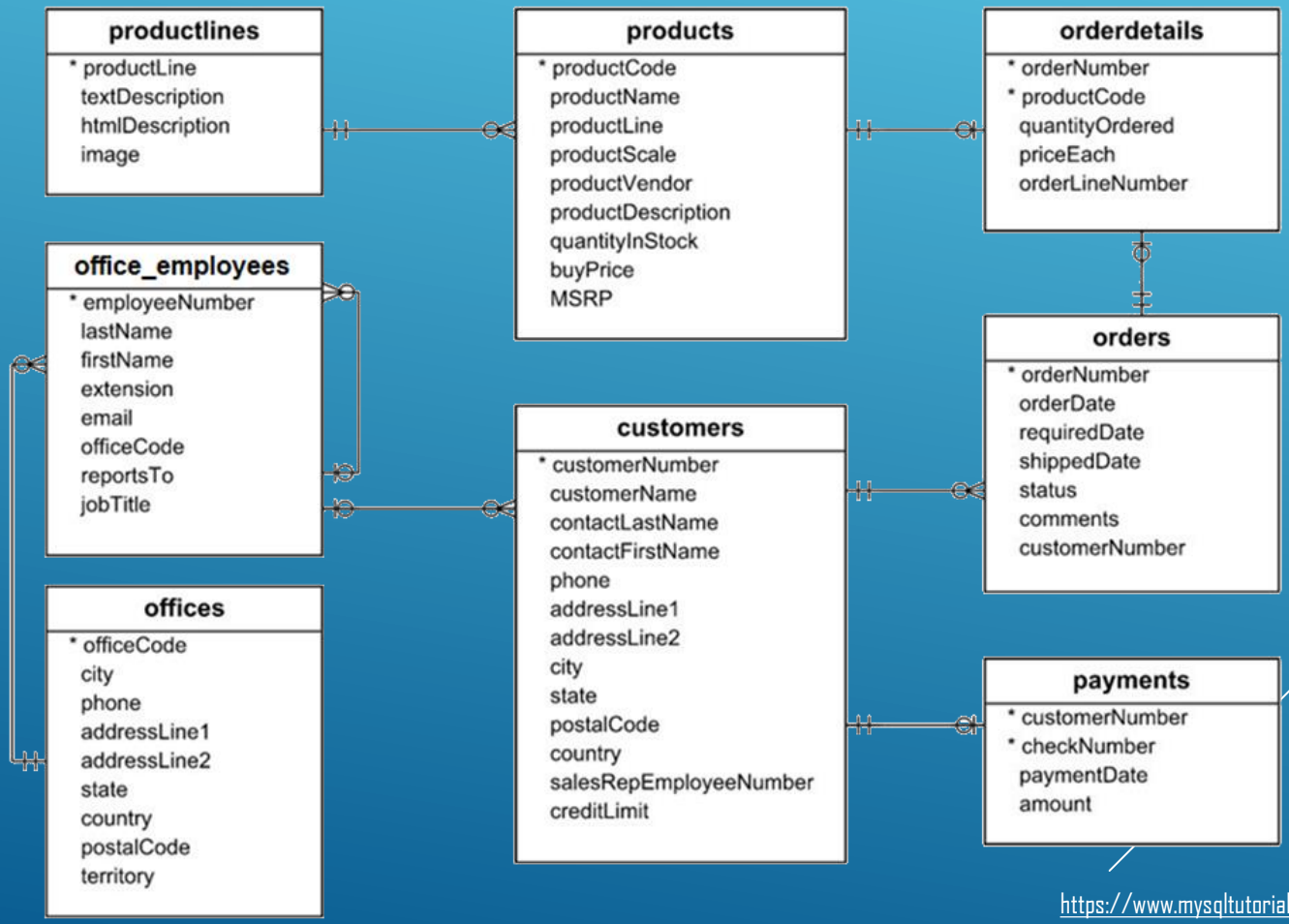
SQL

PART III (JOINS)

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)

Cross Product ($\times$). Table aliasing

Return all pairs of student_is's and book_id's

$\pi_{\text{students.stutudent\_id, books.book\_id}}$ (students $\times$ books).

```
SELECT S.student_id, B.book_id FROM students S, books B
```

Except

```
SELECT S.student_id, B.book_id
 FROM students S CROSS JOIN books B
```

```
SELECT S.student_id, B.book_id
 FROM students S FULL JOIN books B
```

STUDENTS

student_id	name	gender
1	John	M
2	Mike	M
3	Marry	F

BOOKS

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

student_id	book_id
1	id100
2	id100
3	id100
1	id200
2	id200
3	id200
1	Id206
2	id206
3	id206

SQL

Data Manipulation Language (DML)

JOINS ⋈

Creating toy schema and data.

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)

JOINS ⋈

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.

Return the list of student names who borrowed books written by "Linz".

SQL

Data Manipulation Language (DML)

JOINS ⋈

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.

Return the list of student names who borrowed books written by "Linz".

```
SELECT S.Name
  FROM Books B, Borrowed R, Students S
  WHERE S.student_id=R.student_id AND
        R.book_id=B.book_id AND
        B.Author ='Linz'
```

name
John
Sandra
Sandra

SQL

Data Manipulation Language (DML)

JOINS ⋈

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.

Return the list of student names who borrowed books written by "Linz".

```
SELECT DISTINCT S.Name
  FROM Books B, Borrowed R, Students S
  WHERE S.student_id=R.student_id AND
        R.book_id=B.book_id AND
        B.Author ='Linz'
```

name
John
Sandra

SQL

Data Manipulation Language (DML)

JOINS ⋈. INNER JOIN

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.

Return the list of student names who borrowed books written by "Linz".

```
SELECT DISTINCT S.Name
  FROM Books B INNER JOIN Borrowed R ON R.book_id=B.book_id
               INNER JOIN Students S ON S.student_id=R.student_id
  WHERE B.Author ='Linz'
```

name
John
Sandra

SQL

Data Manipulation Language (DML)

JOINS ⋈. INNER JOIN

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.

Return the list of student names who borrowed books written by "Linz".

```
SELECT DISTINCT S.Name
  FROM Books B INNER JOIN Borrowed R USING(book_id)
               INNER JOIN Students S USING(student_id)
  WHERE B.Author ='Linz'
```

ORACLE

MySQL

In case the joining column names is the same (Natural Join)

name
John
Sandra

SQL

Data Manipulation Language (DML)

JOINS ⋈

STUDENTS

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

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

Return the list of student names who borrowed books written by "Linz".

```
SELECT DISTINCT S.Name
  FROM Books B INNER JOIN Borrowed R ON R.book_id=B.book_id
               INNER JOIN Students S ON S.student_id=R.student_id
  WHERE B.Author ='Linz'
```

SQL

Data Manipulation Language (DML)

JOINS ⋈

student_id	name	gender	student_id	book_id	book_id	author	title
1	John	M	1	id100	id100	Ullman	DBMS
1	John	M	1	id200	id200	Linz	Automata
3	Sandra	F	3	id200	id200	Linz	Automata
1	John	M	1	id206	id206	Baader	Term Rew.
3	Sandra	F	3	id200	id200	Linz	Automata

Return the list of student names who borrowed books written by "Linz".

```
SELECT DISTINCT S.Name
  FROM Books B INNER JOIN Borrowed R ON R.book_id=B.book_id
               INNER JOIN Students S ON S.student_id=R.student_id
  WHERE B.Author ='Linz'
```

SQL

Data Manipulation Language (DML)

JOINS ⋈

student_id	name	gender	student_id	book_id	book_id	author	title
1	John	M	1	id200	id200	Linz	Automata
3	Sandra	F	3	id200	id200	Linz	Automata
3	Sandra	F	3	id200	id200	Linz	Automata

Return the list of student names who borrowed books written by "Linz".

```
SELECT DISTINCT S.Name
  FROM Books B INNER JOIN Borrowed R ON R.book_id=B.book_id
               INNER JOIN Students S ON S.student_id=R.student_id
  WHERE B.Author ='Linz'
```

SQL

Data Manipulation Language (DML)

JOINS ⋈

name
John
Sandra

Return the list of student names who borrowed books written by "Linz".

```
SELECT DISTINCT S.Name
  FROM Books B INNER JOIN Borrowed R ON R.book_id=B.book_id
               INNER JOIN Students S ON S.student_id=R.student_id
  WHERE B.Author ='Linz'
```

SQL

Data Manipulation Language (DML)

JOINS ⋈. Join Types

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5

MySQL

```
INSERT INTO A(`key`, value) VALUES (1,'A1');
INSERT INTO A(`key`, value) VALUES (2,'A2');
INSERT INTO A(`key`, value) VALUES (3,'A3');
INSERT INTO B(`key`, value) VALUES (1,'B1');
INSERT INTO B(`key`, value) VALUES (4,'B4');
INSERT INTO B(`key`, value) VALUES (5,'B5');
```

MySQL

```
CREATE TABLE A(`key` INT, value VARCHAR(10));
CREATE TABLE B(`key` INT, value VARCHAR(10));
```

SAP SYBASE, SQL Server

```
CREATE TABLE A([key] INT, value VARCHAR(10));
CREATE TABLE B([key] INT, value VARCHAR(10));
```

PostgreSQL, ORACLE, MySQL

```
CREATE TABLE A("key" INT, value VARCHAR(10));
CREATE TABLE B("key" INT, value VARCHAR(10));
```

MySQL

` if ANSI mode is of " otherwise

```
SET GLOBAL sql_mode = 'ANSI';
SET SESSION sql_mode = 'ANSI';

SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

SQL

Data Manipulation Language (DML)

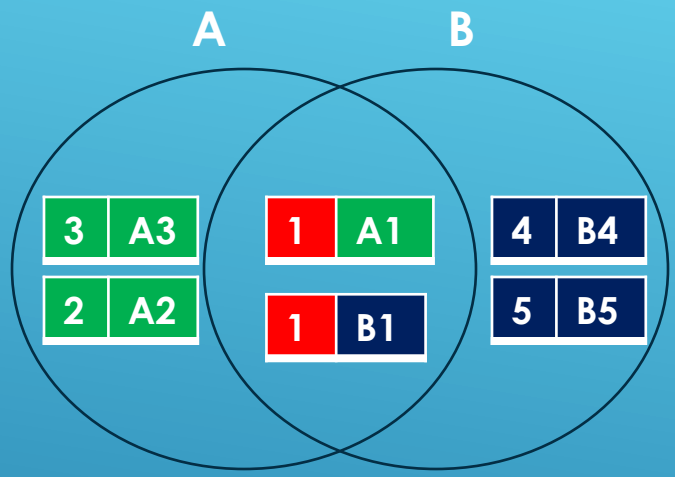
JOINS ⋈. Join Types

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5



SQL

For keywords escaping

Data Manipulation Language (DML)

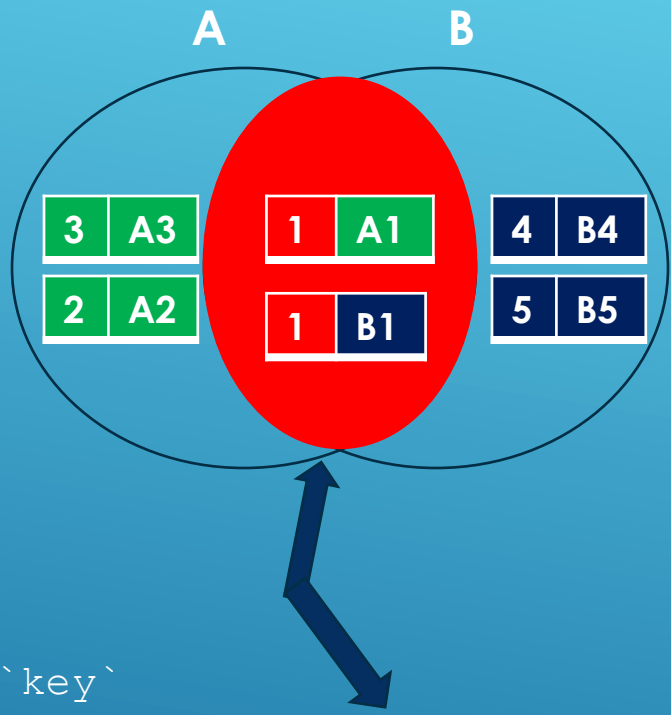
JOINS ⋈. Inner Join

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5



Version 1:

```
SELECT A.value, B.value
    FROM A INNER JOIN B ON A.`key`=B.`key`
```

Version 2:

```
SELECT A.value, B.value
    FROM A, B
    WHERE A.`key`=B.`key`
```

SQL

For keywords escaping

Data Manipulation Language (DML)

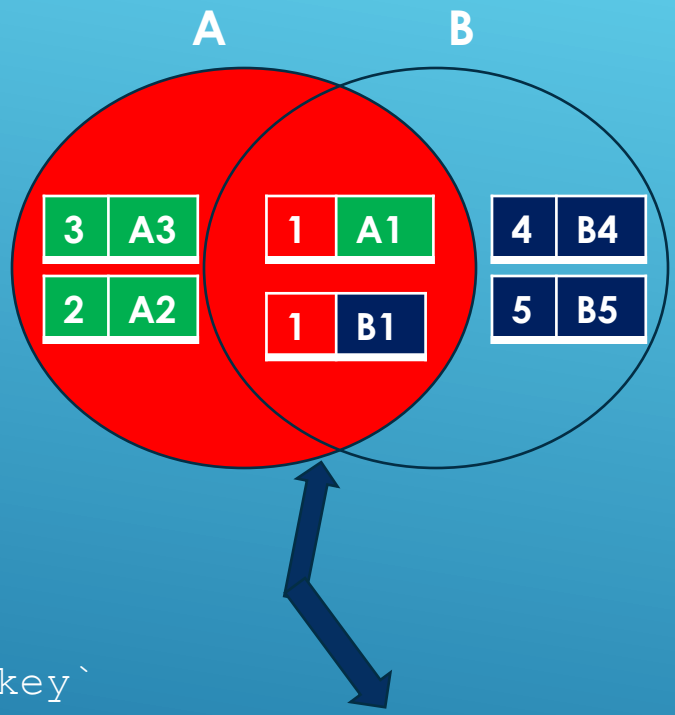
JOINS ⋈. Left Join

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5



Version 1:

```
SELECT A.value, B.value
    FROM A LEFT JOIN B ON A.`key`=B.`key`
```

Version 2:

```
SELECT A.value, B.value
    FROM A LEFT OUTER JOIN B ON A.`key`=B.`key`
```

A.value	B.Value
A1	B1
A2	NULL
A3	NULL

SQL

Data Manipulation Language (DML)

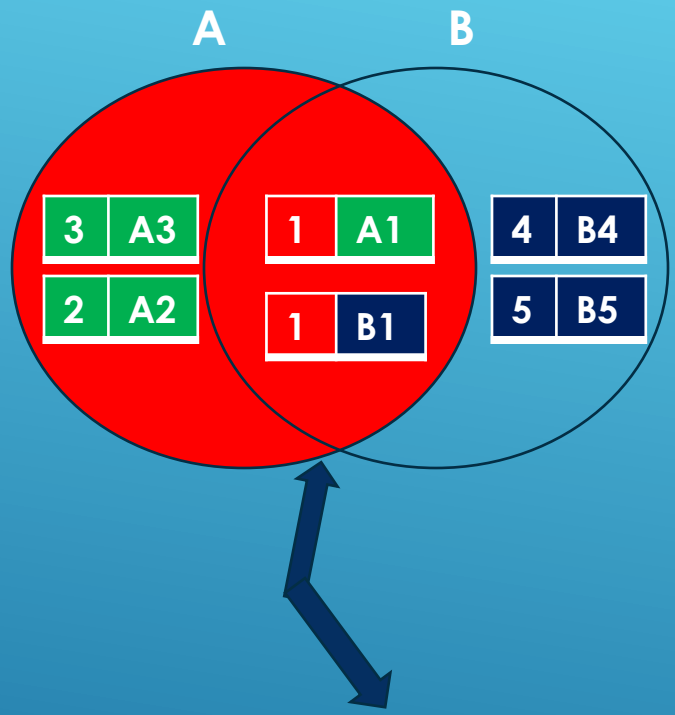
JOINS ⋈. Left Join

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5



Version 1*:

```
SELECT A.value, B.value
    FROM A, B  WHERE A.[key]*=B.[key]
```

Version 2*:

```
SELECT A.value, B.value
    FROM A, B  WHERE A."key"=B."key" (+)
```

A.value	B.Value
A1	B1
A2	NULL
A3	NULL

SQL

For keywords escaping

Data Manipulation Language (DML)

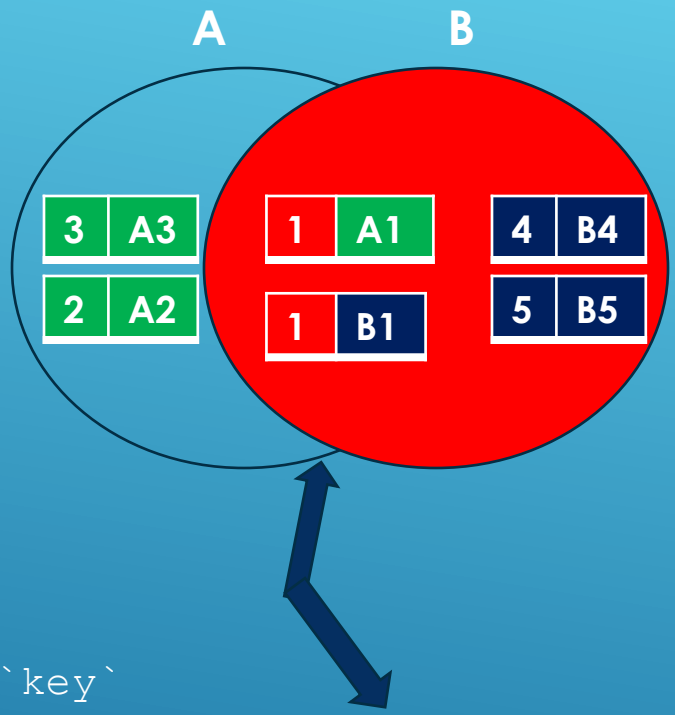
JOINS ⋈. Right Join

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5



Version 1:

```
SELECT A.value, B.value
    FROM A RIGHT JOIN B ON A.`key`=B.`key`
```

Version 2:

```
SELECT A.value, B.value
    FROM A RIGHT OUTER JOIN B ON A.`key`=B.`key`
```

A.value	B.Value
A1	B1
NULL	B4
NULL	B5

SQL

Data Manipulation Language (DML)

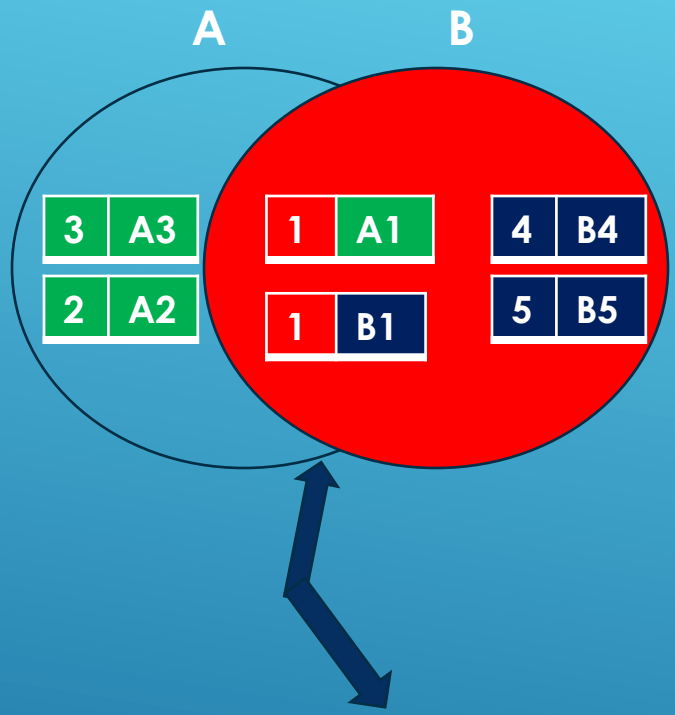
JOINS ⋈. Right Join

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5



A.value	B.Value
A1	B1
NULL	B4
NULL	B5

Version 1*:

```
SELECT A.value, B.value
    FROM A, B  WHERE A.[key]=*B.[key]
```

Version 2*:

```
SELECT A.value, B.value
    FROM A, B  WHERE A."key" (+)=B."key"
```

SQL

For keywords escaping

Data Manipulation Language (DML)

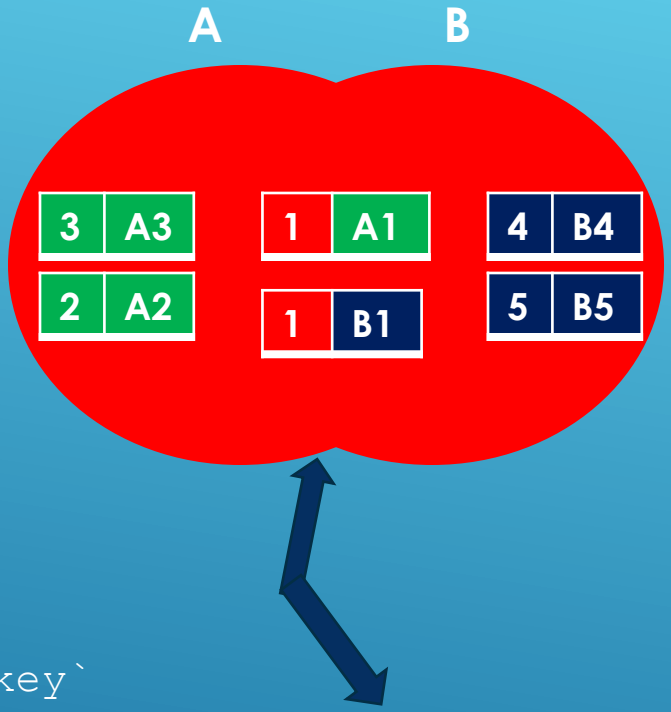
JOINS ⋈. Full Join

A

key	value
1	A1
2	A2
3	A3

B

key	value
1	B1
4	B4
5	B5



Version 1:

```
SELECT A.value, B.value
    FROM A FULL JOIN B ON A.`key`=B.`key`
```

Version 2: Except

```
SELECT A.value, B.value
    FROM A FULL OUTER JOIN B ON A.`key`=B.`key`
```

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL
A3	NULL

SQL

Data Manipulation Language (DML)

JOINS ⋈. Full Join

How to simulate full joins?

A

key	value
1	A1
2	A2
3	**A2**

B

key	value
1	B1
4	B4
5	B5

Expected result

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL
A2	NULL

SQL

Data Manipulation Language (DML)

JOINS ⋈. Full Join

How to simulate full joins?

A

key	value
1	A1
2	A2
3	**A2**

B

key	value
1	B1
4	B4
5	B5

Solution 1:

```
SELECT A.value, B.value
    FROM A LEFT JOIN B ON A.`key`=B.`key`
UNION
SELECT A.value, B.value
    FROM A RIGHT JOIN B ON A.`key`=B.`key`;
```

Expected result

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL
A2	NULL

=

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL

SQL

Data Manipulation Language (DML)

JOINS ⋈. Full Join

How to simulate full joins?

A

key	value
1	A1
2	A2
3	**A2**

B

key	value
1	B1
4	B4
5	B5

Solution 2:

```
SELECT A.value, B.value
    FROM A LEFT JOIN B ON A.`key`=B.`key`
UNION ALL
SELECT A.value, B.value
    FROM A RIGHT JOIN B ON A.`key`=B.`key`;
```

Expected result

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL
A2	NULL

⋈

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL
A2	NULL
A1	B1

SQL

Data Manipulation Language (DML)

JOINS ⋈. Full Join

How to simulate full joins?

A

key	value
1	A1
2	A2
3	**A2**

B

key	value
1	B1
4	B4
5	B5

Solution 3:

```
SELECT A.value, B.value
    FROM A LEFT JOIN B ON A.`key`=B.`key`
UNION ALL
SELECT A.value, B.value
    FROM A RIGHT JOIN B ON A.`key`=B.`key`
    WHERE A.`key` IS NULL;
```

Expected result

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL
A2	NULL

=

A.value	B.Value
A1	B1
NULL	B4
NULL	B5
A2	NULL
A2	NULL

SQL

Data Manipulation Language (DML)

JOINS ⋈. Counting joins

T1	
A	B

T2	
B	C

T3	
C	D

|T1| - number of records in T1;
|T2| - number of records in T2;
|T3| - number of records in T3;

Fill the question marks with the correct numbers or records.

```
SELECT * FROM T1 CROSS JOIN T2;
```

Cross Product	
Min	Max
?	?

```
SELECT * FROM T1 INNER JOIN T2 ON T1.B=T2.B;
```

Inner Join	
Min	Max
?	?

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.B=T2.B;
```

Left Join	
Min	Max
?	?

```
SELECT * FROM T1 FULL JOIN T2 ON T1.B=T2.B;
```

Full Join	
Min	Max
?	?

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.B=T2.B
                 RIGHT JOIN T3 ON T2.C=T3.C;
```

Multiple Outer Joins	
Min	Max
?	?

SQL

Data Manipulation Language (DML)

JOINS ⋈. Counting joins

|T1| - number of records in T1;
|T2| - number of records in T2;
|T3| - number of records in T3;

Fill the question marks with the correct numbers or records.

```
SELECT * FROM T1 CROSS JOIN T2;
```

Cross Product	
Min	Max
|T1|*|T2|	|T1|*|T2|

Number of records returned by the cross product is always the multiplication between the sizes of the two tables.

SQL

Data Manipulation Language (DML)

JOINS ⋈. Counting joins

|T1| - number of records in T1;
|T2| - number of records in T2;
|T3| - number of records in T3;

Fill the question marks with the correct numbers or records.

```
SELECT * FROM T1 INNER JOIN T2 ON T1.B=T2.B;
```

Inner Join	
Min	Max
0	|T1|*|T2|

Minimum records are obtained in case no combination of records make the join condition True. Maximum number of records can be obtained in case the join condition is True for any pairs of records. For example join on Gender where all students are Females.

SQL

Data Manipulation Language (DML)

JOINS ⋈. Counting joins

|T1| - number of records in T1;
|T2| - number of records in T2;
|T3| - number of records in T3;

Fill the question marks with the correct numbers or records.

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.B=T2.B;
```

Left Join	
Min	Max
|T1|	|T1|*|T2|

Minimum records are obtained in case no combination of records make the join condition True.
In this case only the records from T1 are returned.
Maximum number of records can be obtained in case the join condition is True for any pairs of records (we assume that T2 has at least one record, if T2 has no records then maximum is |T1|).

SQL

Data Manipulation Language (DML)

JOINS ⋈. Counting joins

|T1| - number of records in T1;
|T2| - number of records in T2;
|T3| - number of records in T3;

Fill the question marks with the correct numbers or records.

Minimum records are obtained in case the following holds:

1. `SELECT DISTINCT B FROM T1` = `SELECT B FROM T1`
2. `SELECT DISTINCT B FROM T2` = `SELECT B FROM T2`
3. `SELECT B FROM T1` ⊆ `SELECT B FROM T2` OR `SELECT B FROM T1` ⊆ `SELECT B FROM T2`

For the maximum number of records returned we have to keep in mind that there are cases when |T1|+|T2| > |T1|*|T2|, for example 1 and 3. |T1|+|T2| is obtained where the join condition is False for all records and |T1|*|T2| when the join is True for all records.

```
SELECT * FROM T1 FULL JOIN T2 ON T1.B=T2.B;
```

Full Join	
Min	Max
Max(|T1|,|T2|)	Max(|T1|+|T2|, |T1|*|T2|)

SQL

Data Manipulation Language (DML)

JOINS ⋈. Counting joins

|T1| - number of records in T1;
|T2| - number of records in T2;
|T3| - number of records in T3;

Fill the question marks with the correct numbers or records.

Minimum records are obtained when all join conditions are false. Maximum is obtained when all join conditions are true for all records (same assumption that each table has at least 1 row).

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.B=T2.B
                 RIGHT JOIN T3 ON T2.C=T3.C;
```

Multiple Outer Joins	
Min	Max
|T3|	|T1|*|T2|*|T3|

SQL

Data Manipulation Language (DML)

JOINS ⋈. Counting joins

T1	
A	B

T2	
B	C

T3	
C	D

|T1| - number of records in T1;
|T2| - number of records in T2;
|T3| - number of records in T3;

Fill the question marks with the correct numbers or records.

```
SELECT * FROM T1 CROSS JOIN T2;

SELECT * FROM T1 INNER JOIN T2 ON T1.B=T2.B;

SELECT * FROM T1 LEFT JOIN T2 ON T1.B=T2.B;

SELECT * FROM T1 FULL JOIN T2 ON T1.B=T2.B;

SELECT * FROM T1 LEFT JOIN T2 ON T1.B=T2.B
              RIGHT JOIN T3 ON T2.C=T3.C;
```

Cross Product	
Min	Max
|T1|*|T2|	|T1|*|T2|

Inner Join	
Min	Max
0	|T1|*|T2|

Left Join	
Min	Max
|T1|	|T1|*|T2|

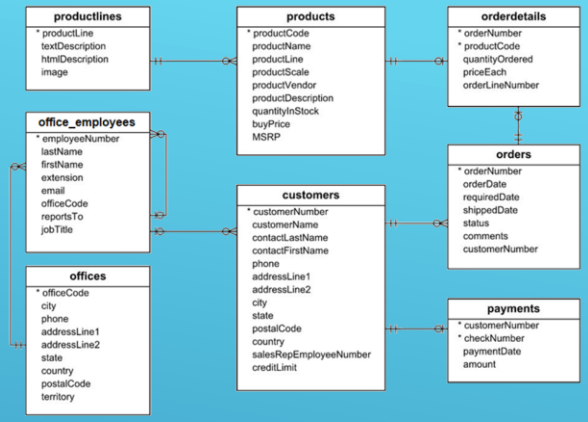
Full Join	
Min	Max
Max(|T1|,|T2|)	Max(|T1|+|T2|, |T1|*|T2|)

Multiple Outer Joins	
Min	Max
|T3|	|T1|*|T2|*|T3|

SQL

Data Manipulation Language (DML)

JOINS ⋈. Mimic Difference

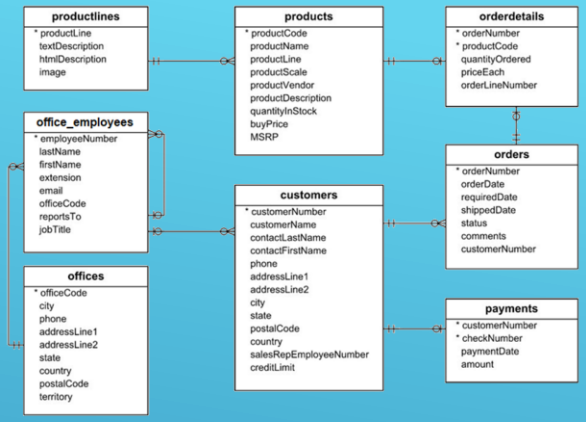


Return the codes for those "Motorcycles" (productLine) products that were never ordered in a quantity greater than 50. Schema details can be found here.


```
SELECT productCode FROM products WHERE productLine='Motorcycles'
EXCEPT
SELECT productCode FROM orderDetails WHERE quantityOrdered>50;
```

How about  ?

SQL



Data Manipulation Language (DML)

JOINS ⋈. Mimic Difference

Problem started here.

Return the codes for those "Motorcycles" (productLine) products that were never ordered in a quantity greater than 50. Schema details can be found here.


```
SELECT productCode FROM products WHERE productLine='Motorcycles'
EXCEPT
SELECT productCode FROM orderDetails WHERE quantityOrdered>50;
```

How about 

?

```
SELECT P.productCode FROM products P LEFT JOIN orderDetails OD
                ON P.productCode=OD.productCode AND quantityOrdered>50
WHERE P.productLine='Motorcycles' AND OD.orderNumber IS NULL;
```

As we will see this can be done also using **NOT IN** or **NOT EXISTS** expressions.

SQL

Data Manipulation Language (DML)

JOINS ⋈. Writing joins

```
SELECT *
FROM   accounts A,
       orders O,
       deals D,
       dealDetails DD,
       ordersDemand OD,
       dealClients DC,
       clients C
WHERE O.accountId=A.accountId AND O.dealId=D.dealId AND
O.dealId=DD.dealId AND OD.orderId=O.orderId AND
DC.dealId=D.dealId AND C.clientId=DC.clientId AND OD.qty>50 AND
O.status='In Progress' AND C.name='John';
```

SQL

Data Manipulation Language (DML)

JOINS ⋈. Writing joins

```
SELECT *
FROM  accounts A INNER JOIN orders O ON O.accountId=A.accountId
                 INNER JOIN deals D  ON O.dealId=D.dealId
                 INNER JOIN dealDetails DD ON O.dealId=DD.dealId
                 INNER JOIN ordersDemand OD ON OD.orderId=O.orderId
                 INNER JOIN dealClients DC ON DC.dealId=D.dealId
                 INNER JOIN clients C ON C.clientId=DC.clientId
WHERE OD.qty>50 AND O.status='In Progress' AND C.name='John';
```

SQL

Data Manipulation Language (DML)

JOINS ⋈. Writing joins

Using below schema return all triples (clientName, accountId, amount) for all clients which has in their account less than 1000$.

clients	
clientName	accountId
Rob	1
Adam	2
John	3

accounts	
accountId	amount
1	10000
2	500

Schema/data definition.

```
CREATE TABLE clients (clientName VARCHAR(20), accountId INT);
CREATE TABLE accounts (accountId INT, amount INT);

INSERT INTO clients (clientName, accountId) VALUES ('Rob', 1);
INSERT INTO clients (clientName, accountId) VALUES ('Adam', 2);
INSERT INTO clients (clientName, accountId) VALUES ('John', 3);

INSERT INTO accounts (accountId, amount) VALUES (1, 10000);
INSERT INTO accounts (accountId, amount) VALUES (2, 500);
```

SQL

Data Manipulation Language (DML)

JOINS ⋈. Writing joins

Using below schema return all triples (clientName, accountId, amount) for all clients which has in their account less than 1000$.

clients	
clientName	accountId
Rob	1
Adam	2
John	3

accounts	
accountId	amount
1	10000
2	500

```
SELECT C.*, ISNULL(A.amount,0)
FROM  clients C, accounts A
WHERE C.accountId *= A.accountId AND ISNULL(A.amount,0)<1000;
```

clientName	accountId	amount
Rob	1	0
Adam	2	500
John	3	0

Is it what expected?

SQL

Data Manipulation Language (DML)

JOINS ⋈. Writing joins

Using below schema return all triples (clientName, accountId, amount) for all clients which has in their account less than 1000$.

clients	
clientName	accountId
Rob	1
Adam	2
John	3

accounts	
accountId	amount
1	10000
2	500

```
SELECT C.*, ISNULL(A.amount,0)
FROM  clients C, accounts A
WHERE C.accountId *= A.accountId AND ISNULL(A.amount,0)<1000;
```

clientName	accountId	amount
Rob	1	0
Adam	2	500
John	3	0

Why is Rob returned?

SQL

Data Manipulation Language (DML)

JOINS ⋈. Writing joins

Using below schema return all triples (clientName, accountId, amount) for all clients which has in their account less than 1000$.

clients	
clientName	accountId
Rob	1
Adam	2
John	3

accounts	
accountId	amount
1	10000
2	500

```
SELECT C.*, ISNULL(A.amount,0)
FROM  clients C, accounts A
WHERE C.accountId *= A.accountId AND ISNULL(A.amount,0)<1000;
```

```
SELECT C.*, ISNULL(A.amount,0)
FROM  clients C INNER JOIN accounts A
ON C.accountId = A.accountId AND ISNULL(A.amount,0)<1000;
```

clientName	accountId	amount
Rob	1	0
Adam	2	500
John	3	0

SQL

Data Manipulation Language (DML)

JOINS ⋈. Writing joins

Using below schema return all triples (clientName, accountId, amount) for all clients which has in their account less than 1000$.

clients	
clientName	accountId
Rob	1
Adam	2
John	3

accounts	
accountId	amount
1	10000
2	500

Correct version.

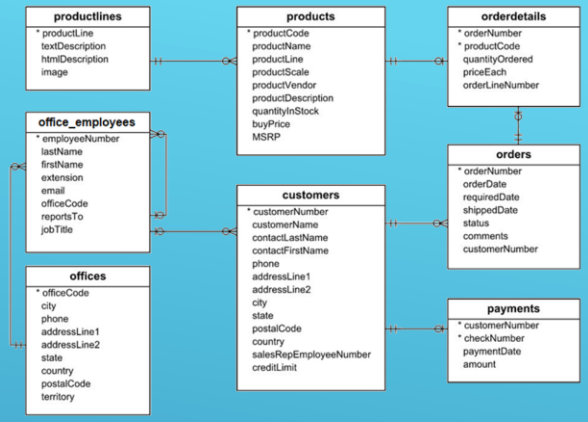
```
SELECT C.*, ISNULL(A.amount,0)
FROM clients C LEFT JOIN accounts A
                    ON C.accountId = A.accountId
     WHERE ISNULL(A.amount,0)<1000;
```

clientName	accountId	amount
Adam	2	500
John	3	0

SQL

Data Manipulation Language (DML)

Updating Joins



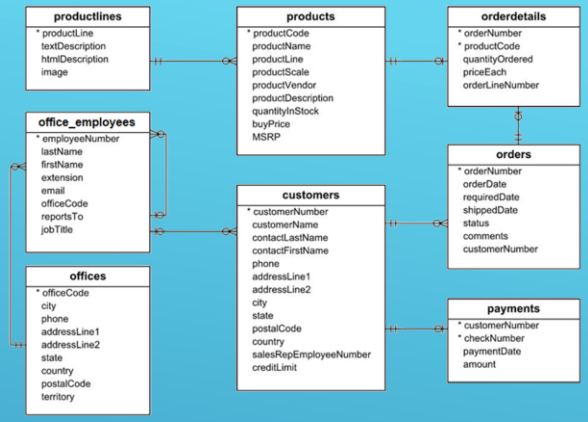
Using consultants table below update the email address in the office_employees table with the vendorEmail from the consultants table based on the employee number. Schema details can be found here.

employeeNumber	vendorEmail
1102	gbondur@vendors.com
1337	lbondur@vendors.com
1611	afixter@vendors.com
1625	ykato@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');
```

SQL



Data Manipulation Language (DML)

Updating Joins

Using consultants table below update the email address in the office_employees table with the vendorEmail from the consultants table based on the employee number. Schema details can be found here.

MySQL

```
UPDATE office_employees E INNER JOIN consultants C ON E.employeeNumber=C.employeeNumber
SET E.email = C.vendorEmail;
```

PostgreSQL

```
UPDATE office_employees E SET E.email = C.vendorEmail
    FROM consultants C WHERE E.employeeNumber=C.employeeNumber;
```

SAP Sybase / Microsoft SQL Server

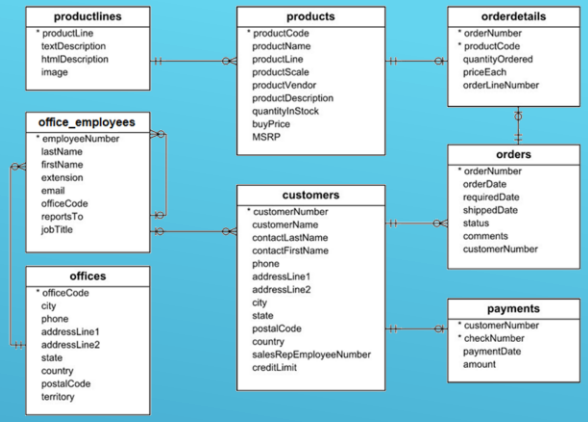
```
UPDATE office_employees SET email = C.vendorEmail
    FROM office_employees E INNER JOIN consultants C ON E.employeeNumber=C.employeeNumber;
```

How about ORACLE IBM DB2 ?

SQL

Data Manipulation Language (DML)

Merge Statement



Using consultants table below update the email address in the office_employees table with the vendorEmail from the consultants table based on the employee number. Schema details can be found here.

How about ORACLE IBM DB2 ? Could be done with bit more complex nested queries.

```
MERGE
    INTO target_table
    USING table_source
    ON (merge_search_condition)
    [ WHEN MATCHED
        THEN (UPDATE SET set_clause | DELETE)]
    [ WHEN NOT MATCHED
        THEN INSERT (column_list) VALUES (values_list) ]
```

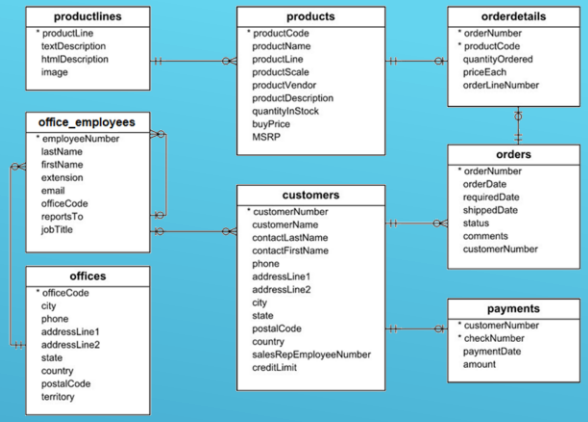
SQL Server ORACLE IBM DB2 PostgreSQL

Merge does INSERT/UPDATE/DELETE of rows in a single statement. When `merge_search_condition` is true, than that row can be either updated or deleted (**WHEN MATCHED**) if false, then we can insert a new row (**WHEN NOT MATCHED**)

SQL

Data Manipulation Language (DML)

Merge Statement



Using consultants table below update the email address in the office_employees table with the vendorEmail from the consultants table based on the employee number. Schema details can be found here.

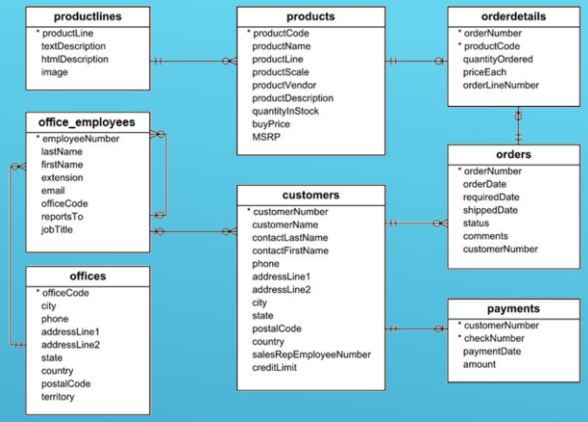
Solution.

```
MERGE
    INTO office_employees E
    USING consultants C
    ON (E.employeeNumber=C.employeeNumber)
    WHEN MATCHED
        THEN UPDATE SET E.email = C.vendorEmail
```

SQL

Data Manipulation Language (DML)

Merge Statement



Using consultants table below update the email address in the office_employees table with the vendorEmail from the consultants table based on the employee number. Schema details can be found here.

Not so orthodox solutions but helps knowing the syntax in case you want to insert data that may violate the key constraints. Note that the constant values in the below select statements do not play any role because we know that all data is duplicated. This is not always the case.

MySQL

```
INSERT INTO office_employees (employeeNumber, email, firstName,lastName,extension,officeCode, jobTitle)
SELECT employeeNumber, vendorEmail, 'x', 'x', 'x', 'x', 'x' FROM consultants
ON DUPLICATE KEY UPDATE email=consultants.vendorEmail;
```

PostgreSQL

```
INSERT INTO office_employees (employeeNumber, email, firstName,lastName,extension,officeCode, jobTitle)
SELECT employeeNumber, vendorEmail, 'x', 'x', 'x', 'x', 'x'  FROM consultants
ON CONFLICT (employeeNumber) DO SET email=consultants.vendorEmail;
```