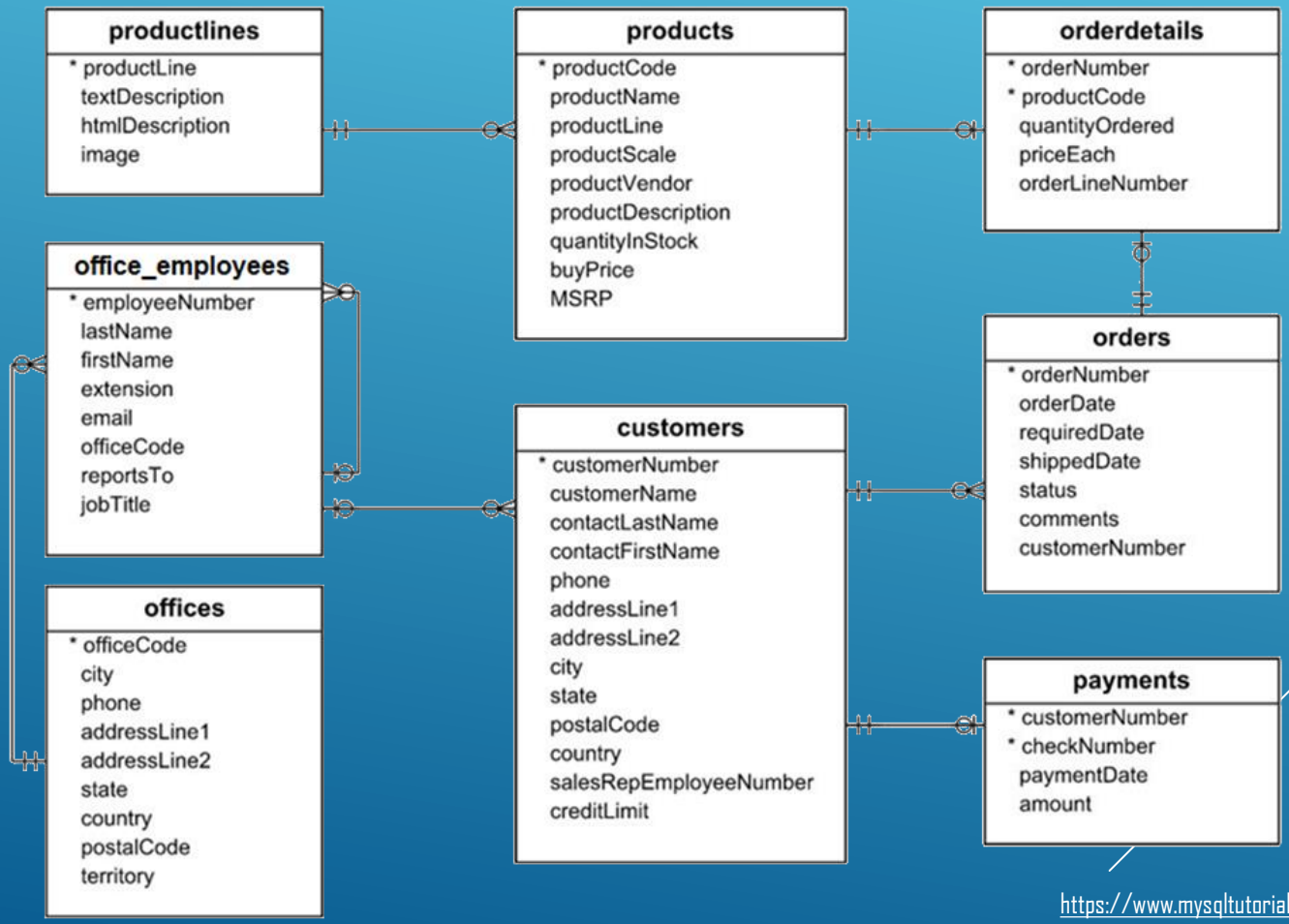
SQL

PART IV (Nested Queries)

SQL

Data Manipulation Language (DML)

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

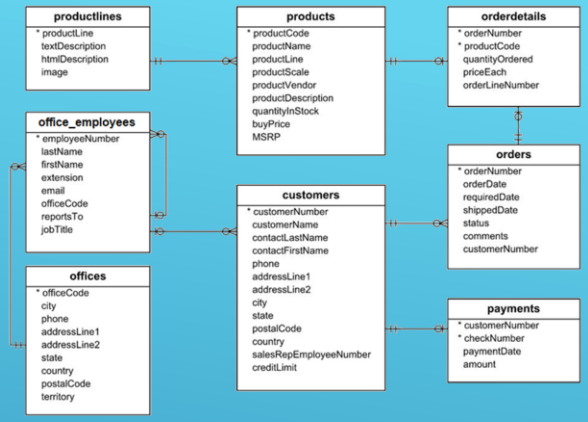


https://www.mysqltutorial.org/mysql-sample-database.aspx

SQL

Data Manipulation Language (DML)

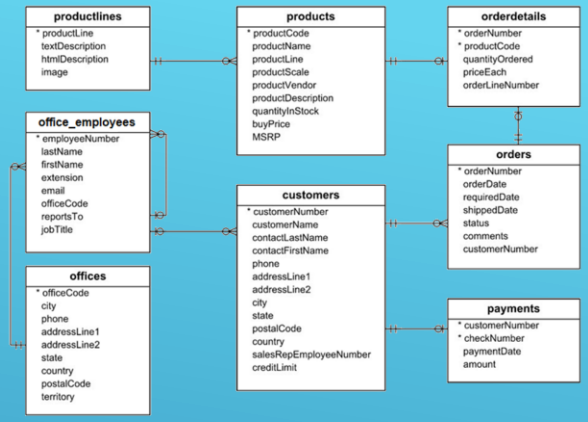
Nested Queries (IN)



Return all customers who made at least one payment greater than 3000$. Schema details can be found here.

```
SELECT customerNumber, customerName FROM customers
 WHERE customerNumber IN (SELECT customerNumber FROM payments WHERE amount>3000)
```

SQL



Data Manipulation Language (DML)

Nested Queries (IN)

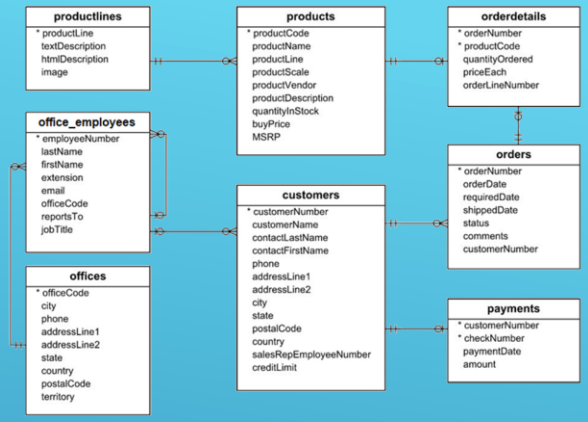
Return all customers who made at least one payment greater than 3000$. Schema details can be found here.

```
SELECT customerNumber, customerName FROM customers
 WHERE customerNumber IN (SELECT customerNumber FROM payments WHERE amount>3000)
```

Are these queries equivalent?

```
SELECT C.customerNumber, C.customerName
        FROM customers C INNER JOIN payments P
                        ON C.customerNumber=P.customerNumber AND amount>3000
```

SQL



Data Manipulation Language (DML)

Nested Queries (IN)

Return all customers who made at least one payment greater than 3000$. Schema details can be found here.

```
SELECT customerNumber, customerName FROM customers
 WHERE customerNumber IN (SELECT customerNumber FROM payments WHERE amount>3000)
```

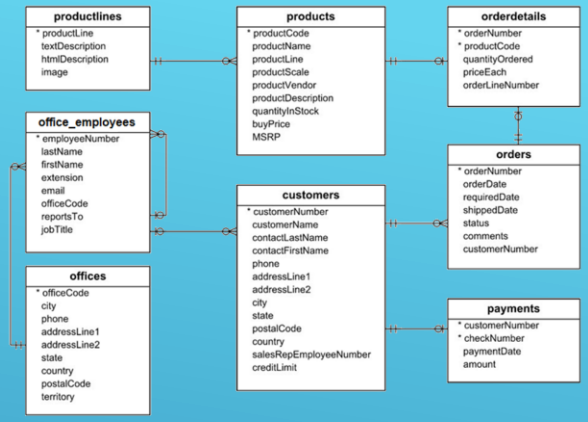
NO ! In this case you may use DISTINCT but does not work in all cases.

```
SELECT C.customerNumber, C.customerName
        FROM customers C INNER JOIN payments P
                        ON C.customerNumber=P.customerNumber AND amount>3000
```

SQL

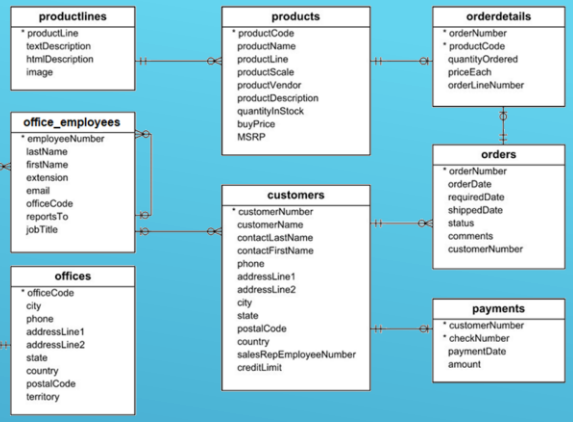
Data Manipulation Language (DML)

Nested Queries (NOT IN)



Return all customers who always made payments greater than 3000$. Schema details can be found here.

SQL



Data Manipulation Language (DML)

Nested Queries (NOT IN)

Return all customers who always made payments greater than 3000$. Schema details can be found here.

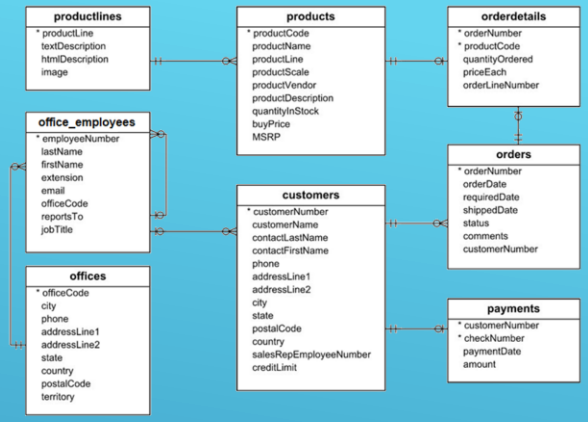
```
SELECT customerNumber, customerName FROM customers
 WHERE customerNumber NOT IN (SELECT customerNumber FROM payments WHERE amount<=3000)
AND customerNumber IN (SELECT customerNumber FROM payments WHERE amount>3000)
```

What can go bad with such queries (NOT IN)?

SQL

Data Manipulation Language (DML)

Nested Queries (NOT IN)



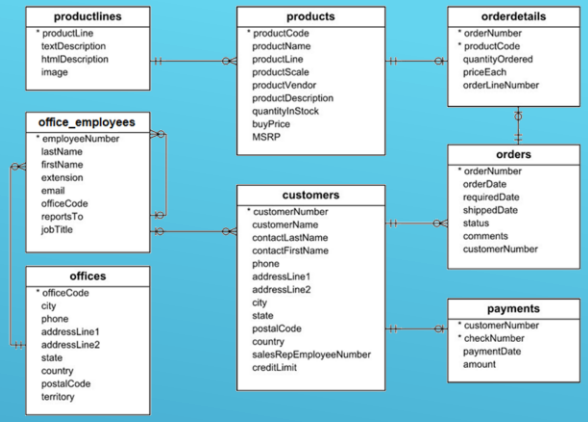
Using below consultants table from here, return first name and last name for all office_employees that are not consultants . Schema details can be found here.

employeeNumber	vendorEmail
1102	gbondur@vendors.com
1337	lbondur@vendors.com
1611	afixter@vendors.com
1625	ykato@vendors.com

SQL

Data Manipulation Language (DML)

Nested Queries (NOT IN)



Using below consultants table from here, return first name and last name for all office_employees that are not consultants . Schema details can be found here.

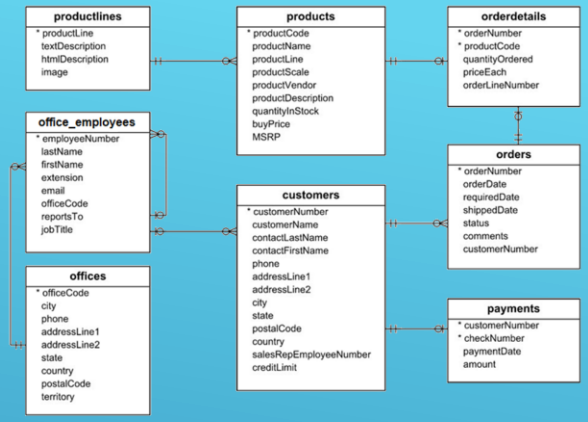
employeeNumber	vendorEmail
1102	gbondur@vendors.com
1337	lbondur@vendors.com
1611	afixter@vendors.com
1625	ykato@vendors.com

```
SELECT firstName, lastName FROM office_employees
WHERE employeeNumber NOT IN (SELECT employeeNumber FROM consultants);
```

SQL

Data Manipulation Language (DML)

Nested Queries (NOT IN)



Using below consultants table from here, return first name and last name for all office_employees that are not consultants . Schema details can be found here.

employeeNumber	vendorEmail
1102	gbondur@vendors.com
1337	lbondur@vendors.com
1611	afixter@vendors.com
1625	ykato@vendors.com
NULL	abt@vendors.com

```
SELECT firstName, lastName FROM office_employees
WHERE employeeNumber NOT IN (SELECT employeeNumber FROM consultants);
```

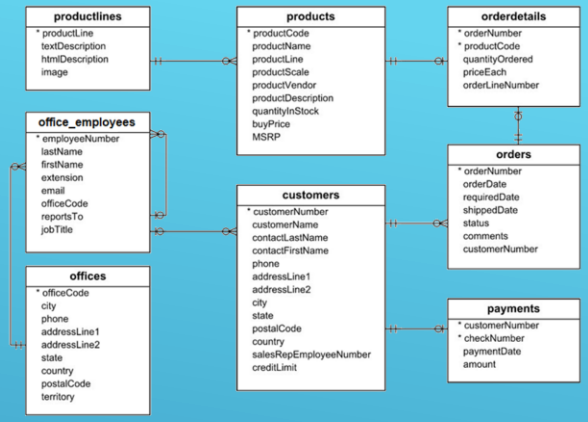
Let us now add a new vendor employee for which we only know his email address:

```
INSERT INTO consultants (employeeNumber, vendorEmail) VALUES (NULL, 'abt@vendors.com');
```

SQL

Data Manipulation Language (DML)

Nested Queries (NOT IN)



Using below consultants table from here, return first name and last name for all office_employees that are not consultants . Schema details can be found here.

employeeNumber	vendorEmail
1102	gbondur@vendors.com
1337	lbondur@vendors.com
1611	afixter@vendors.com
1625	ykato@vendors.com
NULL	abt@vendors.com

```
SELECT firstName, lastName FROM office_employees
WHERE employeeNumber NOT IN (SELECT employeeNumber FROM consultants);
```

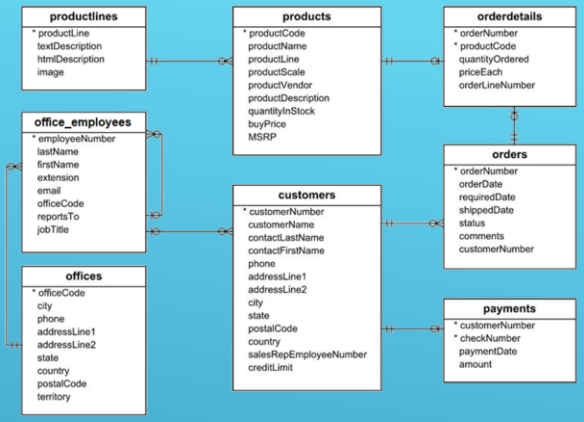
firstName	lastName

Is this the correct result?

SQL

Data Manipulation Language (DML)

Nested Queries (column operator ANY|SOME|ALL)

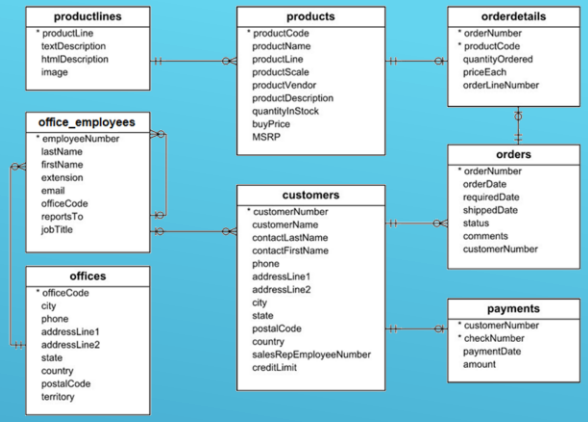


Return customer names for those costumers which has at least one payment greater than 3000.
Schema details can be found here.

SQL

Data Manipulation Language (DML)

Nested Queries (column operator ANY|SOME|ALL)



Return customer names for those costumers which has at least one payment greater than 3000.
Schema details can be found here.

Can be replaces with list of values when need it.

```
SELECT customerName FROM customers C
 WHERE 3000 < ANY (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber);
```

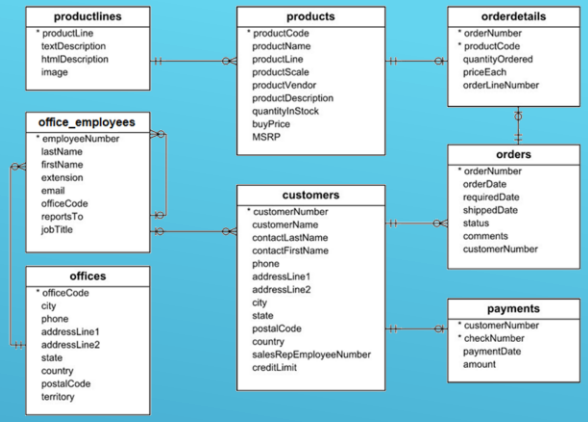
Can be replaces with **SOME**

Can be replaces with any of the following operators: =, !=, >, <, <=, >=

SQL

Data Manipulation Language (DML)

Nested Queries (column operator ANY|SOME|ALL)



Return customer names for those costumers which has all payments greater than 3000. Schema details can be found here.

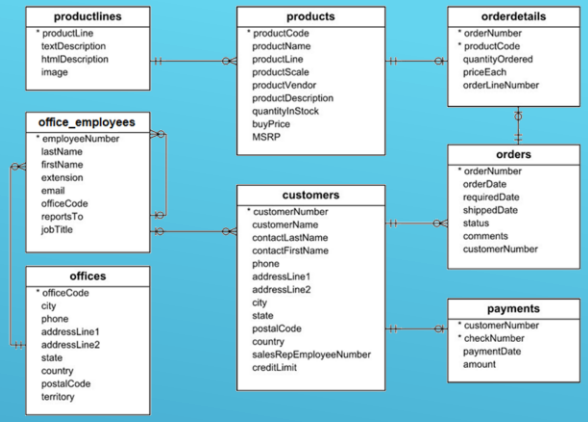
```
SELECT customerName FROM customers C
 WHERE 3000 < ALL (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber);
```

Is this correct?

SQL

Data Manipulation Language (DML)

Nested Queries (column operator ANY|SOME|ALL)



Return customer names for those costumers which has all payments greater than 3000. Schema details can be found here.

Lets check for customerNumber 247.

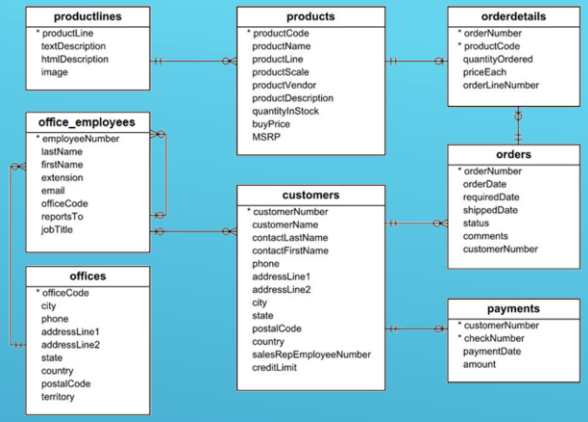
```
SELECT customerName FROM customers C
 WHERE 3000 < ALL (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber)
AND customerNumber=247;
```

customerNumber
247

SQL

Data Manipulation Language (DML)

Nested Queries (column operator ANY|SOME|ALL)



Return customer names for those costumers which has all payments greater than 3000. Schema details can be found here.

```
SELECT customerName FROM customers C
 WHERE 3000 < ALL (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber)
AND customerNumber=247;
```

customerNumber
247

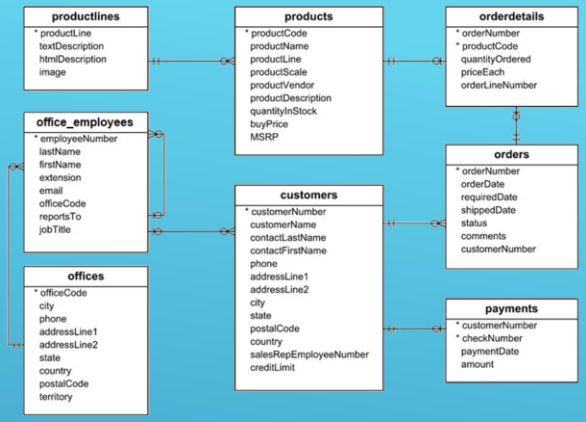
Let us now see the payments for this customer.

```
SELECT amount FROM payments P WHERE P.customerNumber=247
```

amount

Is this what we want?

SQL



Data Manipulation Language (DML)

Nested Queries (column operator ANY|SOME|ALL)

We can rewrite **ALL** using **ANY** expression as follows:

```
SELECT customerName FROM customers C
 WHERE 3000 < ALL (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber);
```

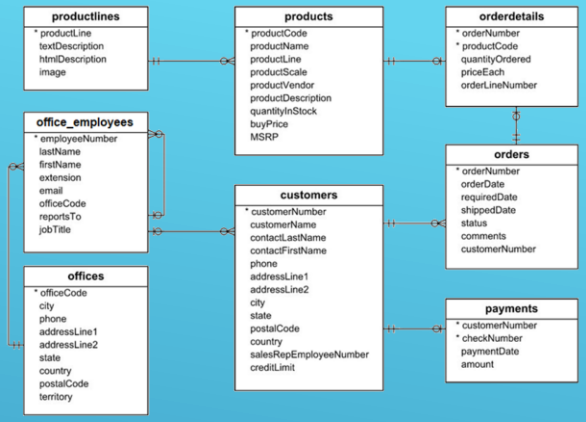
```
SELECT customerName FROM customers C WHERE
NOT (3000 >= ANY (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber));
```

Boolean **NOT** operator.

SQL

Data Manipulation Language (DML)

Nested Queries (EXISTS)

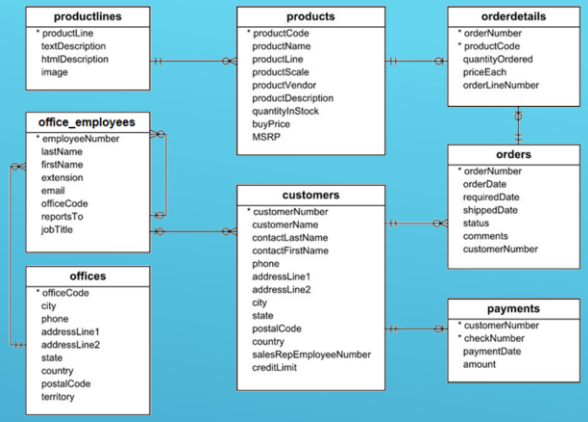


Return customer names for those costumers which has at least one payment greater than 3000 without using **JOIN** or **ANY**. Schema details can be found here.

SQL

Data Manipulation Language (DML)

Nested Queries (EXISTS)



Return customer names for those costumers which has at least one payment greater than 3000.
Schema details can be found here.

```
SELECT customerName FROM customers C WHERE
EXISTS (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber
AND payment > 3000);
```

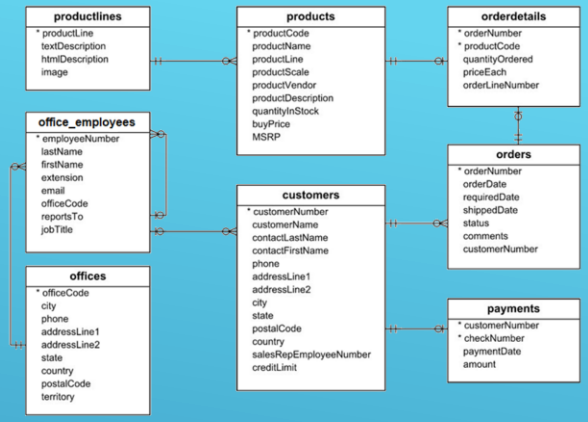
```
SELECT customerName FROM customers C
 WHERE 3000 < ANY (SELECT amount FROM payments P WHERE P.customerNumber=C.customerNumber);
```

```
SELECT DISTINT customerName FROM customers C INNER JOIN payments P
                            ON P.customerNumber=C.customerNumber AND P.payment > 3000;
```

SQL

Data Manipulation Language (DML)

Nested Queries (EXISTS)



Consider consultans table

employeeNumber	vendorEmail
1102	gbondur@vendors.com
1337	lbondur@vendors.com
1611	afixter@vendors.com
1625	ykato@vendors.com
NULL	abt@vendors.com

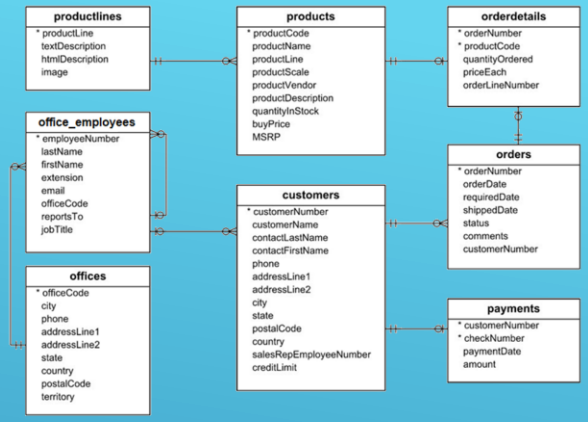
```
CREATE TABLE consultants (employeeNumber INT NULL, vendorEmail VARCHAR(100) NULL);

INSERT INTO consultants (employeeNumber, vendorEmail) VALUES (1102, 'gbondur@vendors.com');
INSERT INTO consultants (employeeNumber, vendorEmail) VALUES (1337, 'lbondur@vendors.com');
INSERT INTO consultants (employeeNumber, vendorEmail) VALUES (1611, 'afixter@vendors.com');
INSERT INTO consultants (employeeNumber, vendorEmail) VALUES (1625, 'ykato@vendors.com');
INSERT INTO consultants (employeeNumber, vendorEmail) VALUES (NULL, 'abt@vendors.com');
```

SQL

Data Manipulation Language (DML)

Nested Queries (EXISTS)



Note that **EXISTS** evaluates to `True` even if the returned column is NULL.

Using consultants table, return 1 if exists and employee with the vendor email 'abt@vendors.com'.

employeeNumber	vendorEmail
1102	gbondur@vendors.com
1337	lbondur@vendors.com
1611	afixter@vendors.com
1625	ykato@vendors.com
NULL	abt@vendors.com

```
SELECT 1 WHERE
EXISTS (SELECT employeeNumber FROM consultants WHERE vendorEmail='abt@vendors.com');
```

```
SELECT 1 FROM DUAL WHERE
EXISTS (SELECT employeeNumber FROM consultants WHERE vendorEmail='abt@vendors.com');
```

```
SELECT 1 FROM consultants WHERE vendorEmail='abt@vendors.com');
```

SQL

Data Manipulation Language (DML)

Nested Queries (implementing DIVISION)

STUDENTS

student_id	name	gender
1	John	M
2	Adam	M
3	Sandra	F

BORROWED

student_id	book_id
1	id100
1	id200
3	id200
1	Id206
3	id200

BOOKS

book_id	author	title
id100	Ullman	DBMS
id200	Linz	Automata
id206	Baader	Term Rew.

```
CREATE TABLE Students(student_id INT, name VARCHAR(20), gender CHAR(1));
CREATE TABLE Borrowed(student_id INT, book_id VARCHAR(20));
CREATE TABLE Books(book_id VARCHAR(20), author VARCHAR(20), title VARCHAR(20));
INSERT INTO Students(student_id, name, gender) VALUES (1, 'John', 'M');
INSERT INTO Students(student_id, name, gender) VALUES (2, 'Adam', 'M');
INSERT INTO Students(student_id, name, gender) VALUES (3, 'Sandra', 'F');
INSERT INTO Books(book_id, author, title) VALUES ('id100', 'Ullman', 'DBMS');
INSERT INTO Books(book_id, author, title) VALUES ('id200', 'Linz', 'Automata');
INSERT INTO Books(book_id, author, title) VALUES ('id206', 'Baader', 'Term Rew.');
INSERT INTO Borrowed(student_id, book_id) VALUES (1, 'id100');
INSERT INTO Borrowed(student_id, book_id) VALUES (1, 'id200');
INSERT INTO Borrowed(student_id, book_id) VALUES (3, 'id200');
INSERT INTO Borrowed(student_id, book_id) VALUES (1, 'id206');
INSERT INTO Borrowed(student_id, book_id) VALUES (3, 'id200');
```

SQL

Data Manipulation Language (DML)

Nested Queries (implementing DIVISION)

STUDENTS

student_id	name	gender
1	John	M
2	Adam	M
3	Sandra	F

BORROWED

student_id	book_id
1	id100
1	id200
3	id200
1	Id206
3	id200

BOOKS

book_id	author	title
id100	Ullman	DBMS
id200	Linz	Automata
id206	Baader	Term Rew.

Find all student names that borrowed all books.

SQL

Data Manipulation Language (DML)

Nested Queries (implementing DIVISION)

STUDENTS

student_id	name	gender
1	John	M
2	Adam	M
3	Sandra	F

BORROWED

student_id	book_id
1	id100
1	id200
3	id200
1	Id206
3	id200

BOOKS

book_id	author	title
id100	Ullman	DBMS
id200	Linz	Automata
id206	Baader	Term Rew.

Find all student names that borrowed all books.

Step 1. Find all possible pairs of student_id and book_id.

SQL

Data Manipulation Language (DML)

Nested Queries (implementing DIVISION)

STUDENTS

student_id	name	gender
1	John	M
2	Adam	M
3	Sandra	F

BORROWED

student_id	book_id
1	id100
1	id200
3	id200
1	Id206
3	id200

BOOKS

book_id	author	title
id100	Ullman	DBMS
id200	Linz	Automata
id206	Baader	Term Rew.

Find all student names that borrowed all books.

Step 1. Find all possible pairs of student_id and book_id.

```
SELECT S.student_id, B.books_id FROM students S, books B;
```

student_id	book_id
1	id100
1	id200
1	id206
2	id100
2	id200
2	id206
3	id100
3	id200
3	id206

SQL

Data Manipulation Language (DML)

Nested Queries (implementing DIVISION)

STUDENTS

student_id	name	gender
1	John	M
2	Adam	M
3	Sandra	F

BORROWED

student_id	book_id
1	id100
1	id200
3	id200
1	Id206
3	id200

BOOKS

book_id	author	title
id100	Ullman	DBMS
id200	Linz	Automata
id206	Baader	Term Rew.

Find all student names that borrowed all books.

Step 2. Find all student ids that **did not** borrowed all the books.


```
SELECT S.student_id, B.books_id FROM students S, books B
EXCEPT
SELECT S.student_id, B.books_id FROM borrowed;
```

student_id	book_id
2	id100
2	id200
2	id206
3	id100
3	id206

```
SELECT A.student_id, A.book_id FROM
        (SELECT S.student_id, B.book_id FROM students S, books B) A
            LEFT JOIN borrowed BO ON A.student_id=BO.student_id AND A.book_id=BO.book_id
WHERE BO.student_id IS NULL AND BO.book_id IS NULL;
```

SQL

Data Manipulation Language (DML)

Nested Queries (implementing DIVISION)

STUDENTS

student_id	name	gender
1	John	M
2	Adam	M
3	Sandra	F

BORROWED

student_id	book_id
1	id100
1	id200
3	id200
1	Id206
3	id200

BOOKS

book_id	author	title
id100	Ullman	DBMS
id200	Linz	Automata
id206	Baader	Term Rew.

Find all student names that borrowed all books.

Step 3. Find all students that **did** borrowed all the books.


```
SELECT name FROM students
WHERE student_id NOT IN
(SELECT student_id FROM
            (SELECT S.student_id, B.books_id FROM students S, books B
             EXCEPT
             SELECT S.student_id, B.books_id FROM borrowed) A
);
```

Name
adam

SQL

Data Manipulation Language (DML)

Nested Queries (implementing DIVISION)

STUDENTS

student_id	name	gender
1	John	M
2	Adam	M
3	Sandra	F

BORROWED

student_id	book_id
1	id100
1	id200
3	id200
1	Id206
3	id200

BOOKS

book_id	author	title
id100	Ullman	DBMS
id200	Linz	Automata
id206	Baader	Term Rew.

Find all student names that borrowed all books.

Step 3. Find all students that **did** borrowed all the books.

```
SELECT name FROM students
WHERE student_id NOT IN
(SELECT A.student_id
   FROM
         (SELECT S.student_id, B.book_id FROM students S, books B) A
          LEFT JOIN borrowed BO ON A.student_id=BO.student_id AND A.book_id=BO.book_id
   WHERE BO.student_id IS NULL AND BO.book_id IS NULL
);
```

Name
adam

SQL

Data Manipulation Language (DML)

CASE expression

```
CASE input_expression
     WHEN when_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END;
```

Evaluates to the first `result_expression` if `input_expression` is equal to the `when_expression` if not `else_result_expression` is returned.

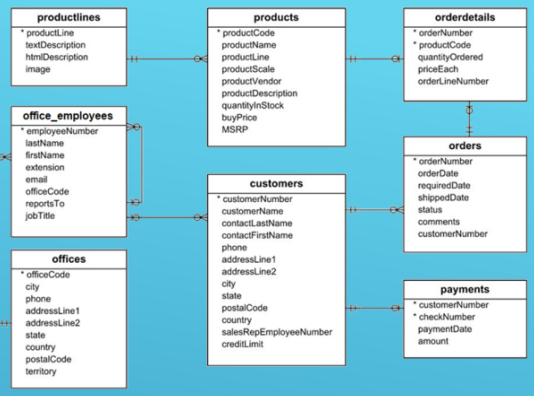
```
CASE
     WHEN boolean_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END;
```

Evaluates to the first `result_expression` where the `boolean_expression` is `True` if not `else_result_expression` is returned.

SQL

Data Manipulation Language (DML)

CASE expression



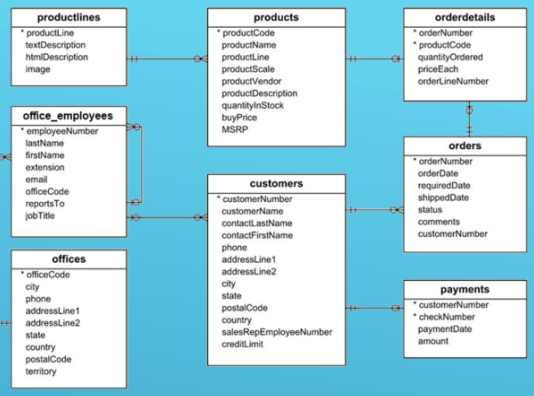
Return name of the product and product line mapped as below for all products with quantity in stock greater then 3000. Schema details can be found here.

Product Line	Name To Be Displayed
Classic Cars	Classic
Motorcycles	Bikes
Planes	Planes
Ships	Ships
Trains	Trains
Trucks and Buses	Large
Vintage Cars	Vintage

SQL

Data Manipulation Language (DML)

CASE expression



Return name of the product and product line mapped as below for all products with quantity in stock greater then 3000. Schema details can be found here.

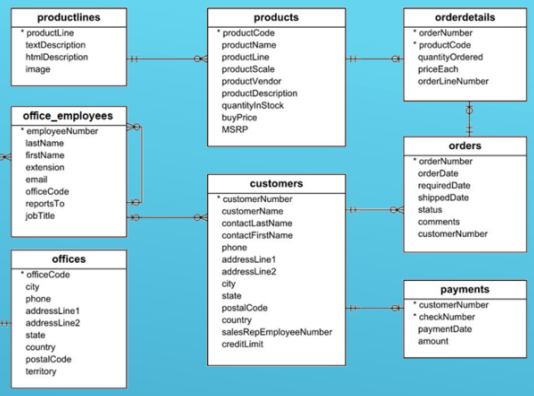
Product Line	Name To Be Displayed
Classic Cars	Classic
Motorcycles	Bikes
Planes	Planes
Ships	Ships
Trains	Trains
Trucks and Buses	Large
Vintage Cars	Vintage

```
SELECT productName,
    CASE productLine
        WHEN 'Classic Cars' THEN 'Classic'
        WHEN 'Motorcycles' THEN 'Bikes'
        WHEN 'Trucks and Buses' THEN 'Large'
        WHEN 'Vintage Cars' THEN 'Vintage'
         ELSE productLine
    END AS prodGroup
FROM products
WHERE quantityInStock>3000;
```

SQL

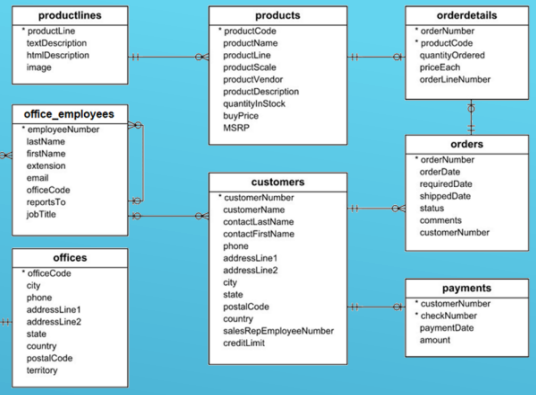
Data Manipulation Language (DML)

CASE expression



Return name of the product and a column that displays either "Large" if the scale is larger or equal to 1:12, "Medium" if scale is strictly smaller than 1:12 and larger or equal to 1:24 and "Small" otherwise. Schema details can be found here.

SQL



Data Manipulation Language (DML)

CASE expression

Return name of the product and a column that displays either "Large" if the scale is larger or equal to 1:12, "Medium" if scale is strictly smaller than 1:12 and larger or equal to 1:24 and "Small" otherwise. Schema details can be found here.

```
SELECT productName,
    CASE
        WHEN CAST(REPLACE(productScale,'1:','') AS INT)<=12 THEN 'LARGE'
        WHEN CAST(REPLACE(productScale,'1:','') AS INT)<=24 THEN 'MEDIUM'
        ELSE 'SMALL'
    END AS prodSize
FROM products;
```


MySQL

```
AS UNASIGNED
```