Datamodel & SQL

Data

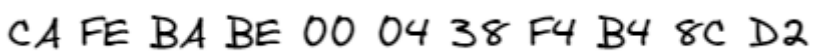
- Who runs the world? Data! (by Digital Beyonce)
- Data is the new digital gold
- Very few applications do *not* store or use data

Data

- Data usage and storage has exploded over the past two decades
    - Facebook has 52.000 data points on each person
    - Google has at least that much on each user
    - Data is stored over time
- Total amount of data created is estimated to be 64 zettabytes in 2020
    - Source: https://www.statista.com/statistics/871513/worldwide-data-created/

Data Models

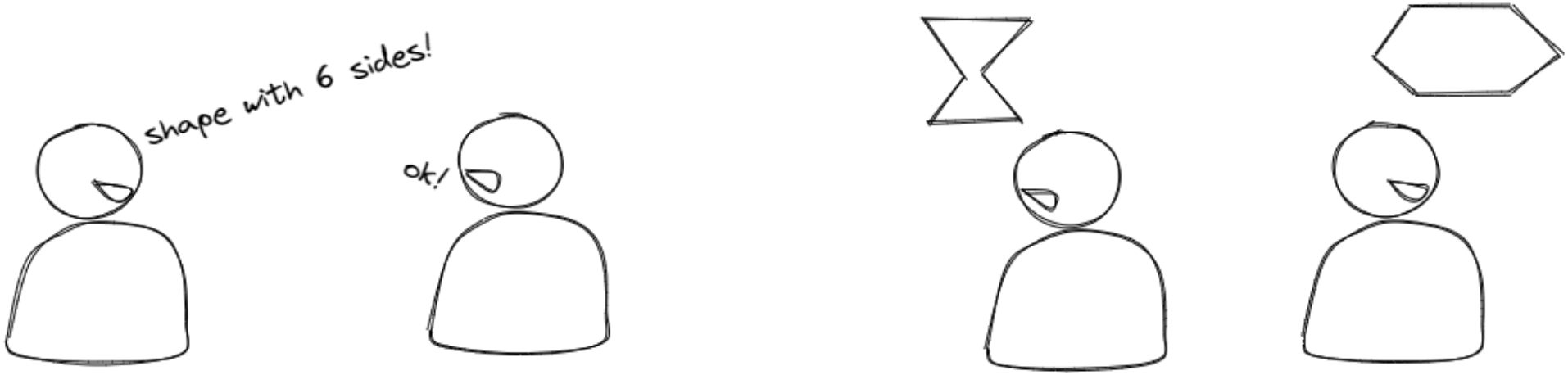
- Data itself is worth nothing



- Interpretation and structure give it meaning
    - The same data can serve multiple purposes, given different structures / interpretations

Data Models

- Data available in any situation far exceeds the need
- Models select important properties and relations
- Models leave out data that is considered useless, or distracting
- Models are used to communicate and create shared understanding



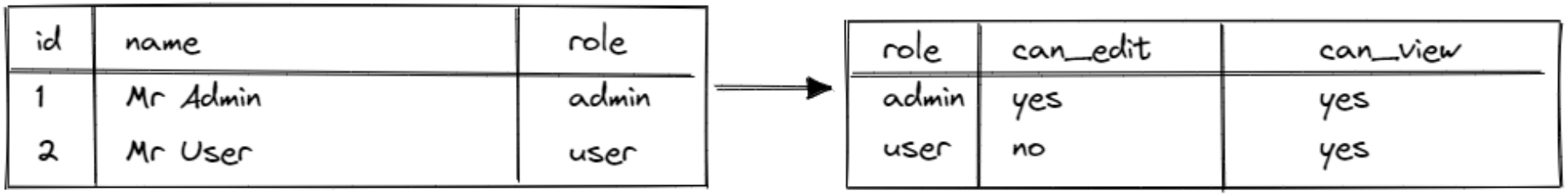
Types of Data Models

- Relational Data

- Document Based Data
- Timeseries Data

Relational Data

- Table-based with columns for fields
    - A row in a table is one instance of data
    - Tables can be related through key fields



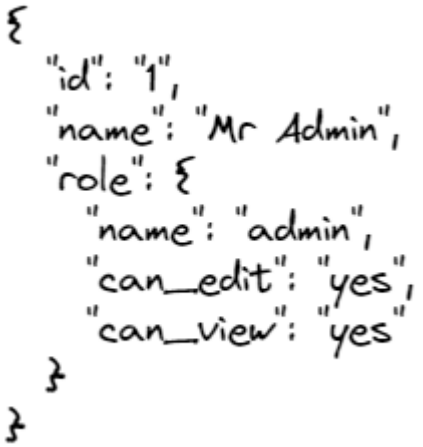
id	name	role
1	Mr Admin	admin
2	Mr User	user

role	can_edit	can_view
admin	yes	yes
user	no	yes

- Popular databases: Oracle, Postgres, MySQL

Document Based Data

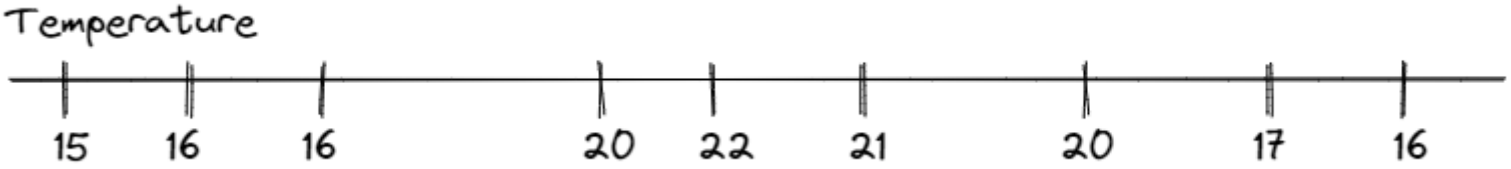
- Collects all relevant data for an entity in one document
    - Related and sub-entities are contained inside document



- Popular databases: MongoDB

Timeseries Data

- Collects data over points in time
    - Time is the key for any datapoint
    - Used to collect sensor data and generate graphs



- Populare databases: Elasticsearch, InfluxDB

Tabular Data (CSV or Excel)

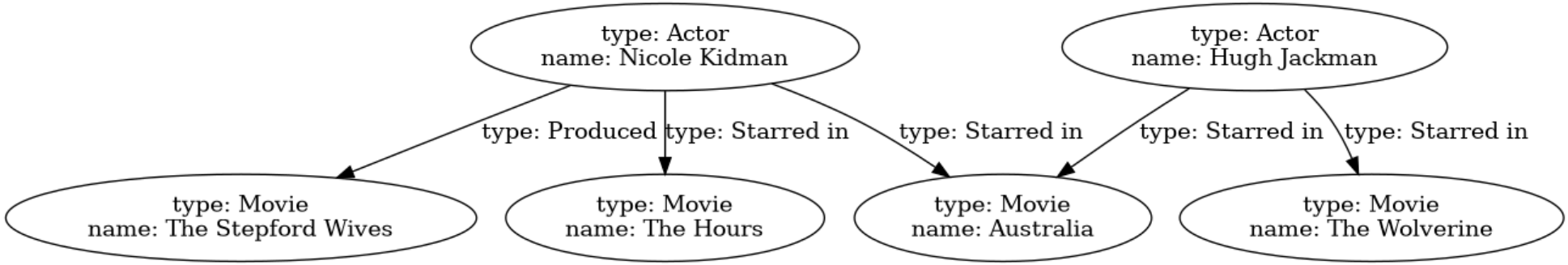
- tabular (`csv`)

```
"first_name","last_name","company_name","address","city","county","state","zip","phone1","phone2","email","we
b"
"James","Butt","Benton, John B Jr","6649 N Blue Gum St","New Orleans","Orleans","LA",70116,"504-621-
8927","504-845-1427","jbutt@gmail.com","http://www.bentonjohnbjr.com"
"Josephine","Darakjy","Chanay, Jeffrey A Esq","4 B Blue Ridge Blvd","Brighton","Livingston","MI",48116,"810-
292-9388","810-374-9840","josephine_darakjy@darakjy.org","http://www.chanayjeffreyaesq.com"
"Art","Venere","Chemel, James L Cpa","8 W Cerritos Ave #54","Bridgeport","Gloucester","NJ","08014","856-636-
8749","856-264-4130","art@venere.org","http://www.chemeljameslcpa.com"
"Lenna","Paprocki","Feltz Printing Service","639 Main St","Anchorage","Anchorage","AK",99501,"907-385-
4412","907-921-2010","lpaprocki@hotmail.com","http://www.feltzprintingservice.com"
```

Other Types

- Object Oriented Data
- Graph Data
- Key-Value Data

Graph Data



Key-Value Data

- key-value (e.g. etcd used in K8s)
    - collections of unique keys mapped to value of arbitrary structure

```
0001 -> {some-json}
0002 -> {another-json-using-different-scheme}
0003 -> <even-xml-is-possible-but-would-be-extremely-suspicious>
```

Entity-relationship Diagram

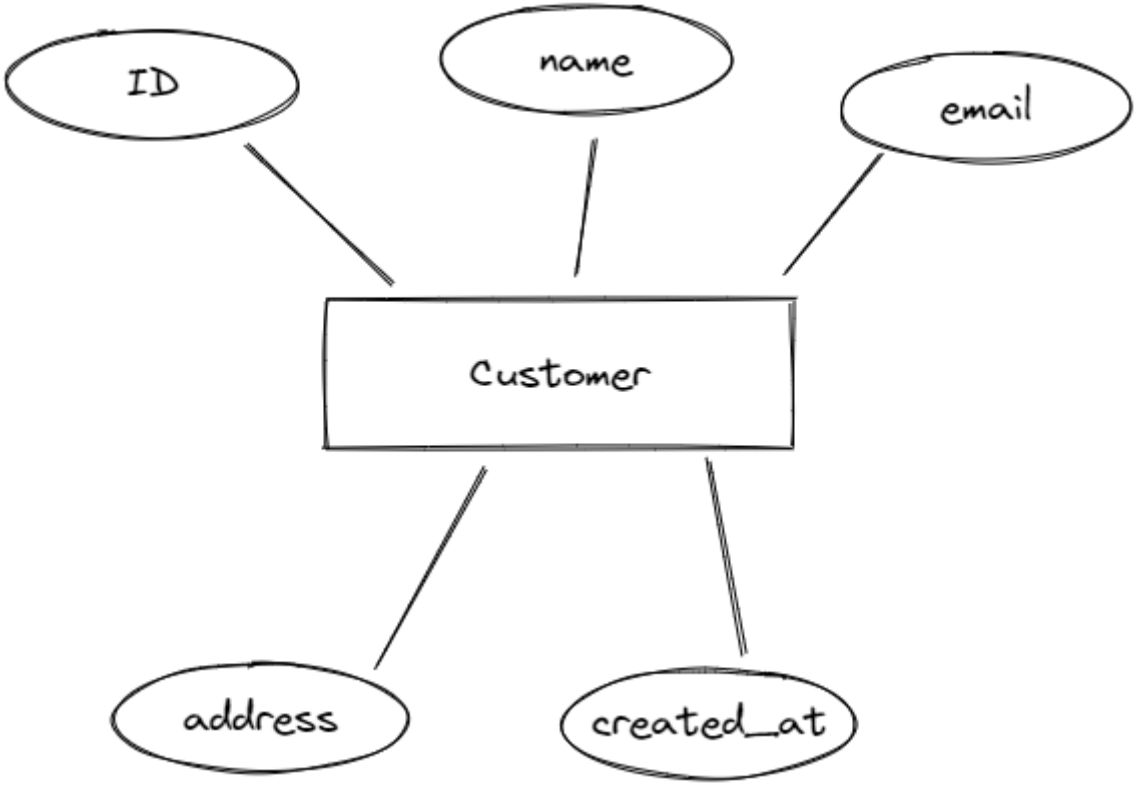
- Like UML, there are diagram types for modelling Data
- Entity Relationship Diagram is a flexible and universal model
- It helps in brainstorming

Elements of an ERD

- Entity
- Attribute - Connect to an Entity
- Relation - Connect two Entities

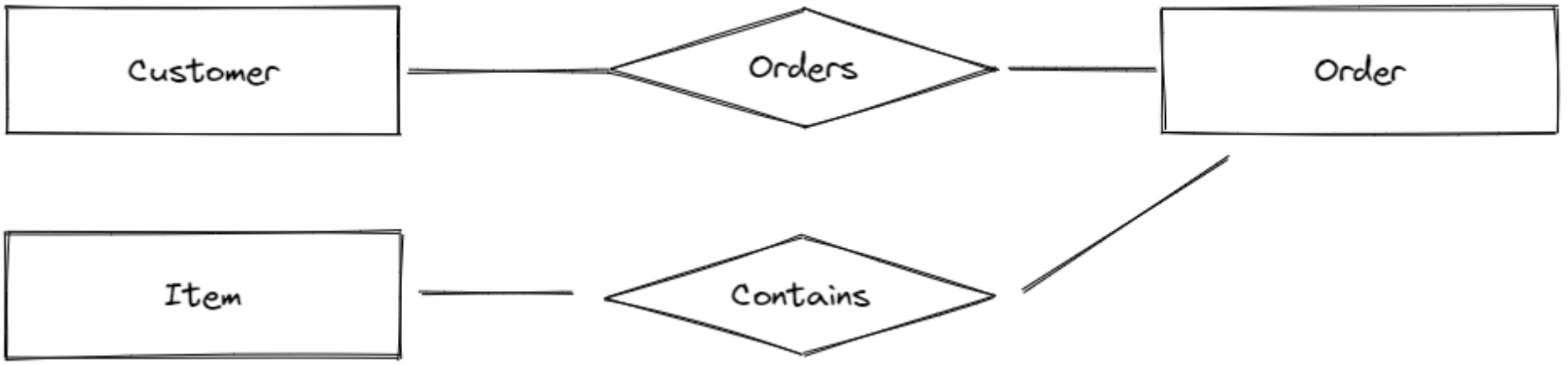
Entities and Attributes

- Attributes connect to an Entity
- If multiple Entities have simiar attributes, each has its own



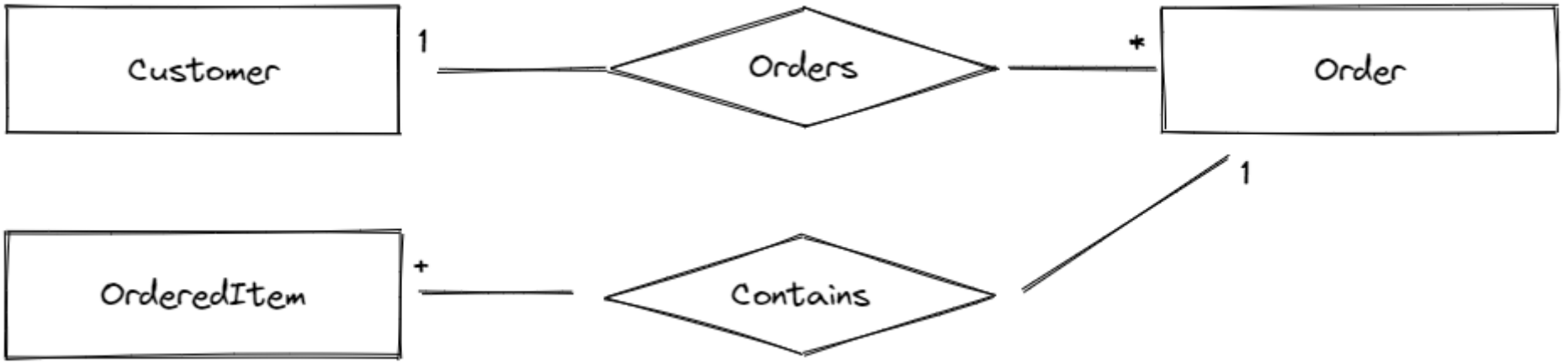
Relations in an ERD

- Entities are connected through relations
    - Relations are often written as actionable nouns
    - A `Customer` places an `Order`



Relations with cardinality

- A relationship can have cardinality
  - Indicate how many of one entity can be in that relationship



- They read like this:
    - "A Customer places zero or more orders"
    - "An Order is placed by exactly one Customer"

Exercise

In the `java-traineeship-exercises`, open the folder `module-04-data-erd`. In there you will find an `exercise.md` file, which contains the exercise. Please read it carefully and try to create an ERD in groups of 2-3 people.

Exercise - Discussions

- Can every group present their solution?
- Are there differences between groups?
    - Differences in level of detail?
    - Differences in chosen entities?
- Was there enough information in the exercise to make all decisions?

ER Diagram types

Depending on author, requirements and/or target audience a different information granularity level is desired.

ER Diagrams can be split in following types:

- Conceptual data model

- Logical data model
- Physical data model

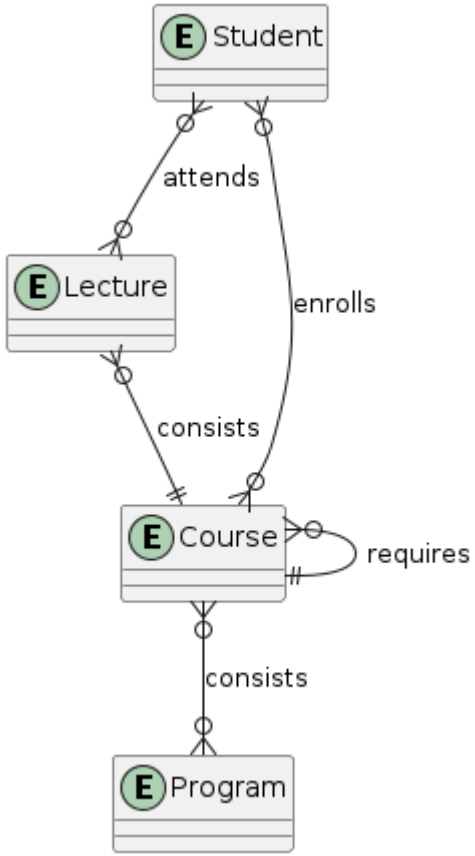
Let’s discuss the differences between those data models/ granularity levels and check the corresponding ER diagrams (based on previous exercise)

[1]

[1]

Conceptual

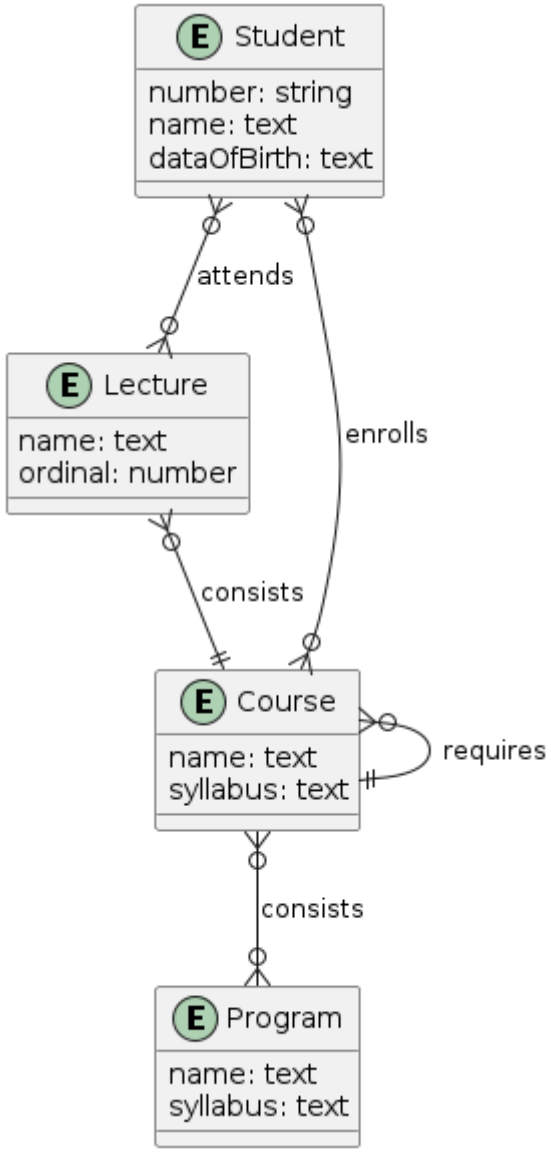
- very high-level
- business-oriented
- database-independent
- support documenting data architecture for an organization
- containts
    - entities
    - relationships



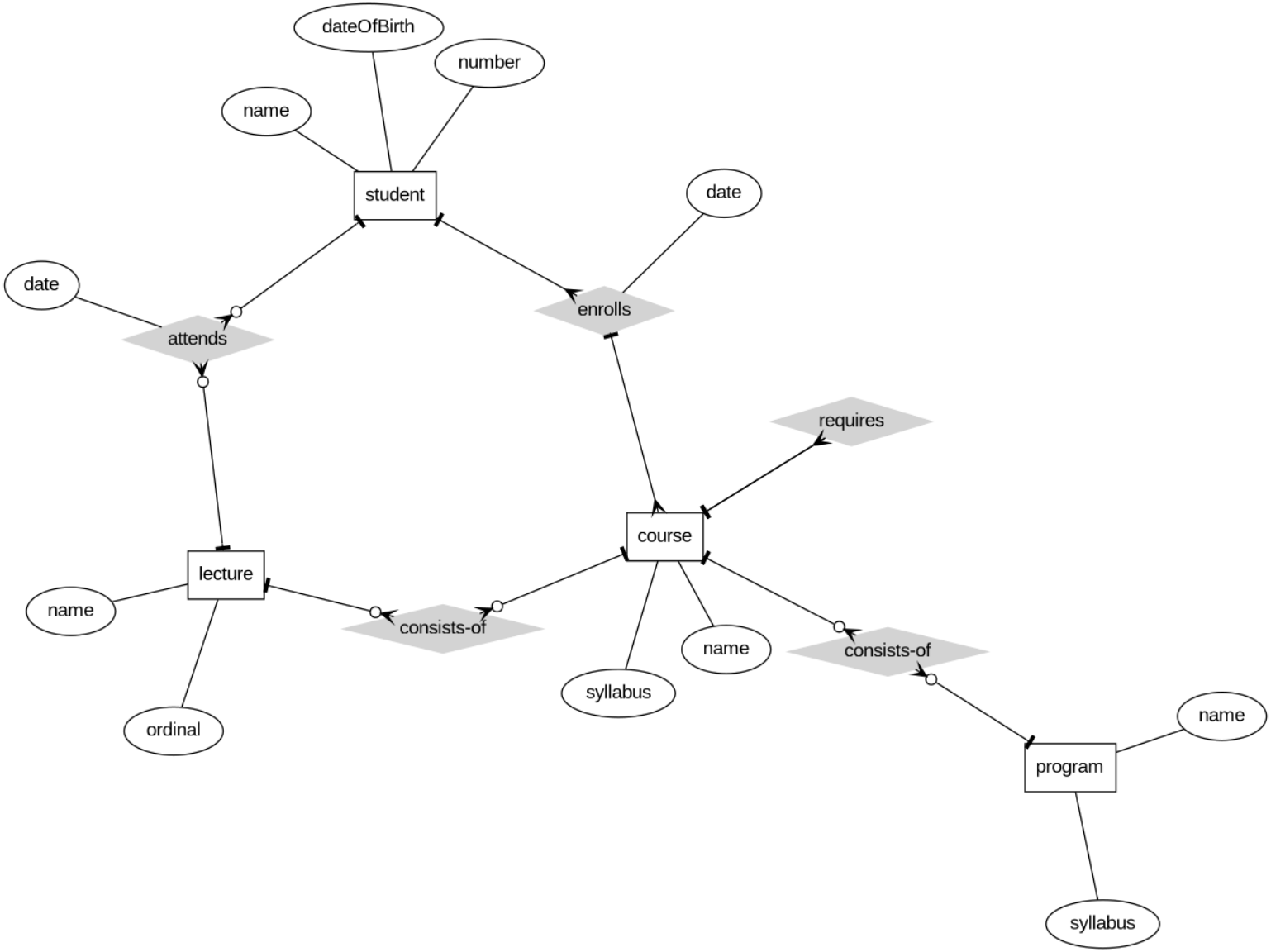
Logical

- business-oriented
- database-independent
- contains
    - enitities

- relationships
- attributes
- attribute types (optional)



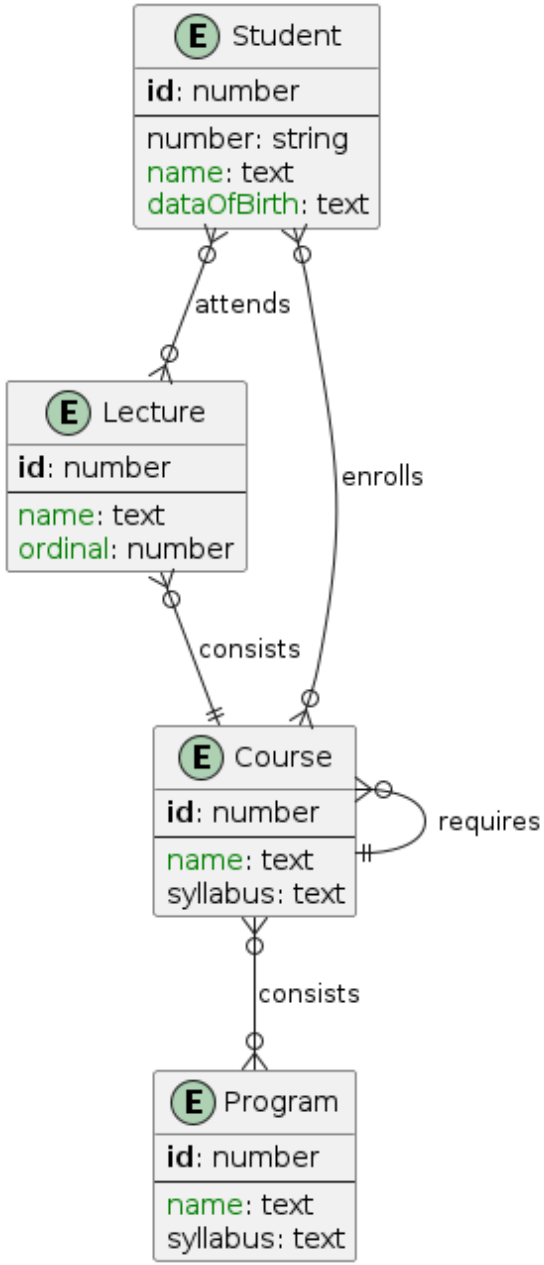
Note: alternative made with Graphviz



Entity Relation Diagram

Physical

- dev-oriented
- database-specific
- contains
    - enitities
    - relationships
    - attributes
    - attribute types
    - primary keys
    - foreign keys
    - (unique) indexes


Student
id: number
number: string
name: text
dataOfBirth: text
attends
enrolls
Lecture
id: number
name: text
ordinal: number
consists
Course
id: number
name: text
syllabus: text
requires
consists
Program
id: number
name: text
syllabus: text

footnote:"PlantUML ER diagram template"[https://gist.github.com/QuantumGhost/0955a45383a0b6c0bc24f9654b3cb561]

Relational Databases and SQL

- Relational databases store data in tables
- It was the first major type of database after "flat files"
- Years of research and development have made them into powerhouses
- Indexing strategies, query languages even aggregations and statistics
- Most applications still use this type of database

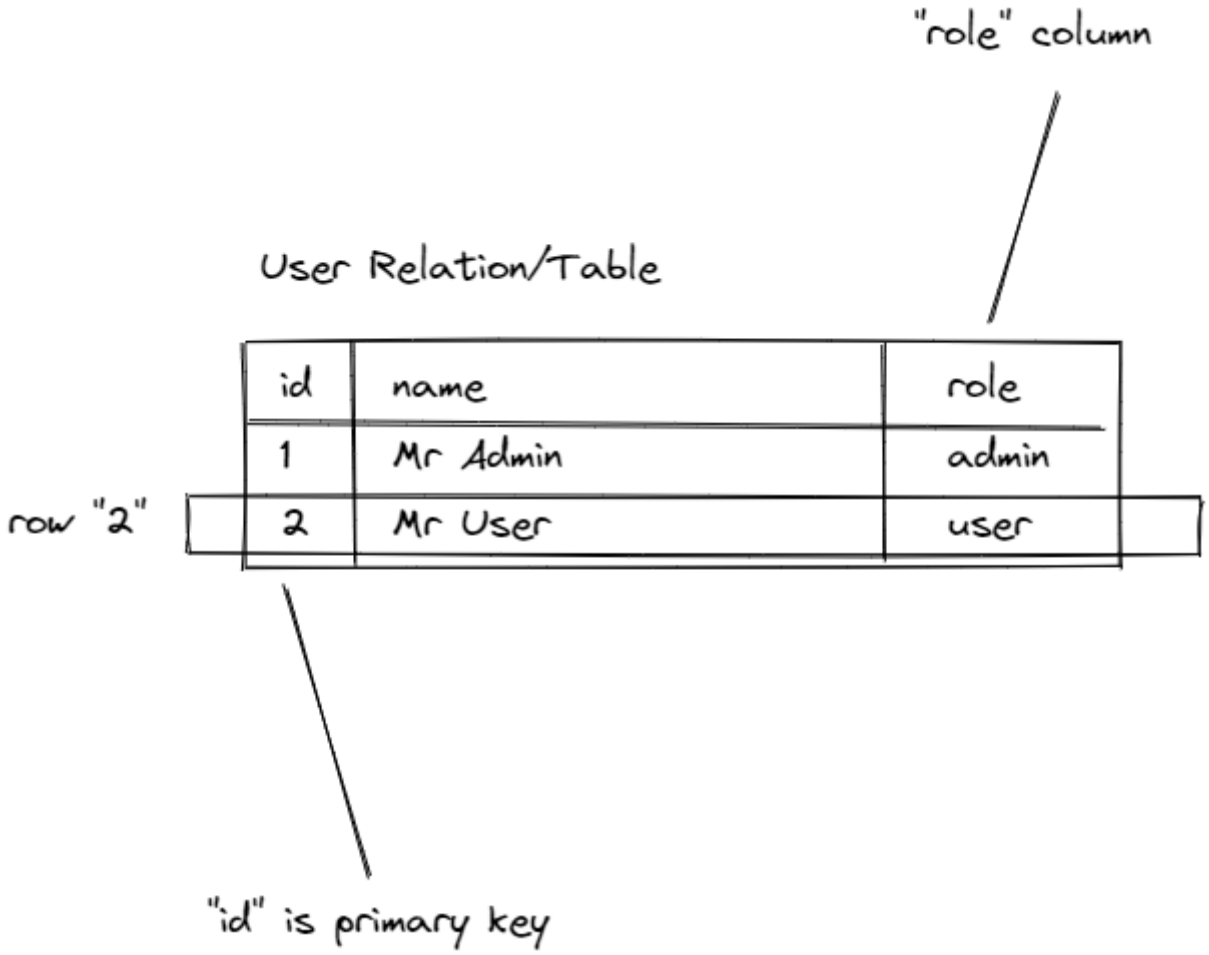
RDMBs

- There are many different databases in the Relational domain
    - Oracle
    - MSSQL
    - Postgres
    - MySQL
    - MariaDB
    - CockroachDB
    - SQLite
    - …

Terminology of RDBMs

- Relations - aka Tables, describe the structure of a row/record
- Row/Record - a set of values that together form one entry in a table/relation
- Attribute - aka Columns, give a name and data type to a part of table
- Data Type - the type of data that fits in a column. Each type has specific optimizations for storage and querying
    - Text data
    - Numerical data
    - Date/Time
    - Currency
    - Boolean
- Keys - values used to lookup a specific row or rows in a table
    - Primary key - the attribute or column that uniquely identifies any row in a table
- Query - a statement to specify what data to retrieve from the database

Terminology



The exciting part: SQL

- Storing data is only a small fraction of its actual use
    - Most databases are queried more often than written to

    - That makes querying the important part of any database
- Enter: Structured Query Language

SQL

- Created in the 70s
- Structured Query Language
- Used to "talk to a relational database"
- Originally pronounced Sequel, but not anymore because of trademarks
- ANSI standard

SQL usage

- Modifying the database structure (tables)
- Querying data
- Adding, updating and deleting data

Modifying database structure

- Also called Data Definition Language (DDL)
- Covered in day 2

SQL syntax rules

- Case-insensitive
- Comments
- Semicolons

Casing rules

- SQL is case insensitive
- Keywords can be any case, or mixed
- And also tables, columns, etc
- Different people, different preferences

Comments

- Useful to document choices in your queries
- A one-line comment starts with two dashes (--)
- SELECT * -- Everything!
  FROM table

Multiline comments

- Just as in Java

- SELECT * /* Everything
  multiline
  */
  FROM table

Semicolons

- You may end an SQL statement with a semicolon
- Mandatory when you have a script with multiple statements

Querying data

- You can query data using the "SELECT" statement
- SELECT * FROM my_table
- The asterisk means: "Gimme all columns"
- The FROM clause defines the table to query data from
- Probably the most basic query

Querying data with filters

- You can filter the retrieved data using the "WHERE" clause
- SELECT * from my_table WHERE id='123'

More advanced filters

- You can use "AND" and "OR"
- And many more operators
- SELECT * from people WHERE first_name='John' AND last_name='Doe'
- Note: Casing matters here!

Filtering null

- NULL values require special syntax
- SELECT * from people WHERE first_name IS NULL
- SELECT * from people WHERE first_name IS NOT NULL

Filtering on multiple values

- SELECT * from people WHERE first_name IN('John', 'Jane')
- Returns all people named John or Jane

Filtering ranges

- You could manually write a range query, like this
- SELECT * FROM PEOPLE WHERE age >= 10 AND age <= 20
- Or use BETWEEN

- SELECT * FROM PEOPLE WHERE age BETWEEN 10 AND 20

Defining the columns in the result

- Part of the SELECT clause
- Just name the columns, separated with commas
- Or specify "*" to get all columns
- SELECT first_name, last_name FROM people

Limiting the result set

- Use LIMIT to retrieve at most N rows
- SELECT * FROM table LIMIT 10
- Note: Some databases don’t support LIMIT

Limiting the result set, part 2

- Use OFFSET to start with ROW N
- SELECT * FROM table OFFSET 10 LIMIT 10
- Gives you row 11 until 20 (included)

Searching in text

- Using LIKE

- SELECT * FROM people WHERE last_name LIKE '%o%'
- Gives all people with an "o" anywhere in the name
- This is still case sensitive!

Case-insensitive search

- Using LOWER, or UPPER
- SELECT * FROM people WHERE LOWER(last_name) LIKE '%o%'
- Gives all people with an "o" or "O" anywhere in the name

Applying operators to the result

- Most operators can be used anywhere
- SELECT UPPER(first_name) FROM people

Naming the resulting columns

- Useful when you need to read the column from your (Java) code
- SELECT UPPER(first_name) FROM people gives you a useless column name
- Name the column using AS
- SELECT UPPER(first_name) AS upper_first_name FROM people

Sorting data

- Using the ORDER BY clause
- Sort on one or more colums
- SELECT *
  FROM people
  ORDER BY last_name, first_name
- Default in ascending order
- Ordering is case-sensitive (a < A)

Sorting data descending

- Using the DESC operator in the ORDER BY clause
- SELECT *
  FROM people
  ORDER BY last_name DESC, first_name DESC

Sorting data ascending, explicitly

- Using the ASC operator in the ORDER BY clause
- SELECT *
  FROM people
  ORDER BY last_name ASC, first_name DESC

Sorting data case-insensitive

- Using the LOWER function
- SELECT *
  FROM people
  ORDER BY LOWER(last_name), LOWER(first_name)

Retrieving to only unique values

- DISTINCT makes every result row unique
- SELECT DISTINCT last_name
  FROM people
- Gives back all distinct (unique) last names

DISTINCT works on the whole row

- SELECT DISTINCT *
  FROM people
- Doesn’t work, because the whole row is never unique (there is always a unique key for every row)

Grouping data

- GROUP BY can be used to group rows with the same values in one or more columns
- Like DISTINCT, but more advanced and useful

- For example: Give me the number of people, grouped by their age
- SELECT age, COUNT(*)
  FROM people
  GROUP BY age

Grouping on multiple columns

- You can also group on multiple columns
- SELECT age, gender, COUNT(*)
  FROM people
  GROUP BY age, gender

Aggregate functions

- Not just COUNT
- Also MIN, MAX, SUM, AVG
- SELECT age, AVG(number_of_children)
  FROM people
  GROUP BY age
- Gives back the average number of children per age

Aggregate functions outside GROUP BY

- This is possible :)

- Function aggregates over the whole result set
- SELECT COUNT(*) FROM people
- Gives you the total number of people

Group by basic rules

- Every field in the result has to be
    - part of the GROUP BY clause
    - in an aggregate function
- This is invalid, as c is neither
- SELECT a, SUM(b), c FROM demo GROUP BY a

Execution order of a query

- First determine the table (FROM)
- Then filter using WHERE
- Then group using GROUP BY
- Then filter using HAVING
- Then determine the result using SELECT
- Then order using ORDER BY
- Then limit using LIMIT and OFFSET

Wut? What’s HAVING?

- HAVING is similar to WHERE
- But it filters on the result from GROUP BY
- Basically a second filter step

HAVING example

- Let’s say we want the number of people with a given age, but only when there are more than 10
- WHEN cannot do this, as it filters on the ungrouped data
- HAVING to the rescue!
- SELECT age, COUNT**() as number_of_people**
  FROM people
  GROUP BY age HAVING COUNT() > 10 -- Need to repeat the COUNT(*), cannot use the "as"

Joining tables

- You can join multiple tables in a query
- Part of SQL day 2

Advanced query usage

- Subqueries (A SELECT in another SELECT)

- UNION (combine 2 result sets)
- EXCEPT (only take the rows from a result set that are not in the second result set)
- Lots more, often specific to a database

Adding data

- You can add data using the "INSERT INTO" statement
- Adds one or more rows to a table
- INSERT INTO people
  VALUES(5, 'John', 'the Teacher')

Adding multiple rows

- You can also add multiple rows at once
- INSERT INTO people
  VALUES
  (5, 'John', 'the Teacher')
  (6, 'Jane', 'the other Teacher')

Specifying columns to insert

- Not every column needs to be specified
- Some have default values

- Or accept NULL
- INSERT INTO people (first_name, last_name)
  VALUES ('John', 'the Teacher') -- Skips the key

Copying rows from another table

- Using INSERT INTO SELECT
- Adds new rows to existing table
- INSERT INTO people2
  SELECT * FROM people
  WHERE age < 10 -- Only copy people younger than 10

Copying a table

- Using SELECT INTO
- Creates the table
- Can create tables across databases
- SELECT * INTO people2
  FROM people
  WHERE age < 10 -- Only copy people younger than 10

Updating data

- Manipulating data using UPDATE

- This updates a column in EVERY row!
- UPDATE people
  SET first_name='John' -- Never do this!
- So never do this!

Updating one row

- So we need to limit the update scope to one row
- UPDATE people
  SET first_name='John'
  WHERE key = 123
- Only updates the first_name to "John" where key is 123

Updating multiple columns

- UPDATE people
  SET first_name='John', last_name='Doe', full_name='John Doe'
  WHERE key = 123
- Only updates the first_name to "John" where key is 123

Dynamic updates

- You can calculate a column value from other column values
- UPDATE people

SET full_name=first_name || ' ' || last_name

- Sets the full_name to be first_name and last_name, separated with a space.
- Note that || is used to concatenate strings

Deleting data

- Finally, how to delete data?
- We use the DELETE statement
- DELETE FROM people
  WHERE key = 123
- Deletes the row with key 123, if it exists, otherwise delete nothing

Deleting everything

- As with UPDATE, DELETE does not require a WHERE
- DELETE FROM people -- This is valid
- This deletes all rows from the table
- Don’t do this!

[1] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model