Web and Database Technology

SQL

Dr. Albert John Power

a.j.power@tudelft.nl

TU Delft WIS Web Information Systems

- WebLab:

  - **SQL practice**

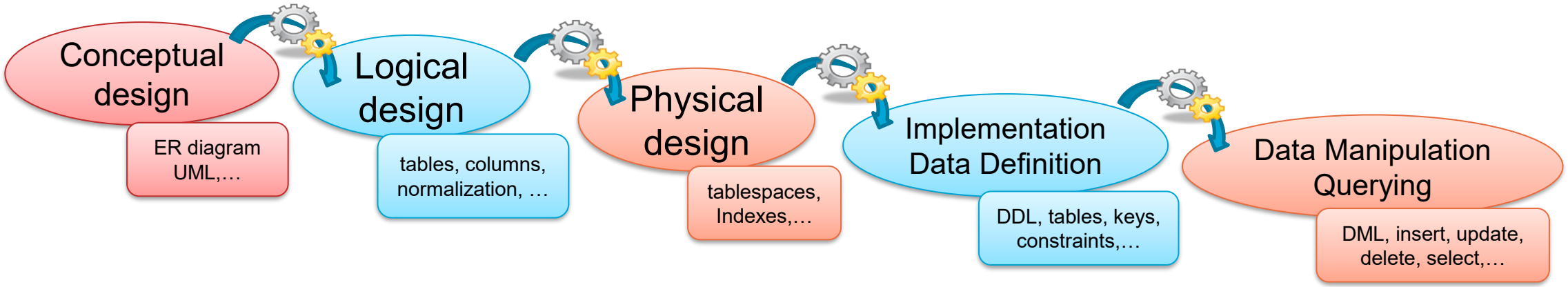
- Brightspace:

  - **Extra material & Quizzes**
  - **Assignment 2**

- On **Wednesday** there will be two **tutorials**:

- One at **14:00 in Flux Hall A** and another at **15:30 in Flux Hall B**.
- Both tutorials will be **streamed in Flux Hall C**.

Databases

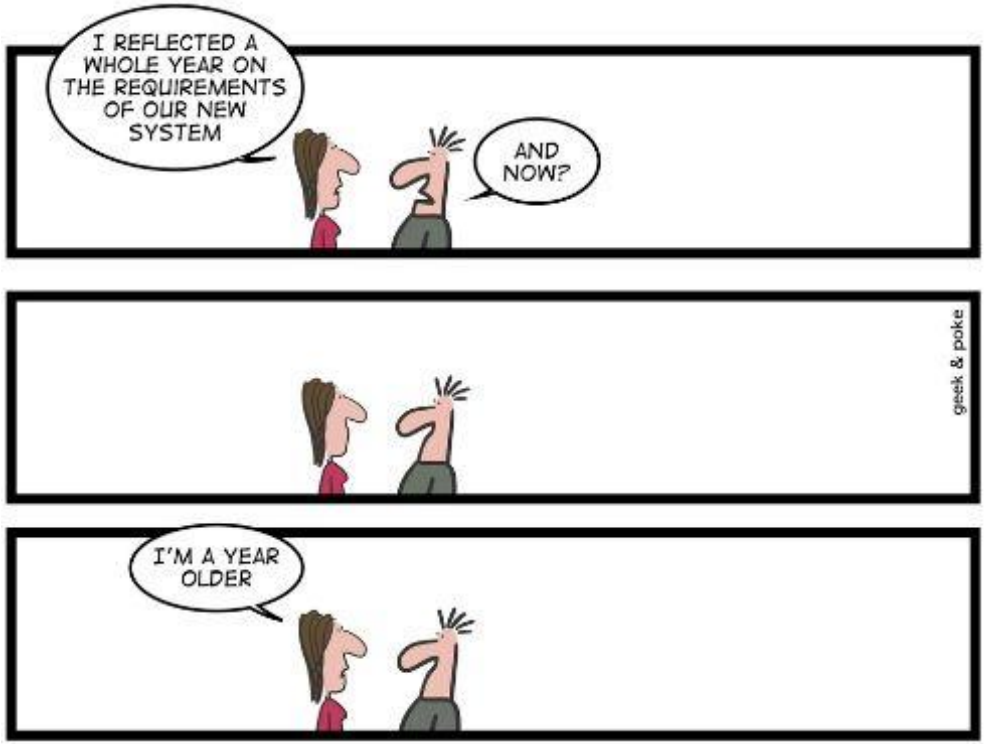

Conceptual design
ER diagram UML,…
Logical design
tables, columns, normalization, …
Physical design
tablespaces, Indexes,…
Implementation Data Definition
DDL, tables, keys, constraints,…
Data Manipulation Querying
DML, insert, update, delete, select,…

SQL

TU Delft WIS Web Information Systems

Requirements of a database language

- Functional requirements
  - Create database and relation structures
  - Perform data management tasks
    - Insertion, Modification, Deletion
  - Perform simple and complex queries



- Non-functional requirements
  - Structure and syntax easy to learn
  - Portability across systems

The SQL Language

- The name is an acronym for **Structured Query Language**
- Far richer than a query language: a **DML**, a **DDL/VDL**
- SQL is a set-based language
  - Operators works on relations (tables)
  - Results are always relations (tables)
  - Rows (tuples/records), Columns (attributes/fields)
- SQL is declarative
  - **It describes what to do with data, not how to do it**

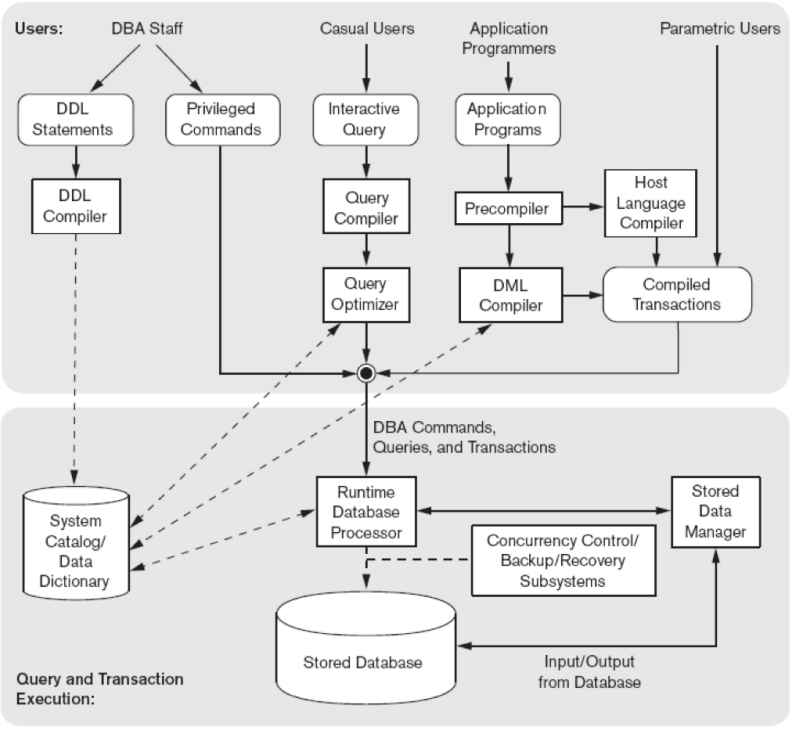
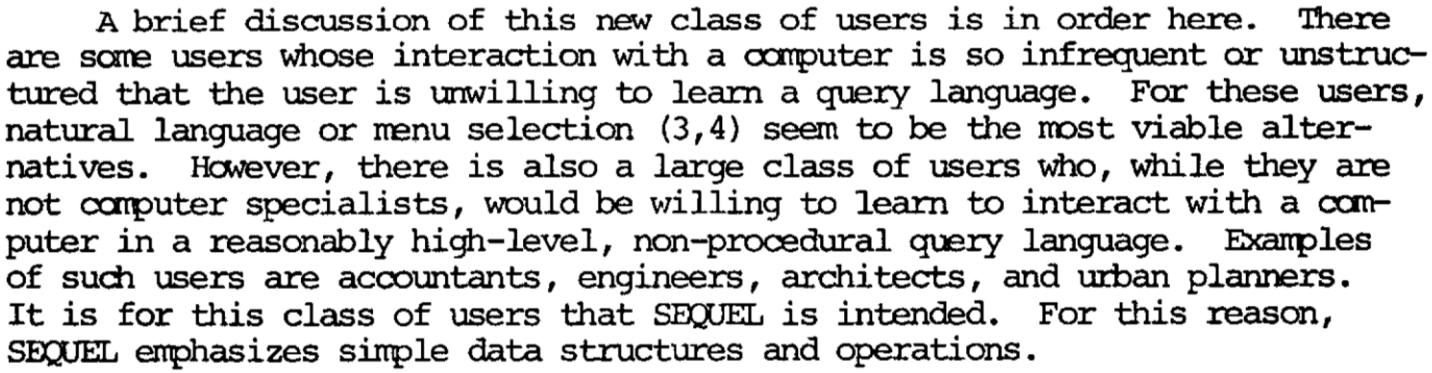


Figure 2.3
Component modules of a DBMS and their interactions.

The SQL Language

- SQL is intergalactic data speak
  - Successful, mainstream, and general purpose **4GL** (fourth generation programming language) — Perhaps the only one…



A brief discussion of this new class of users is in order here. There are some users whose interaction with a computer is so infrequent or unstructured that the user is unwilling to learn a query language. For these users, natural language or menu selection (3,4) seem to be the most viable alternatives. However, there is also a large class of users who, while they are not computer specialists, would be willing to learn to interact with a computer in a reasonably high-level, non-procedural query language. Examples of such users are accountants, engineers, architects, and urban planners. It is for this class of users that SEQUEL is intended. For this reason, SEQUEL emphasizes simple data structures and operations.

Chamberlin, Boyce. http://www.joakimdalby.dk/HTM/sequel.pdf

- Many standards out there:
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
  - Vendors support various subsets (or supersets!)

The SQL Language

- There are **three major classes** of DB operations
  - Defining relations, attributes, domains, constraints, ...
    - **Data Definition Language** (DDL)
  - Adding, deleting, and modifying tuples
    - **Data Manipulation Language** (DML)
  - Asking queries
    - Often part of the DML

Data Definition Language

- Specification of the database schema: creation, modification, deletion of tables
  - CREATE TABLE, ALTER TABLE, DROP TABLE
- Specification of derived tables, or views
  - CREATE VIEW, ALTER VIEW, DROP VIEW
- Specification of indexes
  - CREATE INDEX, DROP INDEX
- Management of user access control
  - GRANT, REVOKE

Data Manipulation Language

- Instructions to retrieve information of interests
  - SELECT
- Instructions to modify instances of the database
  - INSERT: Add new row(s) to a table
  - UPDATE: Modify existing row(s) in a table
  - DELETE: Delete existing row(s) from a table

SQL as a Query Language

- SQL expresses queries in declarative way
  - Queries specify the properties of the result, not the way to obtain it
- Queries are translated by the query optimizer into the procedural language internal to the DBMS
- The programmer should focus on readability, not on efficiency

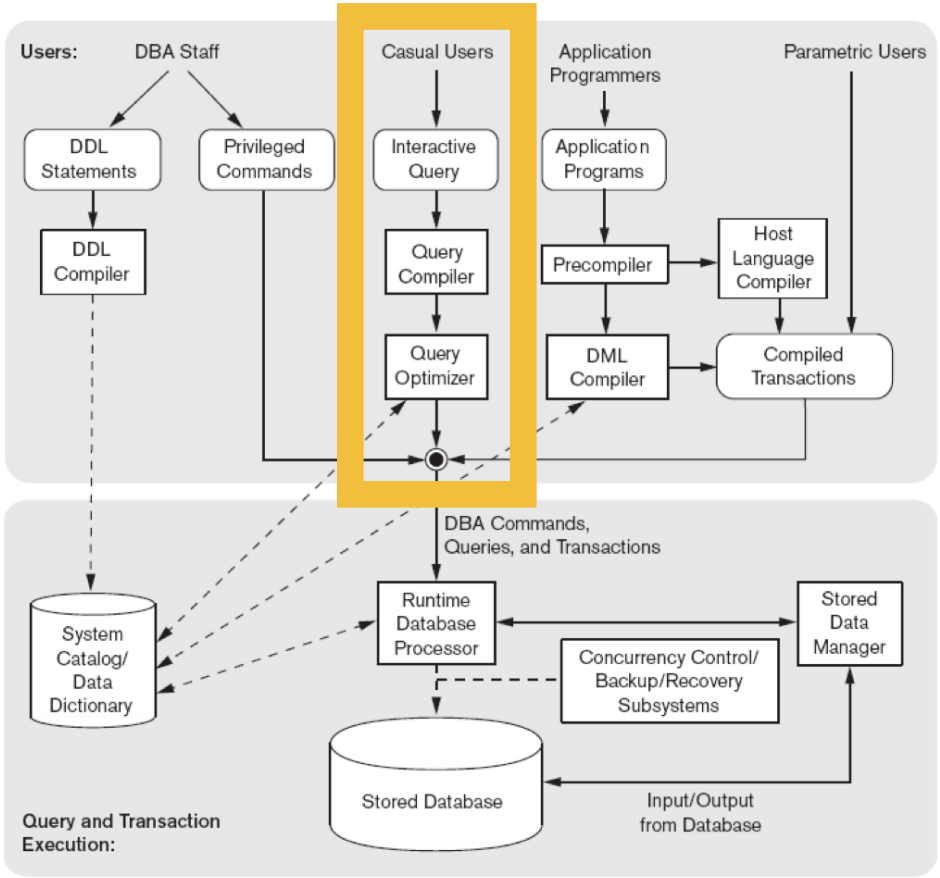


Figure 2.3
Component modules of a DBMS and their interactions.

Example Databases

Example DB1: Employees

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

Example DB2: Products

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

SQL Queries

- The three parts of the query are usually called:
  - **Target list** or SELECT clause (Attributes)
  - FROM clause (Tables)
  - WHERE clause (Conditions)

```
SELECT TARGETLIST
FROM TABLE
[ WHERE CONDITIONS ] [ ORDER BY ORDERINGATTRIBUTESLIST ]
[ GROUP BY GROUPINGATTRIBUTESLIST ] [ HAVING AGGREGATECONDITIONS ]
```

- The query:
  - Considers the cartesian product of the tables in the FROM clause
  - Considers only the rows that **satisfy** (evaluate to TRUE) the condition in the WHERE clause
  - For each row evaluates the attribute expressions in the TargetList, and returns them
  - More on GROUP BY and HAVING later

Select Clause

SELECT query FROM

- Find the *code* of **all** products in the DB

```
SELECT CODEP
FROM PRODUCTS
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP
P1
P2
P3
P4
P5
P6

SELECT query FROM

- Find the *code* and *number of shareholders* of suppliers ***located in*** "Den Haag"

```
SELECT CODES, SHAREHOLDERS
FROM SUPPLIER
WHERE OFFICE = 'DEN HAAG'
```

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

CodeS	Shareholders
S2	1
S3	3

Only the tuples evaluating the logical expression in the WHERE clause to TRUE are selected

* in the target list

- Find all the information relating to employees **named** "Brown"

```
SELECT *
FROM EMPLOYEE
WHERE SURNAME = 'BROWN'
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	Brown	Planning	14	80	London

Aliases

Attribute Expressions with AS

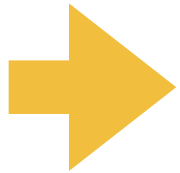
- The keyword AS allows the definition of an **alias**. Used in attribute expressions, it defines a new **temporary** column per the calculated expression

- Find the salaries of employees *named* "Brown", and alias it as "Remuneration"

```
SELECT SALARY AS REMUNERATION
FROM EMPLOYEE
WHERE SURNAME = 'BROWN'
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse



Remuneration
45
80

Attribute Expressions with AS

- Find the monthly salary of the employees **named** "White"

```
SELECT SALARY / 12 AS MONTHLYSALARY
FROM EMPLOYEE
WHERE SURNAME = 'WHITE'
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

MonthlySalary
3.00

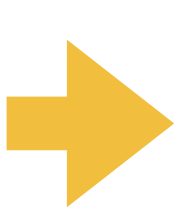
Duplicates

Duplicates

```
SELECT CITY
FROM DEPARTMENT
```

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

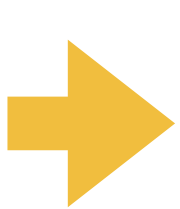


City
London
Toulouse
Brighton
London
San Joné

```
SELECT DISTINCT CITY
FROM DEPARTMENT
```

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné



City
London
Toulouse
Brighton
San Joné

- In relational algebra and calculus, the results of queries **do not contain duplicates** (set semantics)
- In SQL, result sets **may have identical rows** (bag semantics)
- Duplicate **rows** can be removed using the keyword `DISTINCT`
  - DISTINCT comes directly after the SELECT clause
  - DISTINCT also applies to rows with multiple columns (eg SELECT DISTINCT Address, City)

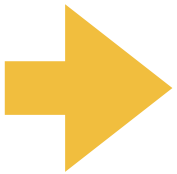
DISTINCT Keyword

- Find the code of the products **supplied <u>at least</u> by one supplier**

```
SELECT DISTINCT CODEP
FROM SUPPLY
```

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400



CodeP
P1
P2
P3
P4
P5
P6

Clarification: ALL, DISTINCT, *

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

- SELECT CITY
- SELECT ALL CITY
- SELECT DISTINCT CITY
- SELECT *
- SELECT DISTINCT *

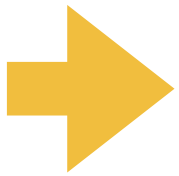
Limit the results

LIMIT Keyword

```
SELECT DISTINCT CITY
FROM DEPARTMENT
```

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

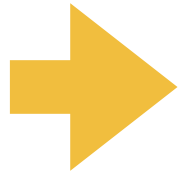


City
London
Toulouse
Brighton
San Joné

```
SELECT DISTINCT CITY
FROM DEPARTMENT
LIMIT 2
```

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné



City
London
Toulouse

WHERE Clause

WHERE Clause

- **Selection conditions** apply to **each single tuple** resulting from the evaluation of the FROM clause
- Defined as a **boolean expression of simple predicates**
- Simple predicates
  - Comparison between attributes and/or constant values (**=**, **<>**, **<**, **>**, **<=** and **>=**)
  - Range (BETWEEN, NOT BETWEEN)
  - Set membership (IN, NOT IN)
  - Pattern matching (LIKE, NOT LIKE)
  - NULL values (IS NULL, IS NOT NULL)
- String comparison in single quotes
  - Case sensitive?

Predicate Conjunction

- Find the *first names* and *surnames* of the employees **who work in office number** 20 **of the** "Administration" **department**

```
SELECT FIRSTNAME, SURNAME
FROM EMPLOYEE
WHERE OFFICE = '20' AND DEPT = 'ADMINISTRATION'
```

a boolean expression composed by two simple predicates

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname
Gus	Green

Predicate Conjunction

- Find the *first names* and *surnames* of the employees **who work in the** "Administration" **department and in the** "Production" **department**

```
SELECT FIRSTNAME, SURNAME
FROM EMPLOYEE
WHERE DEPT = 'ADMINISTRATION' AND DEPT = 'PRODUCTION'
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

- No Results!

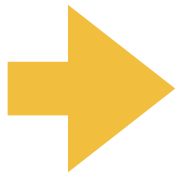
Predicate Disjunction

- Find the *first names* and *surnames* of the employees **who work in <u>either</u> the "Administration" <u>or</u> the "Production" department**

```
SELECT FIRSTNAME, SURNAME
FROM EMPLOYEE
WHERE DEPT = 'ADMINISTRATION' OR DEPT = 'PRODUCTION'
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse



FirstName	Surname
Mary	Brown
Charles	White
Gus	Green
Pauline	Bradshaw
Alice	Jackson

Complex Logical Expressions

- Find the first names of the employees **named** "Brown" **who work in the** "Administration" **department or the** "Production" **department**

```
SELECT FirstName
FROM Employee
WHERE Surname = "Brown" AND (Dept = "Administration" OR Dept = "Production")
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName
Mary

Pattern Matching

TU Delft WIS Web Information Systems

Operator LIKE

- Matching string patterns
- The character "_" is a matching term for any single character, which must be found in the specified position
- The character "%" is a matching term for any sequence of *zero* or more characters

Operator LIKE

- WHERE code **LIKE** 'A_1'

Matches?

'AA1'	yes
'AB1'	yes
'A01'	yes
'A2'	no
'A1'	no
'A21'	yes
'B21'	no
'A221'	no

Operator LIKE

- WHERE name **LIKE** 'A%'

Matches?

'Albert' yes
'A' yes
'JA' no

- WHERE email **LIKE** '%@gmail.com'

Matches any email ending with @gmail.com

- WHERE email **NOT LIKE** '%@gmail.com'

Matches any email that does not end with @gmail.com

Operator LIKE

- Find the *code* and the *name* of the products **having name starting with the letter "S"**

```
SELECT CODEP, NAMEP
FROM PRODUCTS
WHERE NAMEP LIKE "S%"
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P1	Sweater
P3	Shirt
P4	Shirt
P5	Skirt

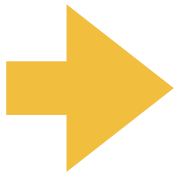
Operator LIKE

- Find the employees with **surnames that have "r" as the second letter and that end in "n"**

```
SELECT *
FROM EMPLOYEE
WHERE SURNAME LIKE "_R%N"
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse



FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Gus	Green	Administration	20	40	Oxford
Charles	Brown	Planning	14	80	London

Operator LIKE

- Find Suppliers **having the Office address containing the string** "Den Haag"

```
WHERE ADDRESS LIKE "%DEN HAAG%"
```

- Find Suppliers where **the supplier code ends in** 2

```
WHERE CODES LIKE "_2"
```

- Find Products that are in storehouses **having names that do not contain an "e" as second character**

```
WHERE STOREHOUSE NOT LIKE "_E%"
```

PostgreSQL LIKE Operator Reference: https://www.postgresql.org/docs/current/functions-matching.html#FUNCTIONS-LIKE
Other PostgreSQL Pattern Matching Operators: https://www.postgresql.org/docs/current/functions-matching.html

Which of the following queries return the same result set?

1

```
SELECT *
FROM CHAR_NAME
WHERE NAME = 'A'
```

2

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE 'A'
```

3

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE 'A_'
```

4

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE 'A%'
```

5

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE 'A%_'
```

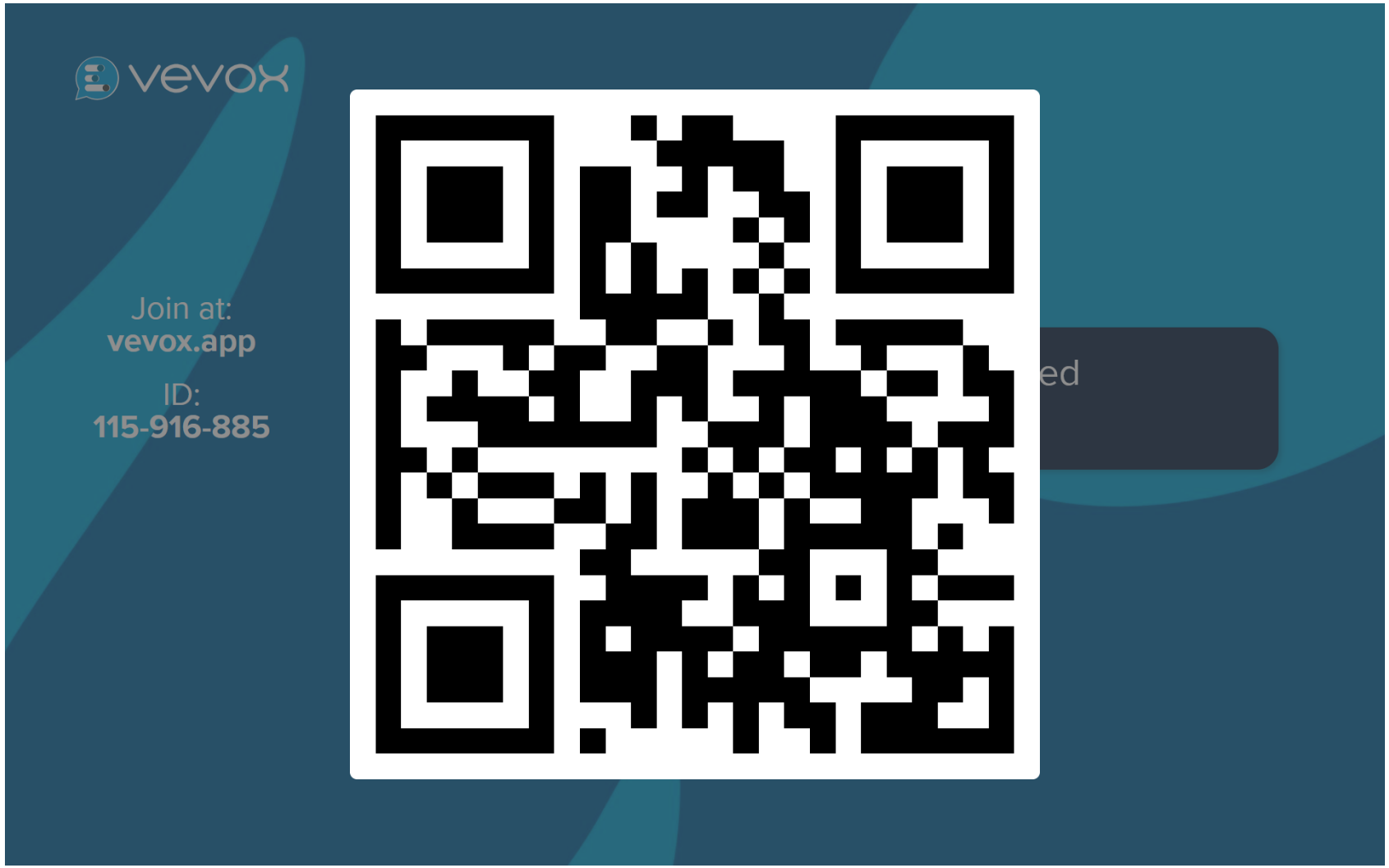
A) Only **1)** and **2)**

B) Only **3)** and **4)**

C) **1)** and **2)** , **4)** and **5)**

D) All

QUIZ



Which of the following queries return the same result set?

1

```
SELECT *
FROM CHAR_NAME
WHERE NAME = 'A'
```

2

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE 'A'
```

3

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE ‘A_’
```

4

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE 'A%'
```

5

```
SELECT *
FROM CHAR_NAME
WHERE NAME LIKE 'A%_'
```

A) Only **1)** and **2)**

B) Only **3)** and **4)**

C) 1) and **2)** , **4)** and **5)**

D) All

Unknown Values

Dealing with NULL values

- SQL uses a three-valued logic
  - TRUE, FALSE, and UNKNOWN
  - All logical operators evaluate to TRUE, FALSE, or UNKNOWN
  - In PostgreSQL, these are implemented as true, false, and NULL
  - Most of this is common to different SQL database servers, although some servers may return any nonzero

- NULL values may mean that:
  - A value is **unknown** (exists but it is not known)
  - A value is **not available** (exists but it is purposely withheld)
  - A value is **not applicable** (undefined for this tuple)

- Each individual NULL value is considered to be different from every other NULL value, so NULL = NULL evaluates to UNKNOWN (not TRUE or FALSE)

- When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN

- DISTINCT and UNKNOW values?

Comparisons involving NULL and Three-Valued Logic

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

The IS NULL Operator

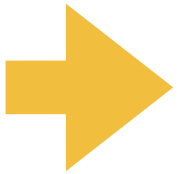
- AttributeName IS [NOT] NULL

- Find the *code* and the *name* of products **having no specified** Size

```
SELECT CODEP, NAMEP
FROM PRODUCTS
WHERE SIZE IS NULL
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam



CodeP	NameP
P5	Skirt

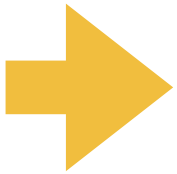
The IS NULL Operator

- Find the *code* and the *name* of products **having size greater than** 44, **or that might have size greater than** 44

```
SELECT CODEP, NAMEP
FROM PRODUCTS
WHERE SIZE > 44 OR SIZE IS NULL
```

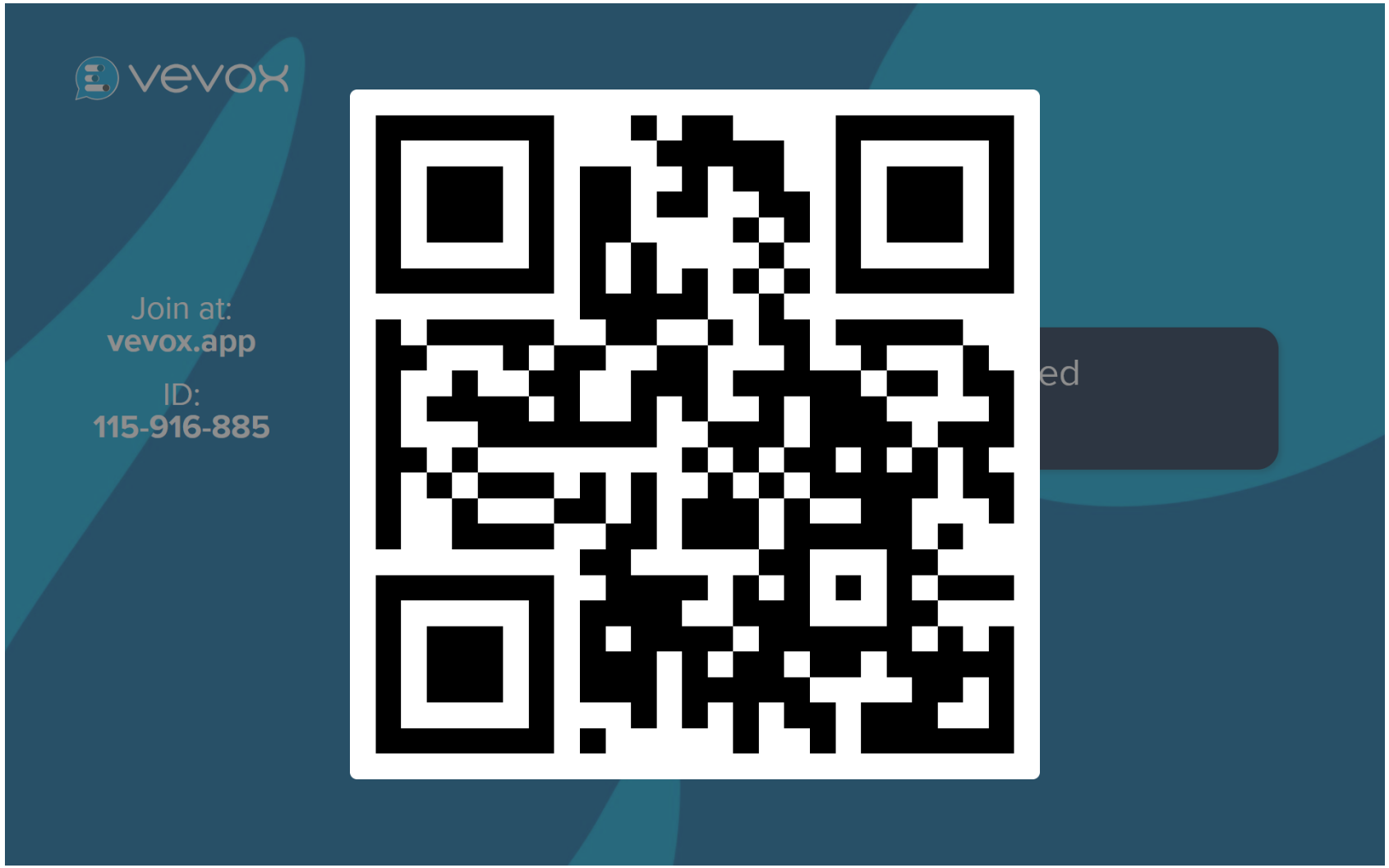
Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam



CodeP	NameP
P2	Jeans
P3	Shirt
P5	Skirt

QUIZ



Ordering of Results

Ordering

- The **ORDER BY** clause, at the end of the query, orders the rows of the results
- Last operator applied by the database in the query execution plan
- Syntax:

```
ORDER BY ORDERINGATTRIBUTE [ ASC | DESC ]
                {, ORDERINGATTRIBUTE [ ASC | DESC ]}
```

- The implicit ordering is **ASC**: ascending

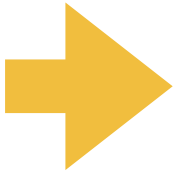
ORDER BY

- Find the *code*, *name* and the *size* of all the products, **in descending order of size**

```
SELECT CODEP, NAMEP, SIZE
FROM PRODUCTS
ORDER BY SIZE DESC
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam



CodeP	NameP	Size
P2	Jeans	48
P3	Shirt	48
P4	Shirt	44
P6	Coat	42
P1	Sweater	40
P5	Skirt	40

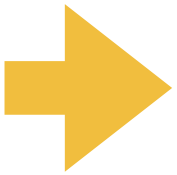
ORDER BY

- Find all the information about products, **in ascending order of** *name* **and descending order of** *size*

```
SELECT *
FROM PRODUCTS
ORDER BY NAMEP, SIZE DESC
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam



CodeP	NameP	Color	Size	Storehouse
P6	Coat	Red	42	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P1	Sweater	Red	40	Amsterdam

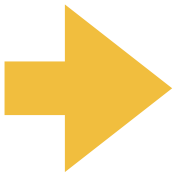
ORDER BY

- Find the code and the *american size* of all the products, **in ascending order of size**

```
SELECT CODEP, SIZE - 14 AS AMERICANSIZE
FROM PRODUCTS
ORDER BY AMERICANSIZE
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam



CodeP	AmericanSize
P1	26
P5	26
P6	28
P4	30
P2	34
P3	34

Querying Multiple Tables

TU Delft WIS Web Information Systems

Querying Multiple Tables

Students	
id	name
1	Melita
2	Albert
3	John

Scores	
id	score
2	85
3	90
4	80

```
SELECT id, name, score
FROM …
```

```
SELECT students.id, students.name, scores.score
FROM …
```

JOINs in SQL92

- SQL-2 introduced an alternative syntax for the representation of JOINs, representing them explicitly in the FROM clause:

```
SELECT TARGETLIST
FROM TABLE [[AS] ALIAS]
  { [JOINTYPE] JOIN TABLE [[AS] ALIAS] [ON BOOLEANEXPRESSION | | USING JOINCOLUMNS]}
[ WHERE CONDITIONS ]
```

- JoinType can be any of INNER, RIGHT [OUTER], LEFT [OUTER] or FULL [OUTER], permitting the representation of outer joins
- The keyword NATURAL may precede JoinType

http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

JOINs in SQL92

- CROSS JOIN
  - Cartesian Product
- INNER JOIN
  - Default type of join in a joined table (equivalent to JOIN)
  - Must specify JOIN attributes
    - Tuple is included in the results only if a matching tuple exists in the other relation
- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple: values for the attributes in the right table set to NULL
- RIGHT OUTER JOIN
  - Every tuple in right table must appear in result
  - If no matching tuple: values for the attributes in the left table set to NULL
- FULL OUTER JOIN
  - If no matching tuple: values for the attributes in the left and/or right tables set to NULL
- NATURAL JOIN on two relations R and S
  - No join condition specified
    - Implicit EQUI JOIN condition for each pair of attributes with the same name from R and S

Joins

Students	
id	name
1	Melita
2	Albert
3	John

Scores	
id	score
2	85
3	90
4	80

```
SELECT students.id, students.name, scores.score
FROM students INNER JOIN scores ON students.id = scores.id
```

```
SELECT students.id, students.name, scores.score
FROM students LEFT JOIN scores ON students.id = scores.id
```

```
SELECT students.id, students.name, scores.score
FROM students RIGHT JOIN scores ON students.id = scores.id
```

```
SELECT students.id, students.name, scores.score
FROM students FULL JOIN scores ON students.id = scores.id
```

INNER JOIN		
id	name	score
2	Albert	85
3	John	90

LEFT JOIN		
id	name	score
1	Melita	NULL
2	Albert	85
3	John	90

RIGHT JOIN		
id	name	score
2	Albert	85
3	John	90
4	NULL	80

FULL JOIN		
id	name	score
1	Melita	NULL
2	Albert	85
3	John	90
4	NULL	80

Using

Students	
id	name
1	Melita
2	Albert
3	John

Scores	
id	score
2	85
3	90
4	80

```
SELECT students.id, students.name, scores.score
FROM students INNER JOIN scores USING(id)
```

```
SELECT students.id, students.name, scores.score
FROM students LEFT JOIN scores USING(id)
```

```
SELECT students.id, students.name, scores.score
FROM students RIGHT JOIN scores USING(id)
```

```
SELECT students.id, students.name, scores.score
FROM students FULL JOIN scores USING(id)
```

Be careful when the tables do not share a common attribute name!

INNER JOIN		
id	name	score
2	Albert	85
3	John	90

LEFT JOIN		
id	name	score
1	Melita	NULL
2	Albert	85
3	John	90

RIGHT JOIN		
id	name	score
2	Albert	85
3	John	90
4	NULL	80

FULL JOIN		
id	name	score
1	Melita	NULL
2	Albert	85
3	John	90
4	NULL	80

Natural Joins

Students	
id	name
1	Melita
2	Albert
3	John

Scores	
id	score
2	85
3	90
4	80

```
SELECT students.id, students.name, scores.score
FROM students NATURAL INNER JOIN scores
```

```
SELECT students.id, students.name, scores.score
FROM students NATURAL LEFT JOIN scores
```

```
SELECT students.id, students.name, scores.score
FROM students NATURAL RIGHT JOIN scores
```

```
SELECT students.id, students.name, scores.score
FROM students NATURAL FULL JOIN scores
```

Be careful when the tables do not share a common attribute name, and also when they share more than one!

NATURAL INNER JOIN		
id	name	score
2	Albert	85
3	John	90

NATURAL LEFT JOIN		
id	name	score
1	Melita	NULL
2	Albert	85
3	John	90

NATURAL RIGHT JOIN		
id	name	score
2	Albert	85
3	John	90
4	NULL	80

NATURAL FULL JOIN		
id	name	score
1	Melita	NULL
2	Albert	85
3	John	90
4	NULL	80

Cross Join

Students	
id	name
1	Melita
2	Albert
3	John

Scores	
id	score
2	85
3	90
4	80

```
SELECT students.id, students.name, scores.score
FROM students CROSS JOIN scores
```

```
SELECT students.id, students.name, scores.score
FROM students, scores
```

CROSS JOIN		
id	name	score
1	Melita	85
1	Melita	90
1	Melita	80
2	Albert	85
2	Albert	90
2	Albert	80
3	John	85
3	John	90
3	John	80

```
SELECT *
FROM students, scores
```

CROSS JOIN			
id	name	id	score
1	Melita	2	85
1	Melita	3	90
1	Melita	4	80
2	Albert	2	85
2	Albert	3	90
2	Albert	4	80
3	John	2	85
3	John	3	90
3	John	4	80

Cross Join

Students	
id	name
1	Melita
2	Albert
3	John

Scores	
id	score
2	85
3	90
4	80

```
SELECT students.id, students.name, scores.score
FROM students CROSS JOIN scores
WHERE students.id = scores.id
```

```
SELECT students.id, students.name, scores.score
FROM students, scores
WHERE students.id = scores.id
```

CROSS JOIN		
id	name	score
2	Albert	85
3	John	90

Joins

- The following statements do the same!
  - `SELECT * FROM r1 JOIN r2 ON r1.a=r2.a`
  - `SELECT * FROM r1 JOIN r2 USING (a)`
  - `SELECT * FROM r1 INNER JOIN r2 ON r1.a=r2.a`
  - `SELECT * FROM r1 INNER JOIN r2 USING (a)`
  - `SELECT * FROM r1, r2 WHERE r1.a=r2.a`
  - `SELECT * FROM r1 CROSS JOIN r2 WHERE r1.a=r2.a`

Using AS keyword for tables

```
SELECT students.id, students.name, scores.score
FROM students INNER JOIN scores ON students.id = scores.id
```

```
SELECT st.id, st.name, sc.score
FROM students AS st INNER JOIN scores sc ON st.id = sc.id
```

Querying Multiple Tables

- What if we want to retrieve:

- The name of all the **suppliers of product** "P2"

Supplier			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply		
CodeS	**CodeP**	**Amount**
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products				
CodeP	**NameP**	**Color**	**Size**	**Storehouse**
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

Cross Join

- All possible tuple combinations / cartesian product

- Find the *name* of **all** the suppliers **of** product "P2"

```
SELECT NAMES
FROM SUPPLIER, SUPPLY
```

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

Cross Join

- Find the *name* of **all** the suppliers **of** product "P2"

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

What is the problem with this result set?

Cross JOIN

```
SELECT NAMES
FROM SUPPLIER, SUPPLY
WHERE SUPPLIER.CODES = SUPPLY.CODES
```

Mention the dot

=

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

Cross JOIN

- Supplier.CodeS = Supply.CodeS is a **JOIN CONDITION**

Supplier				**Supply**		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

Our Original Query

- Find the *name* of **all** the suppliers **of** product "P2"

```
SELECT NAMES
FROM SUPPLIER, SUPPLY
WHERE SUPPLIER.CODES = SUPPLY.CODES AND CODEP = "P2"
```

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

NameS
John
Victor
Anna

Another Query

If there are *N* tables in the FROM clause =>

At least *N – 1* JOIN conditions in the WHERE clause

- Find the *names* of suppliers of **at least one red product**

```
SELECT DISTINCT NAMES
FROM SUPPLIER, SUPPLY, PRODUCTS
WHERE SUPPLIER.CODES = SUPPLY.CODES AND SUPPLY.CODEP = PRODUCTS.CODEP
    AND COLOR = “RED"
```

Supplier

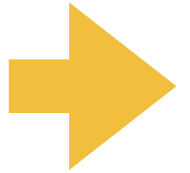
CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam



NameS
John
Victor

Result

- All possible tuple combinations

- Find the *code* **pairs** of suppliers **having their office in the same city**

```
SELECT S1.CODES, S2.CODES
FROM SUPPLIER AS S1, SUPPLIER AS S2
WHERE S1.OFFICE = S2.OFFICE
```

Supplier AS S1			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supplier AS S2			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Result

- **Pairs** of identical values
- **Permutations** of the same pair of values, a redundant information

- Find the *code* **pairs** of suppliers **having their office in the same city**

```
SELECT S1.CODES, S2.CODES
FROM SUPPLIER AS S1, SUPPLIER AS S2
WHERE S1.OFFICE = S2.OFFICE
```

Supplier AS S1			
CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supplier AS S2			
CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

S1.CodeS	S2.CodeS
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1
S4	S4
S5	S5

Result

- Find the *code* **pairs** of suppliers **having their office in the same city**

```
SELECT S1.CODES, S2.CODES
FROM SUPPLIER AS S1, SUPPLIER AS S2
WHERE S1.OFFICE = S2.OFFICE AND S1.CODES <> S2.CODES
```

- Remove pairs with the same code value

S1.CodeS	S2.CodeS
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1
S4	S4
S5	S5

Result

- Let's keep only the right ones

- Find the *code* **pairs** of suppliers **having their office in the same city**

```
SELECT S1.CODES, S2.CODES
FROM SUPPLIER AS S1, SUPPLIER AS S2
WHERE S1.OFFICE = S2.OFFICE AND S1.CODES < S2.CODES
```

S1.CodeS	S2.CodeS
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1
S4	S4
S5	S5

INNER JOIN

- Find the names of suppliers of **at least one red product**

```
SELECT DISTINCT NAMES
FROM PRODUCTS JOIN SUPPLY USING (CODEP)
        JOIN SUPPLIER USING (CODES)
WHERE COLOR = “RED"
```

Supplier

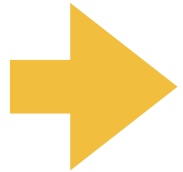
CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam



NameS
John
Victor

LEFT OUTER JOIN

- Find the *code* and *name* of Supplier, and the *code* of the supplied Products, showing also suppliers of no products

```
SELECT SUPPLY.CODES, SUPPLIER.NAMES, SUPPLY.CODEP
FROM SUPPLIER LEFT OUTER JOIN SUPPLY
        ON SUPPLIER.CODES = SUPPLY.CODES
```

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeS	NameS	CodeP
S1	John	P1
S1	John	P2
S1	John	P3
S1	John	P4
S1	John	P5
S1	John	P6
S2	Victor	P1
S2	Victor	P2
S3	Anna	P2
S4	Angela	P3
S4	Angela	P4
S4	Angela	P5
S5	Paul	NULL


Can I join you?
comic.browserling.com
An SQL query walks into a bar and sees two tables.
He walks up to them and says "Can I join you?"

Writing SQL Code

TU Delft WIS Web Information Systems

Writing *Good* SQL Code

- What is ***good* SQL code?**
  - Easy to read
  - Easy to write
  - Easy to understand!

- There is no *official* SQL style guide, but here are some general hints

Writing *Good* SQL Code

1. Write SQL keywords in uppercase, names in lowercase!

BAD

```
SELECT MOVIE_TITLE
FROM MOVIE
WHERE MOVIE_YEAR = 2009;
```

GOOD

```
SELECT movie_title
FROM movie
WHERE movie_year = 2009;
```

Writing *Good* SQL Code

2. Use proper qualification!
(if your DB supports that, not all do)

BAD

```
SELECT imdbraw.movie.movie_title,
imdbraw.movie.movie_year
FROM imdbraw.movie
WHERE imdbraw.movie.movie_year = 2009;
```

GOOD

```
SET SCHEMA imdbraw;
SELECT movie_title, movie_year
FROM movie
WHERE movie_year = 2009;
```

DETOUR

Writing *Good* SQL Code

3. Use aliases to keep your code short and the result clear!

BAD

```
SELECT movie_title, movie_year
FROM movie, genre
WHERE movie.movie_id = genre.movie_id
  AND genre.genre = 'Action';
```

GOOD

```
SELECT movie_title, movie_year
FROM movie m, genre g
WHERE m.movie_id = g.movie_id
  AND g.genre = 'Action';
```

Writing *Good* SQL Code

4. Use JOINs to join!

BAD

```
SELECT movie_title title, genre g
FROM movie m
  JOIN genre g ON g.genre='Action'
WHERE m.movie_id = g.movie_id
```

GOOD

```
SELECT movie_title title, genre g
FROM movie m
  JOIN genre g ON m.movie_id = g.movie_id
WHERE g.genre='Action'
```

Writing *Good* SQL Code

5. Separate JOINs from conditions!

BAD

```
SELECT movie_title title, movie_year year
FROM movie m, genre g, actor a
WHERE m.movie_id = g.movie_id
  AND g.genre = 'Action'
  AND m.movie_id = a.movie_id
  AND a.person_name LIKE '%Schwarzenegger%';
```

GOOD

```
SELECT movie_title title, movie_year year
FROM movie m
  JOIN genre g ON m.movie_id = g.movie_id
  JOIN actor a ON g.movie_id = a.movie_id
WHERE g.genre = 'Action'
  AND a.person_name LIKE '%Schwarzenegger%';
```

Writing *Good* SQL Code

6. Use proper indentation!

BAD

```
SELECT movie_title title, movie_year year
FROM movie m JOIN genre g ON m.movie_id =
g.movie_id JOIN actor a ON g.movie_id =
a.movie_id WHERE g.genre = 'Action' AND
a.person_name LIKE '%Schwarzenegger%';
```

GOOD

```
SELECT movie_title title, movie_year year
FROM movie m
  JOIN genre g ON m.movie_id = g.movie_id
  JOIN actor a ON g.movie_id = a.movie_id
WHERE g.genre = 'Action'
  AND a.person_name LIKE '%Schwarzenegger%';
```

DETOUR

Writing *Good* SQL Code

7. Extract uncorrelated subqueries!

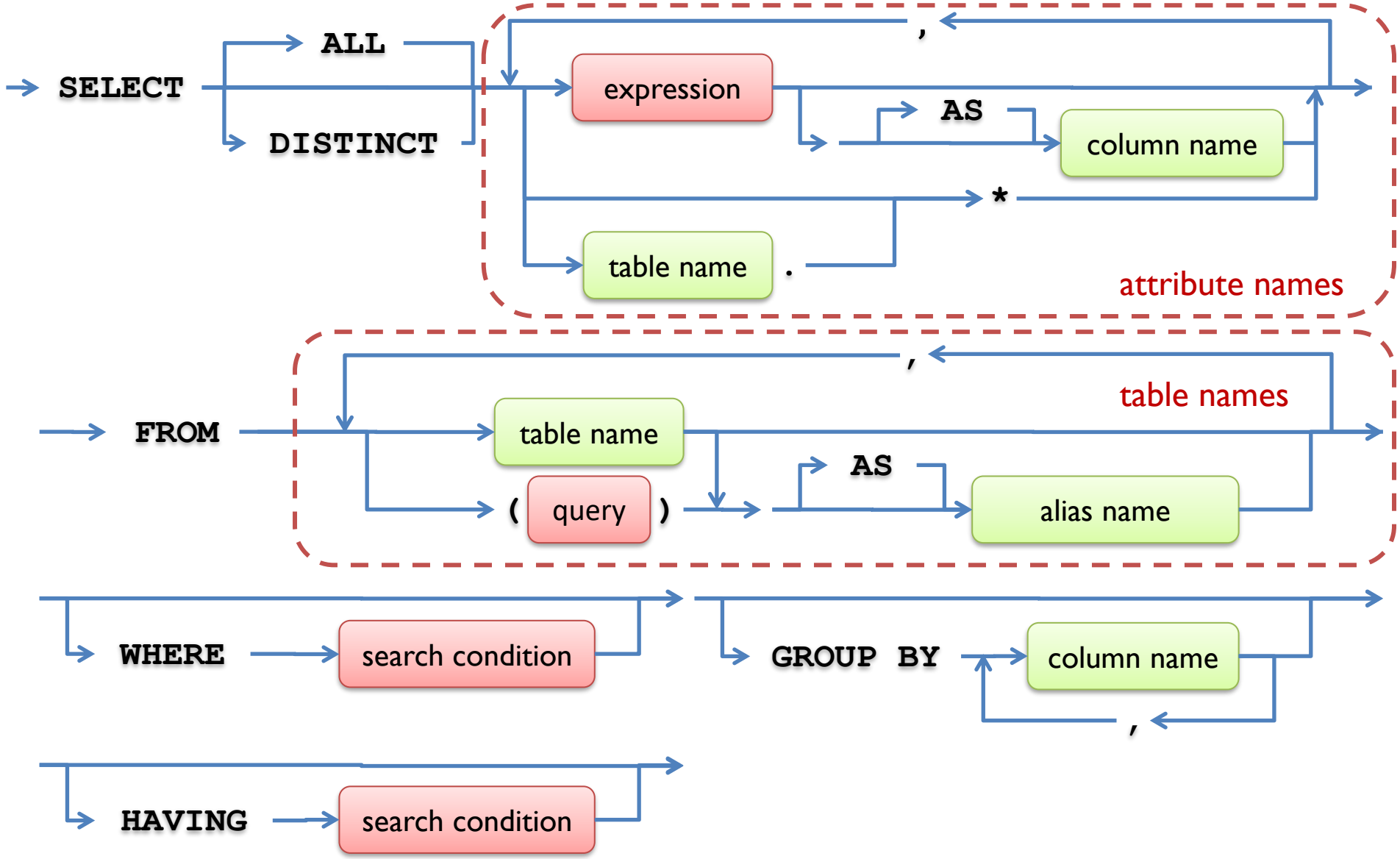
BAD

```
SELECT DISTINCT person_name name
FROM director d
WHERE d.person_id IN (
  SELECT DISTINCT person_id
  FROM actor a
    JOIN movie m ON a.movie_id = m.movie_id
  WHERE movie_year >= 2007
);
```

GOOD

```
WITH recent_actor AS (
  SELECT DISTINCT person_id AS pid
  FROM actor a
    JOIN movie m ON a.movie_id = m.movie_id
  WHERE movie_year >= 2007
)
SELECT DISTINCT person_name name
FROM director d
WHERE d.person_id IN (SELECT * FROM recent_actor);
```

Query block

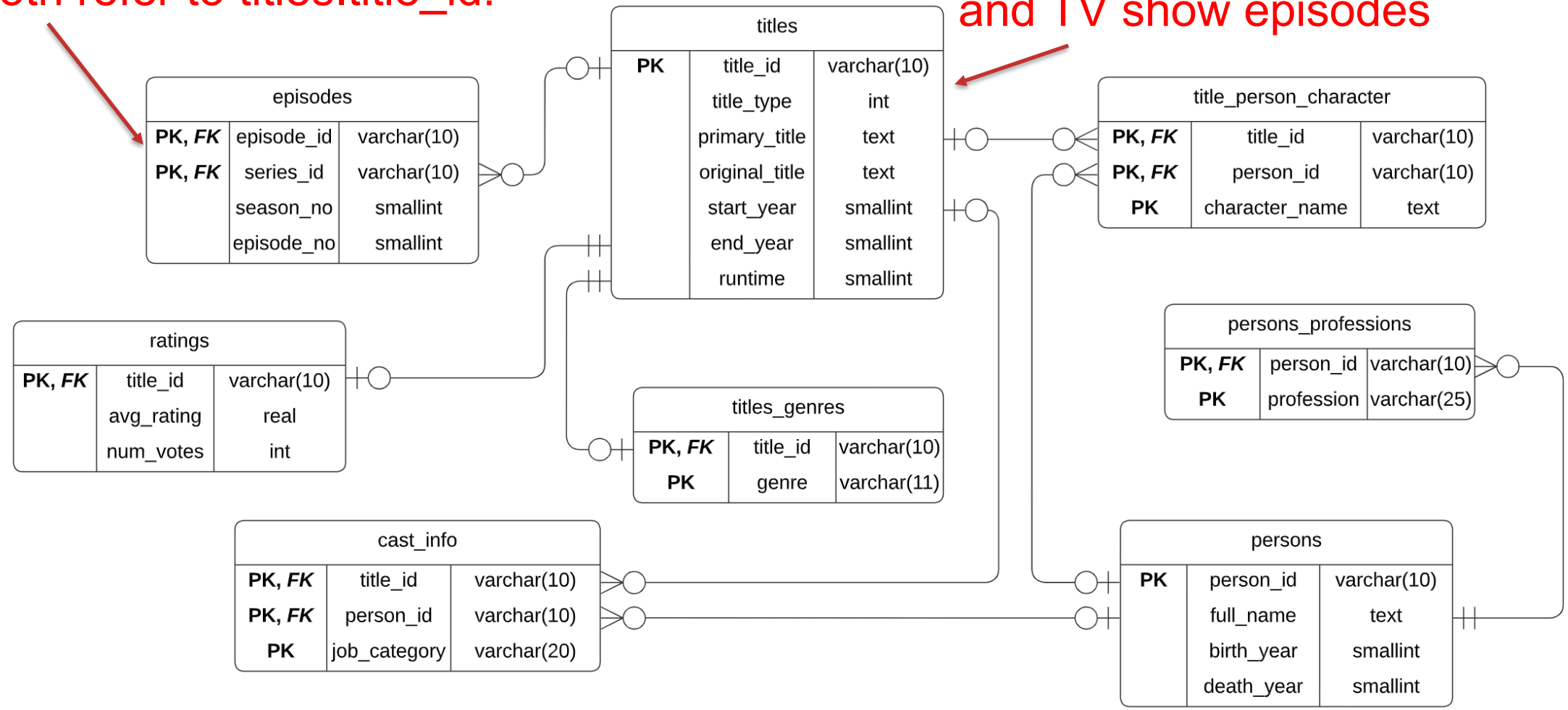

SELECT
ALL
DISTINCT
,
expression
AS
column name
*
table name
.
attribute names
FROM
,
table name
(query)
AS
alias name
table names
WHERE
search condition
GROUP BY
column name
,
HAVING
search condition

The Labs: How do TV shows work

TU Delft WIS Web Information Systems

Both refer to titles.title_id!

Contains movies, TV shows, and TV show episodes



Querying TV Shows / Series

```
Return all TV Episodes released in 2020 with their average rating which
belongs to a TV series/show which started in 2019, sorted by the series'
popularity (given by number of rating).
```

- For example, the most rated show of 2019 was "The Boys", the first 2020 episode of "The Boys" was "The Big Ride" (Season 2 Episode 1) and got an average rating of 8.1.
- This is a good example of joining the same set of tables multiple times in the same query

Querying TV Shows / Series

Return all TV Episodes released in 2020 with their average rating which belongs to a TV series/show which started in 2019, sorted by the series' popularity (given by number of rating).

```
SELECT showTitles.primary_title, showTitles.start_year, showRatings.num_votes AS showRatingVotes,
episodeTitles.primary_title, episodeTitles.start_year, episodes.season_no, episodes.episode_no,
episodeRatings.avg_rating  AS episodeRating
FROM titles AS showTitles  -- These are the titles which represent TV series/shows
JOIN episodes ON showTitles.title_id = episodes.series_id -- This links series and episode titles
JOIN titles AS episodeTitles ON episodeTitles .title_id  = episodes.episode_id  -- These are the titles which
represent episodes
JOIN ratings AS showRatings ON showTitles.title_id = showRatings.title_id -- Get the ratings of the shows/series
JOIN ratings AS episodeRatings ON episodeTitles.title_id = episodeRatings.title_id --Get the rating of the
episodes
WHERE showTitles.title_type='tvSeries' AND  episodeTitles.title_type='tvEpisode' -- Bind to the correct types
(also, no miniseries, etc.)
AND showTitles.start_year=2019  -- Series which start in 2019
AND episodeTitles .start_year = 2020 -- But only episodes which are released in 2020+
ORDER BY showRatings.num_votes DESC, episodes.season_no, episodes.episode_no -- First ordered in descending
order by the no. of votes, then chronologically by the season no. and ep. No.
```

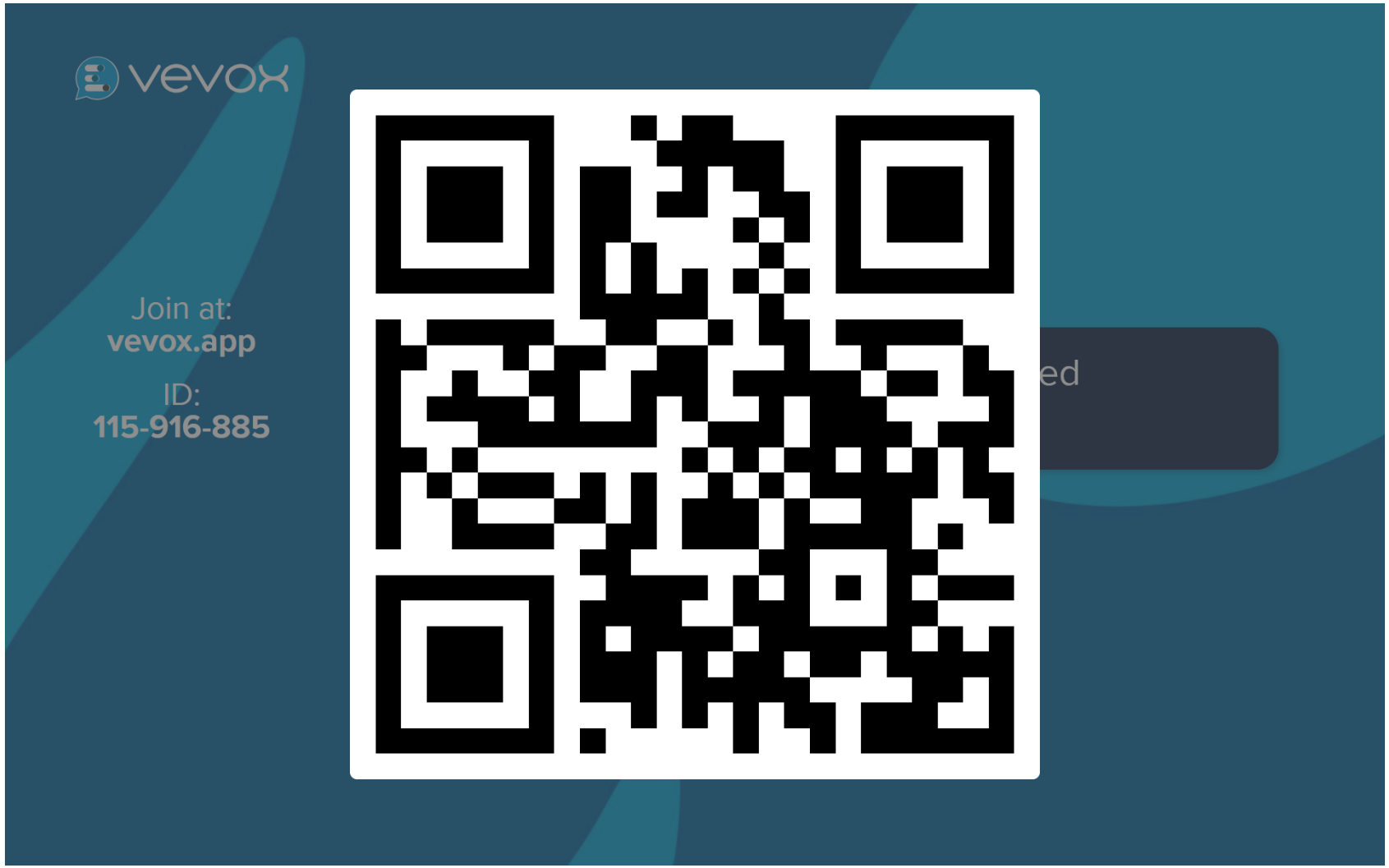
```
SELECT showTitles.primary_title, showTitles.start_year, showRatings.num_votes AS shov
```

Enter a SQL expression to filter results (use Ctrl+Space)

Grid | Text | Record

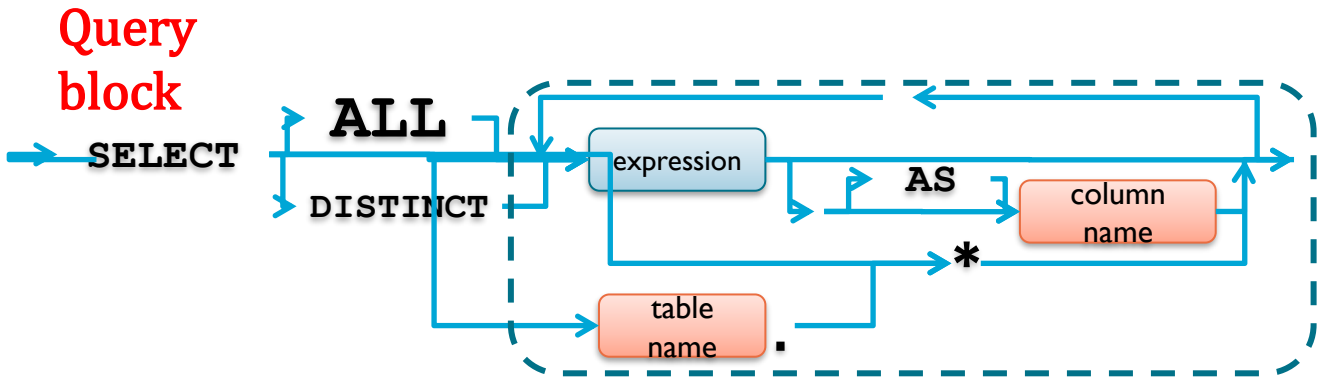
	primary_title	start_year	showRatingVotes	primary_title	start_year	season_no	episode_no	episodeRating
1	The Boys	2,019	484,497	The Big Ride	2,020	2	1	8.1000003815
2	The Boys	2,019	484,497	Proper Preparation and Planning	2,020	2	2	7.6999998093
3	The Boys	2,019	484,497	Over the Hill with the Swords of a Thousand Men	2,020	2	3	9
4	The Boys	2,019	484,497	Nothing Like It in the World	2,020	2	4	8
5	The Boys	2,019	484,497	We Gotta Go Now	2,020	2	5	8.3000001907
6	The Boys	2,019	484,497	The Bloody Doors Off	2,020	2	6	8.8999996185
7	The Boys	2,019	484,497	Butcher, Baker, Candlestick Maker	2,020	2	7	9
8	The Boys	2,019	484,497	What I Know	2,020	2	8	9.3999996185
9	The Mandalorian	2,019	469,766	Chapter 9: The Marshal	2,020	2	1	8.8000001907
10	The Mandalorian	2,019	469,766	Chapter 10: The Passenger	2,020	2	2	7.8000001907
11	The Mandalorian	2,019	469,766	Chapter 11: The Heiress	2,020	2	3	8.6999998093
12	The Mandalorian	2,019	469,766	Chapter 12: The Siege	2,020	2	4	8.3000001907
13	The Mandalorian	2,019	469,766	Chapter 13: The Jedi	2,020	2	5	9.3000001907
14	The Mandalorian	2,019	469,766	Chapter 14: The Tragedy	2,020	2	6	9.1000003815
15	The Mandalorian	2,019	469,766	Chapter 15: The Believer	2,020	2	7	8.8999996185
16	The Mandalorian	2,019	469,766	Chapter 16: The Rescue	2,020	2	8	9.8000001907
17	Sex Education	2,019	276,344	Episode #2.1	2,020	2	1	8.1000003815
18	Sex Education	2,019	276,344	Episode #2.2	2,020	2	2	8
19	Sex Education	2,019	276,344	Episode #2.3	2,020	2	3	8.1999998093
20	Sex Education	2,019	276,344	Episode #2.4	2,020	2	4	8.1000003815
21	Sex Education	2,019	276,344	Episode #2.5	2,020	2	5	7.6999998093
22	Sex Education	2,019	276,344	Episode #2.6	2,020	2	6	8.3999996185
23	Sex Education	2,019	276,344	Episode #2.7	2,020	2	7	9
24	Sex Education	2,019	276,344	Episode #2.8	2,020	2	8	7.8000001907
25	The Umbrella Academy	2,019	245,688	Right Back Where We Started	2,020	2	1	8.1999998093
26	The Umbrella Academy	2,019	245,688	The Frankel Footage	2,020	2	2	8
27	The Umbrella Academy	2,019	245,688	The Swedish Job	2,020	2	3	8.1999998093

QUIZ



Summary

- We learnt how to make simple SQL select queries
  - SELECT attributes FROM tables WHERE condition
- More advanced concepts exist
  - Aggregate and statistical queries
  - Subqueries
  - Sorting
  - Recursion?!
  - Etc.



Web & Database Technology

See you next time!

TU Delft WIS Web Information Systems