTO players(name, repl) VALUES('John',3);
INSERT INTO players(name, repl) VALUES('Leslie',2);
INSERT INTO players(name, repl) VALUES('Marry',4);
```

name	repl
John	3
Leslie	2
Marry	4

Result

name
John
John
John
Leslie
Leslie
Marry
Marry
Marry
Marry

SQL

Data Manipulation Language (DML)

Recursive queries – Duplicate Rows

Consider below players table with 2 columns: name as PK and repl the replication number. Return the list of player names replicated the number of times given by the repl value.

name	repl
John	3
Leslie	2
Marry	4

Microsoft SQL Server

```
WITH recPlay AS (
SELECT name, repl-1 AS repl FROM players WHERE repl>0
UNION ALL
SELECT P.name, R.repl-1 FROM players P INNER JOIN recPlay R ON P.name=R.name
 WHERE R.repl>0
)
SELECT name FROM recPlay ORDER BY name;
```

SQL

Data Manipulation Language (DML)

Recursive queries – Duplicate Rows

Consider below players table with 2 columns: name as PK and repl the replication number. Return the list of player names replicated the number of times given by the repl value.

name	repl
John	3
Leslie	2
Marry	4

```
WITH RECURSIVE recPlay AS (
SELECT name, repl-1 AS repl FROM players WHERE repl>0
UNION
SELECT P.name, R.repl-1 FROM players P INNER JOIN recPlay R ON P.name=R.name
 WHERE R.repl>0
)
SELECT name FROM recPlay ORDER BY name;
```

SQL

Data Manipulation Language (DML)

Recursive queries – Duplicate Rows

Consider below players table with 2 columns: name as PK and repl the replication number. Return the list of player names replicated the number of times given by the repl value.

name	repl
John	3
Leslie	2
Marry	4

ORACLE

allow using repl field from players table

```
SELECT std.name
  FROM players, LATERAL (SELECT LEVEL FROM DUAL CONNECT BY LEVEL<=players.repl)
 ORDER BY 1;
```

To better understand try this:

```
SELECT LEVEL FROM DUAL CONNECT BY LEVEL<=3);
```

ORACLE

```
WITH rec AS (
SELECT 1 AS level
UNION ALL
SELECT level+1 FROM rec WHERE level<3
)
SELECT level FROM rec;
```

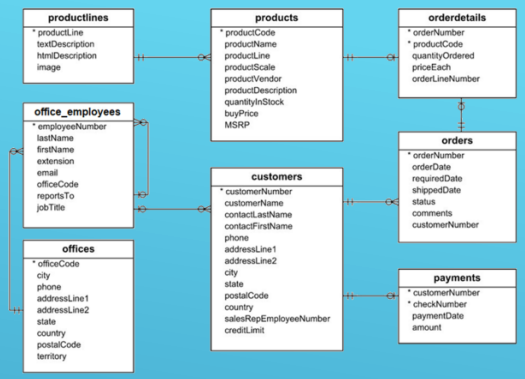
level
1
2
3

SQL Server

SQL

Data Manipulation Language (DML)

Rows denormalization into string



For each office return a comma delimited list of the firstname of each employee.

Example:

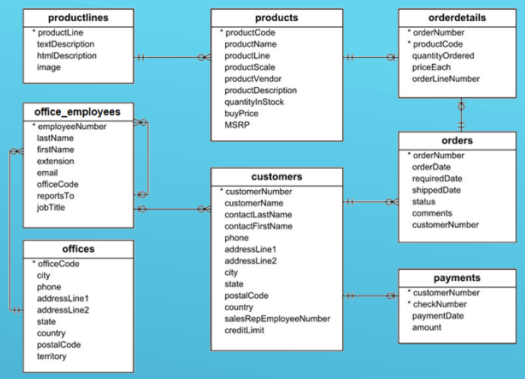
officeCode	firstname
1	Diane
1	Mary
1	Jeff
6	William
4	Gerard
1	Anthony
1	Leslie
1	Leslie
2	Julie
2	Steve
3	Foon Yue
3	George
4	Loui
4	Gerard
4	Pamela
7	Larry
7	Barry
6	Andy
6	Peter
6	Tom
5	Mami
5	Yoshimi
4	Martin

officecode	names
1	Anthony,Diane,Jeff,Leslie,Leslie,Mary
2	Julie,Steve
3	Foon Yue,George
4	Gerard,Gerard,Loui,Martin,Pamela
5	Mami,Yoshimi
6	Andy,Peter,Tom,William
7	Barry,Larry

SQL

Data Manipulation Language (DML)

Rows denormalization into string



For each office return a comma delimited list of the firstname of each employee.

office_employees

officeCode	firstname

field or expression that will be concatenated within the specified group

delimiter used

Order in which values will be concatenated

```
SELECT officecode, LISTAGG(firstname,',') WITHIN GROUP (ORDER BY firstname) AS names
FROM office_employees GROUP BY officecode;
```

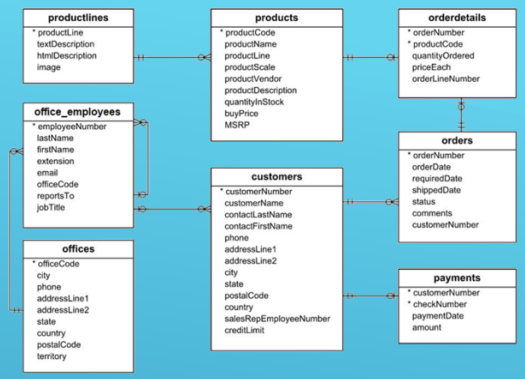
officecode	names
1	Anthony,Diane,Jeff,Leslie,Leslie,Mary
2	Julie,Steve
3	Foon Yue,George
4	Gerard,Gerard,Loui,Martin,Pamela
5	Mami,Yoshimi
6	Andy,Peter,Tom,William
7	Barry,Larry

Note, NULLs are not considered. If the group has only null values NULL value will be returned.

SQL

Data Manipulation Language (DML)

Rows denormalization into string



For each office return a comma delimited list of the firstname of each employee.

office_employees

officeCode	firstname

Eliminate duplicates. Oracle ≥ **19c**

```
SELECT officecode, LISTAGG(DISTINCT firstname,',') WITHIN GROUP (ORDER BY firstname) AS names
FROM office_employees GROUP BY officecode;
```

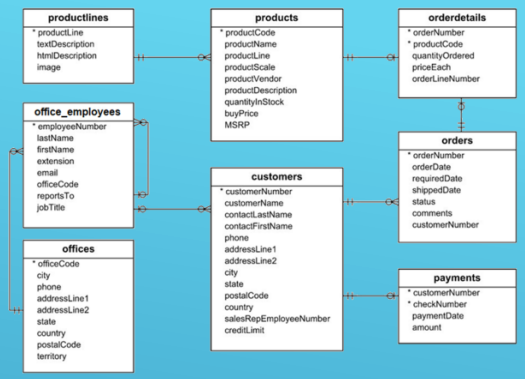
officecode	names
1	Anthony,Diane,Jeff,Leslie,Mary
2	Julie,Steve
3	Foon Yue,George
4	Gerard,Gerard,Loui,Martin,Pamela
5	Mami,Yoshimi
6	Andy,Peter,Tom,William
7	Barry,Larry

Homework: write a query with the same behaviour for Oracle <19c.

SQL

Data Manipulation Language (DML)

Rows denormalization into string



For each office return a comma delimited list of the firstname of each employee.

office_employees

officeCode	firstname

specified the partition for which the grouping is performed

```
SELECT officecode,lastname, firstname,
LISTAGG(DISTINCT firstname,',') WITHIN GROUP (ORDER BY firstname) OVER (PARTITION BY officecode)
AS names FROM office_employees;
```

No group by is specified.

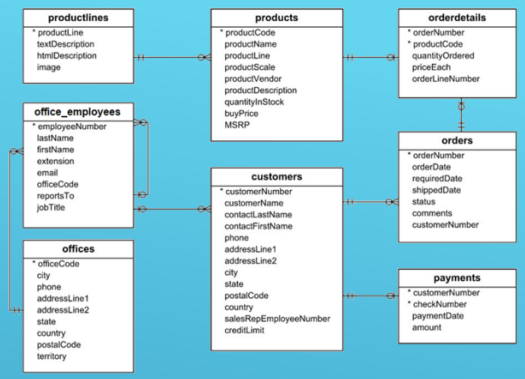
officecode	lastname	firstname	names
1	Bow	Anthony	Anthony,Diane,Jeff,Leslie,Leslie,Mary
1	Murphy	Diane	Anthony,Diane,Jeff,Leslie,Leslie,Mary
1	Firrelli	Jeff	Anthony,Diane,Jeff,Leslie,Leslie,Mary
1	Thompson	Leslie	Anthony,Diane,Jeff,Leslie,Leslie,Mary
1	Jennings	Leslie	Anthony,Diane,Jeff,Leslie,Leslie,Mary
1	Patterson	Mary	Anthony,Diane,Jeff,Leslie,Leslie,Mary
2	Firrelli	Julie	Julie,Steve
2	Patterson	Steve	Julie,Steve
3	Tseng	Foon Yue	Foon Yue,George
3	Vanauf	George	Foon Yue,George

...

SQL

Data Manipulation Language (DML)

Rows denormalization into string



For each office return a comma delimited list of the firstname of each employee.

office_employees

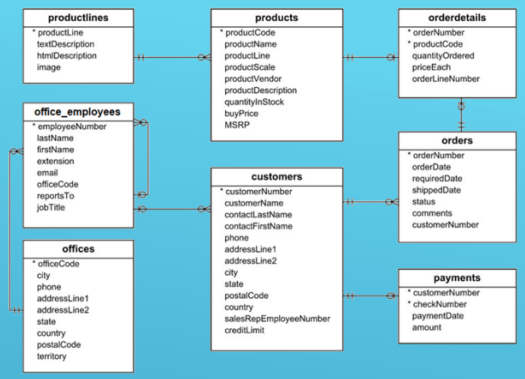
officeCode	firstname

If we want the result specified as an JSON array.

```
SELECT officecode,
JSON_ARRAYAGG(firstname ORDER BY firstname) FROM office_employees GROUP BY officecode;
```

officecode	names
1	["Anthony","Diane","Jeff","Leslie","Leslie","Mary"]
2	["Julie","Steve"]
3	["Foon Yue","George"]
4	["Gerard","Gerard","Loui","Martin","Pamela"]
5	["Mami","Yoshimi"]
6	["Andy","Peter","Tom","William"]
7	["Barry","Larry"]

SQL



Data Manipulation Language (DML)

Rows denormalization into string

For each office return a comma delimited list of the firstname of each employee.

Microsoft SQL Server

```
SELECT officecode, STRING_AGG(firstname,',') WITHIN GROUP (ORDER BY firstname) AS names
FROM office_employees GROUP BY officecode;
```

PostgreSQL

```
SELECT officecode, STRING_AGG(firstname,',' ORDER BY firstname) AS names
FROM office_employees GROUP BY officecode;
```

MySQL

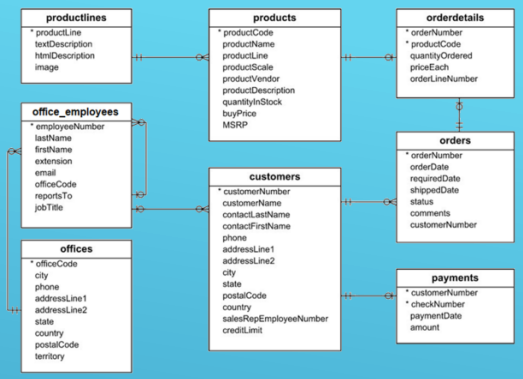
Can be omitted as ',' is default separator.

```
SELECT officecode, GROUP_CONCAT(firstname ORDER BY firstname ASC SEPARATOR ',') AS names
FROM office_employees GROUP BY officecode;
```

SQL

Pagination

Pagination



We want to display 10 orders at the time. Each time the user clicks next page other 10 orders are displayed.

mandatory for pagination.

```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

Ordernumber	...
10100	
10101	
10102	
10103	
10104	
10105	
10106	
10107	
10108	
10109	

```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

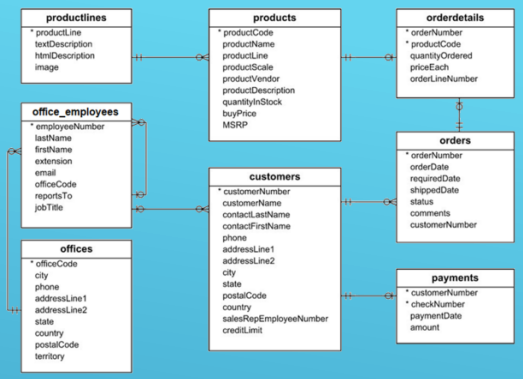
ordernumber	...
10110	
10111	
10112	
10113	
10114	
10115	
10116	
10117	
10118	
10119	

Microsoft SQL Server

SQL

Pagination

Pagination



We want to display 10 orders at the time. Each time the user clicks next page other 10 orders are displayed.

mandatory for pagination.

```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
OFFSET 0 ROWS FETCH FIRST 10 ROWS ONLY;
```

Ordernumber	...
10100	
10101	
10102	
10103	
10104	
10105	
10106	
10107	
10108	
10109	

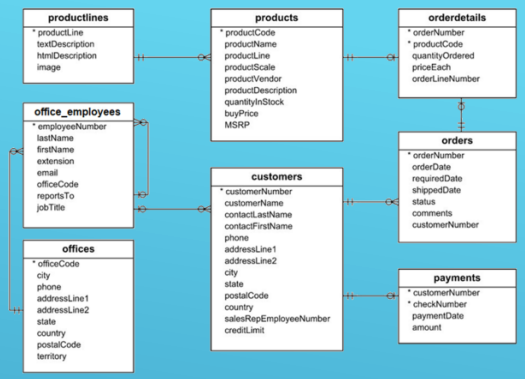
```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
OFFSET 10 ROWS FETCH FIRST 10 ROWS ONLY;
```

ordernumber	...
10110	
10111	
10112	
10113	
10114	
10115	
10116	
10117	
10118	
10119	

SQL

Pagination

Pagination



We want to display 10 orders at the time. Each time the user clicks next page other 10 orders are displayed.

mandatory for pagination.

```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
LIMIT 0 , 10;
```

Ordernumber	...
10100	
10101	
10102	
10103	
10104	
10105	
10106	
10107	
10108	
10109	

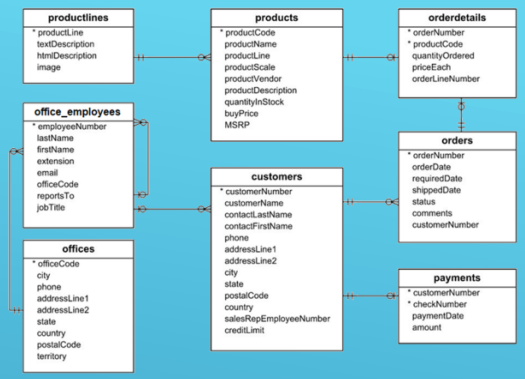
```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
LIMIT 10 , 10;
```

ordernumber	...
10110	
10111	
10112	
10113	
10114	
10115	
10116	
10117	
10118	
10119	

SQL

Pagination

Pagination



We want to display 10 orders at the time. Each time the user clicks next page other 10 orders are displayed.

mandatory for pagination.

```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
LIMIT 10 OFFSET 0;
```

Ordernumber	...
10100	
10101	
10102	
10103	
10104	
10105	
10106	
10107	
10108	
10109	

```
SELECT ordernumber, orderdate, status FROM orders ORDER BY ordernumber
LIMIT 10 OFFSET 10;
```

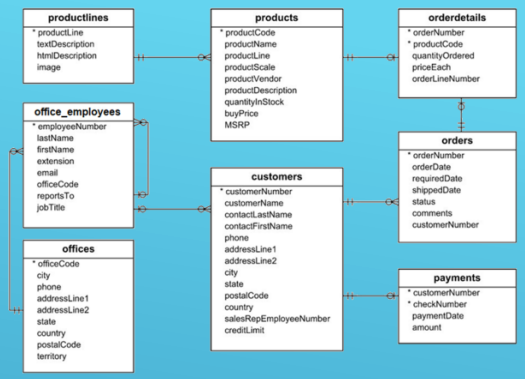
ordernumber	...
10110	
10111	
10112	
10113	
10114	
10115	
10116	
10117	
10118	
10119	

PostgreSQL

SQL

Pagination

Pagination



We want to display 10 orders at the time. Each time the user clicks next page other 10 orders are displayed.

```
SELECT ordernumber, orderdate, status FROM
(SELECT orders.*, rownum AS n FROM orders ORDER BY ordernumber) A
WHERE n BETWEEN 1 AND 10;
```

Ordernumber	...
10100	
10101	
10102	
10103	
10104	
10105	
10106	
10107	
10108	
10109	

```
SELECT ordernumber, orderdate, status FROM
(SELECT orders.*, rownum AS n FROM orders ORDER BY ordernumber) A
WHERE n BETWEEN 11 AND 20;
```

ordernumber	...
10110	
10111	
10112	
10113	
10114	
10115	
10116	
10117	
10118	
10119	