SQL Queries –required part

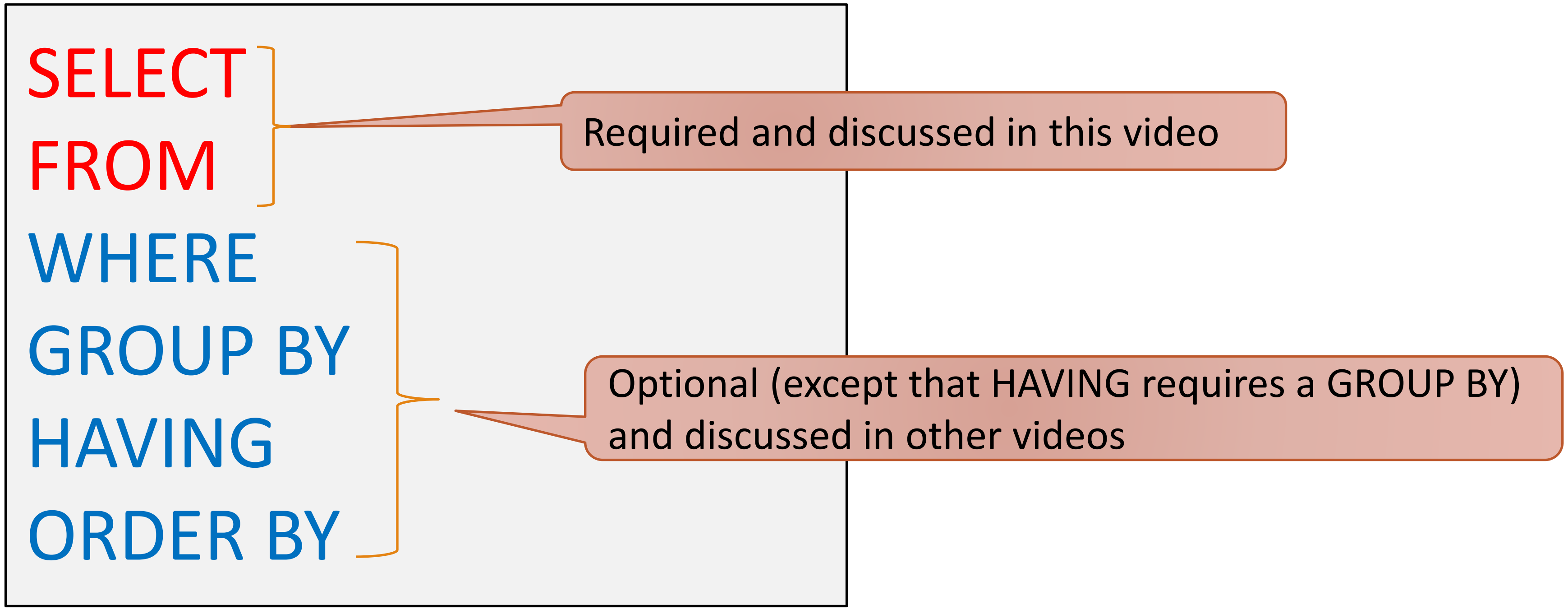
NOTE: BOTH THIS VIDEO AND THE ONE ON THE OPTIONAL PART SHOWS THINGS THAT ARE REQUIRED KNOWLEDGE FOR THE COURSE!

Overview of this video

A run through of the required part of SQL queries

SQL Queries

Queries in SQL have the following form:



SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

Required and discussed in this video

Optional (except that HAVING requires a GROUP BY) and discussed in other videos

Most basic query

SELECT is basically output

* means everything

```
SELECT *
FROM Employees;
```

FROM (like in DELETE) is set input table

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

SELECT

SELECT defines what is outputted, making you able to do four kinds of modifications of it:

1. Projection (π)
2. DISTINCT
3. Renaming (ρ)
4. Creating new columns

“Weird” symbols will be explained in SQL misc. video

Projection (π)

Projection allows you to select attributes you want to keep (the rest are discarded)

You use it by writing a list of attributes

Attribute order matters

```
SELECT family_name, birthday
FROM Employees;
```

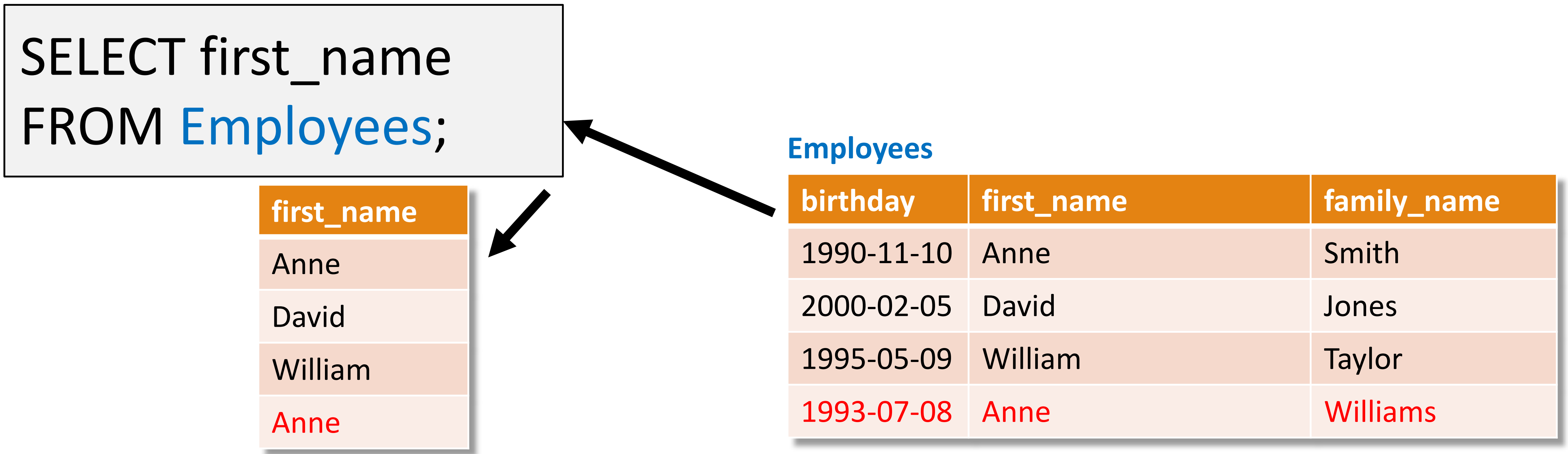
Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

family_name	birthday
Smith	1990-11-10
Jones	2000-02-05
Taylor	1995-05-09

DISTINCT

DISTINCT is for removing duplicated rows



```
SELECT first_name
FROM Employees;
```

first_name
Anne
David
William
Anne

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor
1993-07-08	Anne	Williams

DISTINCT

DISTINCT is for removing duplicated rows

```
SELECT DISTINCT first_name
FROM Employees;
```

first_name
Anne
David
William

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor
1993-07-08	Anne	Williams

Renaming (ρ)

Renaming allows you to rename attributes

You use it by writing AS and then the new name after the attribute

```
SELECT birthday, first_name, family_name AS surname
FROM Employees;
```

birthday	first_name	surname
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Creating new columns

Can create new columns, using e.g. math on current columns (can also do constants and other things)

Can also do e.g. +, -, / and % (modulo, i.e. reminder after division)

Typically you would also give them a name ala renaming

```
SELECT name, price, number, price * number AS total_cost
FROM Items;
```

name	price	number	total_cost
2L Cola	3.00	30	90.00
Banana	0.10	120	12.00
Toilet paper	2.00	0	0.00

Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120
Toilet paper	2.00	0

Creating new columns: cont.

One can also do aggregates, like sums or counts (over the output table)

- We will, in the optional part of SQL queries, get to GROUP BY that lets you do it over sub-parts of the output table

E.g. if you want to find the number of items in the shop – i.e. the sum of the item numbers:

```
SELECT SUM(number)
FROM Items;
```

Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120
Toilet paper	2.00	0

SUM(number)
150

Can also do COUNT (for counting the number of rows), AVG, MIN and MAX (for average, min or max resp.)

Mix and match

The four modification types can also be mixed and matched as wanted

```
SELECT name, price AS price_for_each, price * number AS price_for_all
FROM Employees;
```

name	price_for_each	price_for_all
2L Cola	3.00	90.00
Banana	0.10	12.00
Toilet paper	2.00	0.00

Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120
Toilet paper	2.00	0

FROM

FROM defines the input and would be easy if we only allowed one input

In general, FROM can contain many input tables and we combine them together in various ways

- How we combine the tables is defined by FROM

Only some ways of combining will be discussed in this course, but there exists others

The primarily used ones will be:

1. Cross product (×)
2. Natural join (⋈)
   - (We will also use something called left semijoins, but they are syntactically not done in FROM)

Cross product (×)

Did not use * due to space on this slide

Simplest in a mathematical sense kind of joining table (but it is one students often struggle with)

```
SELECT first_name, name
FROM Employees, Items;
```

first_name	name
Anne	2L Cola
Anne	Banana
David	2L Cola
David	Banana
William	2L Cola
William	Banana

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

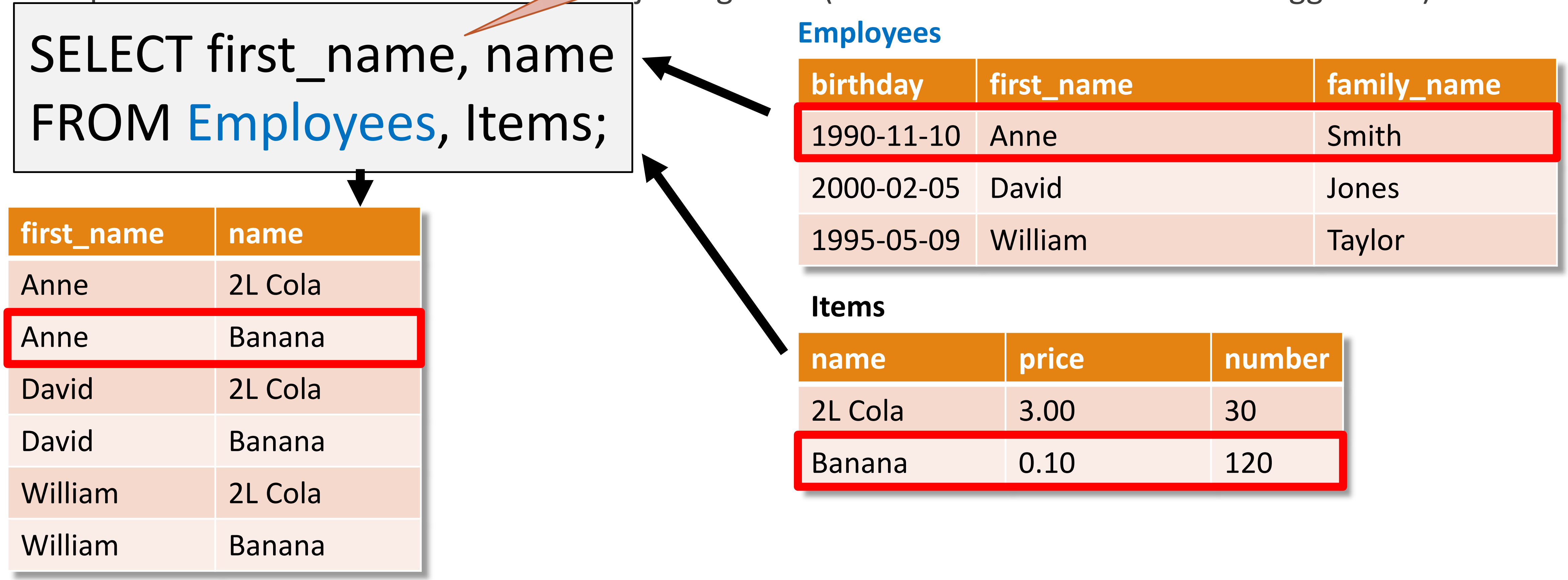
Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120

Cross product (×)

Did not use * due to space on this slide

Simplest in a mathematical sense kind of joining table (but it is one students often struggle with)



```
SELECT first_name, name
FROM Employees, Items;
```

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120

first_name	name
Anne	2L Cola
Anne	Banana
David	2L Cola
David	Banana
William	2L Cola
William	Banana

Cross product (×)

Did not use * due to space on this slide

Simplest in a mathematical sense kind of joining table (but it is one students often struggle with)

```
SELECT first_name, name
FROM Employees, Items;
```

first_name	name
Anne	2L Cola
Anne	Banana
David	2L Cola
David	Banana
William	2L Cola
William	Banana

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120

Cross product (×)

Did not use * due to space on this slide

Simplest in a mathematical sense kind of joining table (but it is one students often struggle with)

```
SELECT first_name, name
FROM Employees, Items;
```

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120

first_name	name
Anne	2L Cola
Anne	Banana
David	2L Cola
David	Banana
William	2L Cola
William	Banana

Cross product (×)

Did not use * due to space on this slide

Simplest in a mathematical sense kind of joining table (but it is one students often struggle with)

```
SELECT first_name, name
FROM Employees, Items;
```

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Items

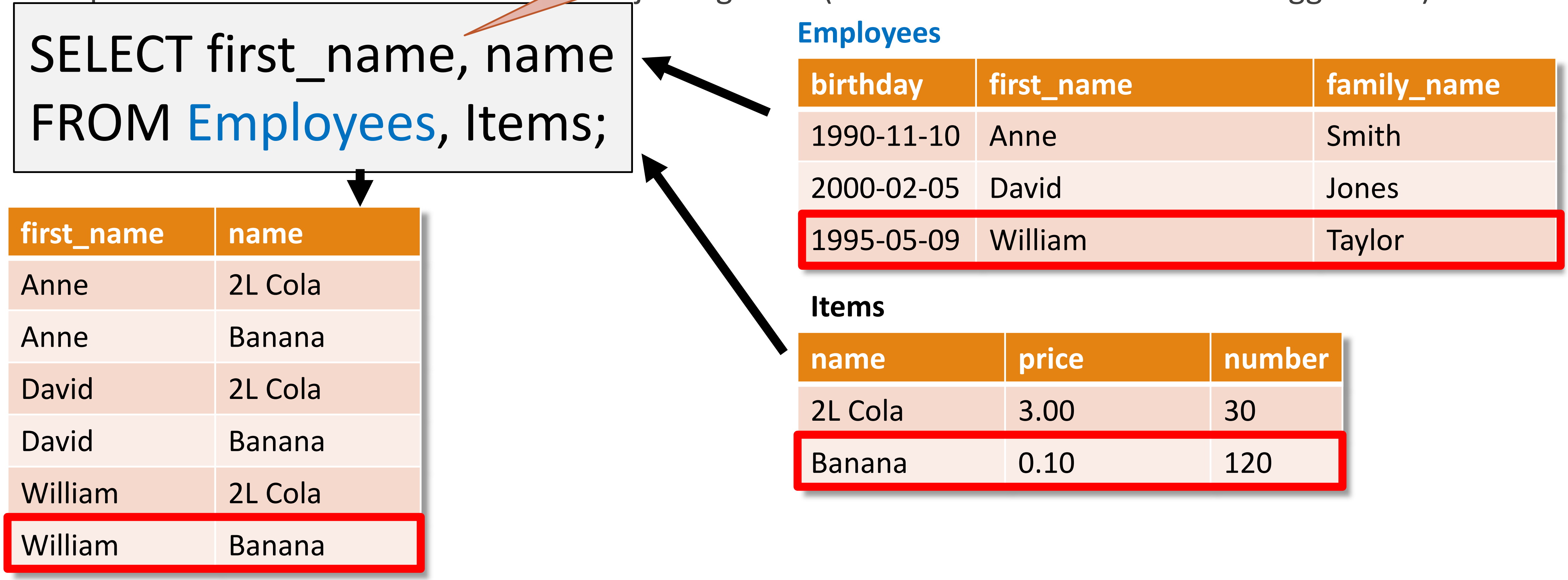
name	price	number
2L Cola	3.00	30
Banana	0.10	120

first_name	name
Anne	2L Cola
Anne	Banana
David	2L Cola
David	Banana
William	2L Cola
William	Banana

Cross product (×)

Did not use * due to space on this slide

Simplest in a mathematical sense kind of joining table (but it is one students often struggle with)



```
SELECT first_name, name
FROM Employees, Items;
```

first_name	name
Anne	2L Cola
Anne	Banana
David	2L Cola
David	Banana
William	2L Cola
William	Banana

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

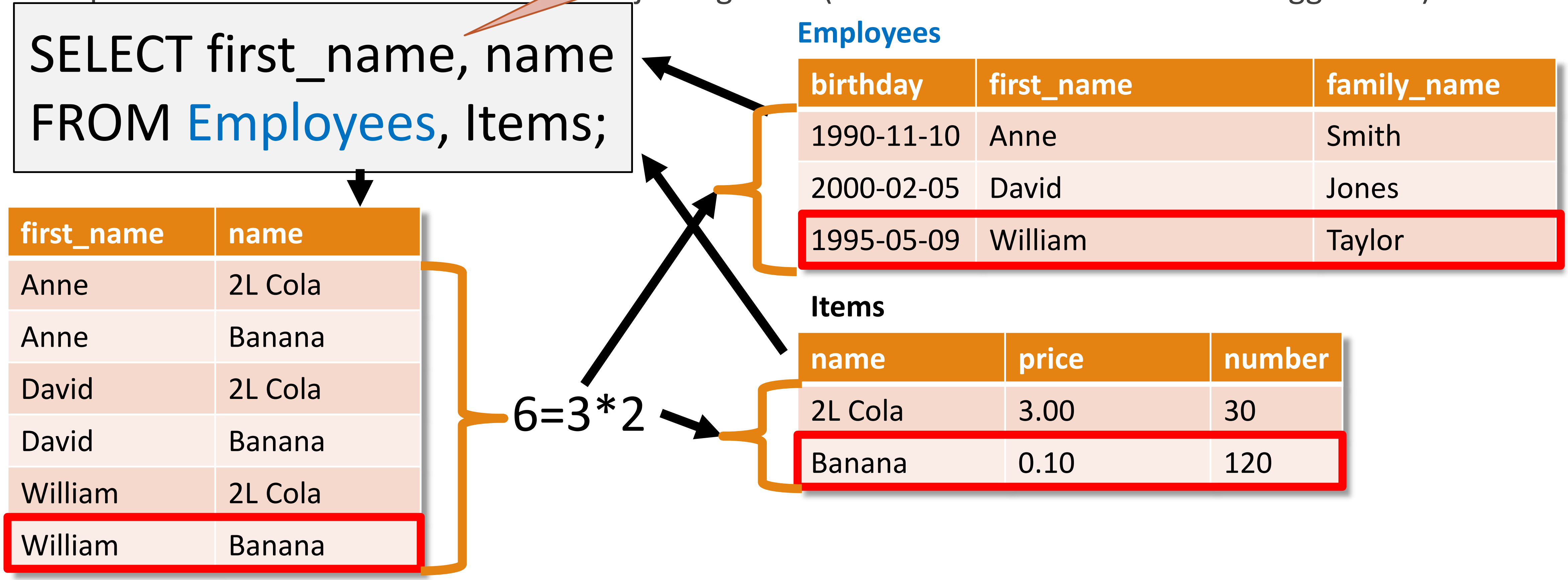
Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120

Cross product (×)

Did not use * due to space on this slide

Simplest in a mathematical sense kind of joining table (but it is one students often struggle with)



```
SELECT first_name, name
FROM Employees, Items;
```

first_name	name
Anne	2L Cola
Anne	Banana
David	2L Cola
David	Banana
William	2L Cola
William	Banana

6=3*2

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Items

name	price	number
2L Cola	3.00	30
Banana	0.10	120

Natural join (⋈)

While more mathematically complex, most seems to understand it much easier (as the name suggest, it is a quite natural way of joining things)

The two tables should have some overlap (often id numbers, e.g. student or employee id)

```
SELECT *
FROM Employees NATURAL JOIN
Transactions;
```

Employees

birthday	first_name	family_name	e_id
1990-11-10	Anne	Smith	1
2000-02-05	David	Jones	2
1995-05-09	William	Taylor	3

Transactions

t_id	c_id	e_id
1	3	1
2	6	1
3	19	3

Natural join (⋈) – cont.

```
SELECT *
FROM Employees NATURAL JOIN
Transactions;
```

birthday	first_name	family_name	e_id	t_id	c_id
1990-11-10	Anne	Smith	1	1	3
1990-11-10	Anne	Smith	1	2	6
1995-05-09	William	Taylor	3	3	19

Employees

birthday	first_name	family_name	e_id
1990-11-10	Anne	Smith	1
2000-02-05	David	Jones	2
1995-05-09	William	Taylor	3

Transactions

t_id	c_id	e_id
1	3	1
2	6	1
3	19	3

Natural join (⋈) – cont.

```
SELECT *
FROM Employees NATURAL JOIN
Transactions;
```

Employees

birthday	first_name	family_name	e_id
1990-11-10	Anne	Smith	1
2000-02-05	David	Jones	2
1995-05-09	William	Taylor	3

Transactions

t_id	c_id	e_id
1	3	1
2	6	1
3	19	3

birthday	first_name	family_name	e_id	t_id	c_id
1990-11-10	Anne	Smith	1	1	3
1990-11-10	Anne	Smith	1	2	6
1995-05-09	William	Taylor	3	3	19

Natural join (⋈) – cont.

```
SELECT *
FROM Employees NATURAL JOIN
Transactions;
```

birthday	first_name	family_name	e_id	t_id	c_id
1990-11-10	Anne	Smith	1	1	3
1990-11-10	Anne	Smith	1	2	6
1995-05-09	William	Taylor	3	3	19

Employees

birthday	first_name	family_name	e_id
1990-11-10	Anne	Smith	1
2000-02-05	David	Jones	2
1995-05-09	William	Taylor	3

Transactions

t_id	c_id	e_id
1	3	1
2	6	1
3	19	3

Natural join (⋈) – cont. 2

Formally, you take a cross product, remove all rows where the common attributes do not match and then only keep one column for each common attribute

```
SELECT *
FROM Employees NATURAL JOIN
Transactions;
```

I.e. if we took the two tables from last slide, with schema
Employees(birthday,first_name,family_name,e_id)
and Transactions(t_id,c_id,e_id) respectively.
Then, these two queries are equal…

```
SELECT birthday, first_name, family_name,
Employees.e_id AS e_id, t_id, c_id
FROM Employees, Transactions
WHERE Employees.e_id = Transactions.e_id;
```

This is how you reference an attribute multiple tables have in common

Natural join (⋈) – cont. 2

Formally, you take a cross product, remove all rows where the common attributes do not match and then only keep one column for each common attribute

```
SELECT *
FROM Employees NATURAL JOIN
Transactions;
```

I.e. if we took the two tables from last slide, with schema
Employees(birthday,first_name,family_name,e_id)
and Transactions(t_id,c_id,e_id) respectively.
Then, these two queries are equal…

```
SELECT birthday, first_name, family_name,
E.e_id AS e_id, t_id, c_id
FROM Employees E, Transactions T
WHERE E.e_id = T.e_id;
```

… and this is how to write a short hand for the names

Natural join (⋈): WARNING

Warning: while students generally have an easier time understanding them, natural join has some issues!

```
SELECT *
FROM Employees NATURAL JOIN
Transactions;
```

birthday	first_name	family_name	e_id	t_id	c_id
1990-11-10	Anne	Smith	1	1	3
1990-11-10	Anne	Smith	1	2	6
1995-05-09	William	Taylor	3	3	19

Natural join (⋈): WARNING

Warning: while students generally have an easier time understanding them, natural join has some issues!

```
SELECT *
FROM Employees NATURAL JOIN
Transactions NATURAL JOIN Customers;
```

Customers

first_name	family_name	c_id
Victor	Wiliams	1
Benjamin	Jones	2
Kate	Sanders	3

birthday	first_name	family_name	e_id	t_id	c_id

birthday	first_name	family_name	e_id	t_id	c_id
1990-11-10	Anne	Smith	1	1	3
1990-11-10	Anne	Smith	1	2	6
1995-05-09	William	Taylor	3	3	19

This happens because e.g. first_name in Customers gets matched to first_name in Employees

Could solve it by using c_first_name and e_first_name e.g. – still you will miss that sometime