III. Back-End Technologies

1 Basic Technologies

- Web-Server
- Server-side Website Programming

2 Using a Web Application Framework

- Processing Requests
- Templates
- Model-View-Controller

3 Accessing Databases

- Foundations of relational databases
- Introduction to SQL
- Object-relational mapper

3 Accessing Databases

Relational databases, SQL, Object-relational mapper

III.3

Topics

Accessing Databases

- characteristics, terminology and architecture of databases
- concepts of the database design
- relational data model
- working with a database: SQL – structured query language

III.3

Fundamental Concepts

System "Database"

- Objectives of an efficient organisation of data persistency
  - data independence
    - independence from application program
    - independence of logical and physical data organization
    - physical data independence
  - no redundancy
  - data integrity
  - data protection
  - data security

III.3

Fundamental Concepts

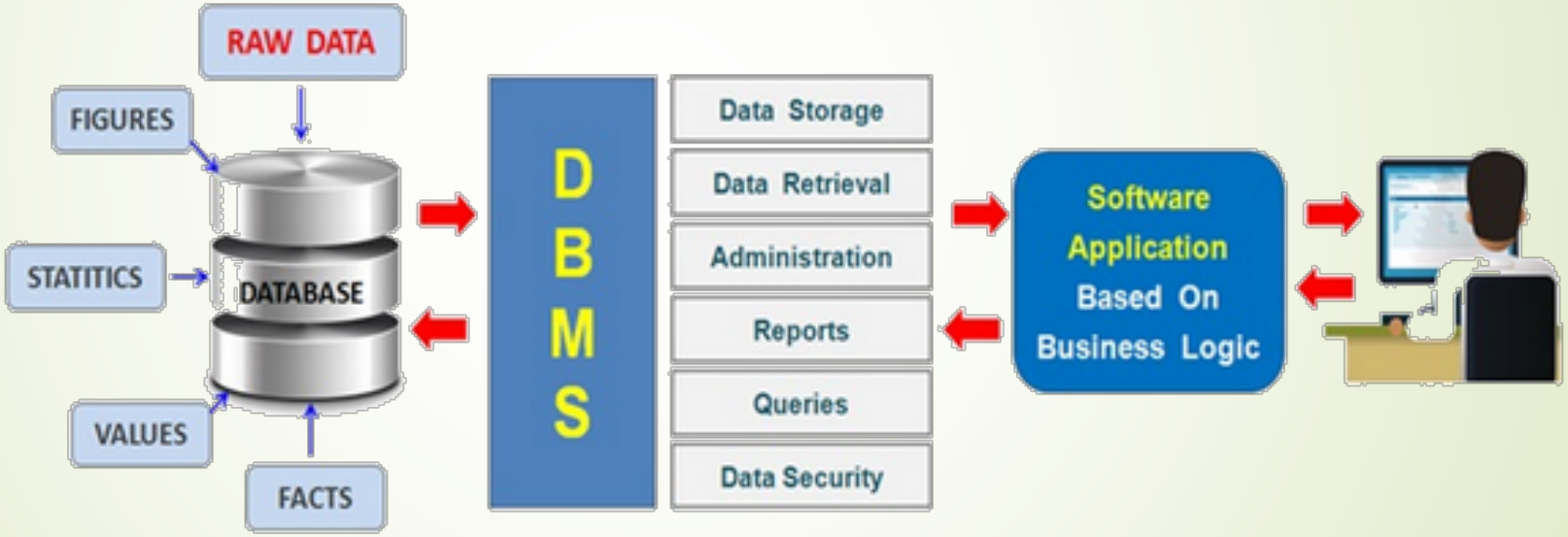
Definitions

- data
  - units processable by computing machines
- database
  - an integrated, persistent dataset including all relevant information about the represented information, which is available to a group of users as a unique entity
- database management system (DBMS)
  - the totality of all programs for the creation, administration and manipulation of a database

III.3

Fundamental Concepts

Definitions



III.3

Fundamental Concepts

Definitions

- relation $R(A_1, A_2, ..., A_n)$
  - a named set of n-tuples,
    where an n-tuple is an arrangement of n atomic, i.e. not further decomposable,
    attributes $A_1, A_2, ..., A_n$
- table
  - multiset of tuples
- (Data) model / scheme
  - descriptive reproduction of an existing situation / object

III.3

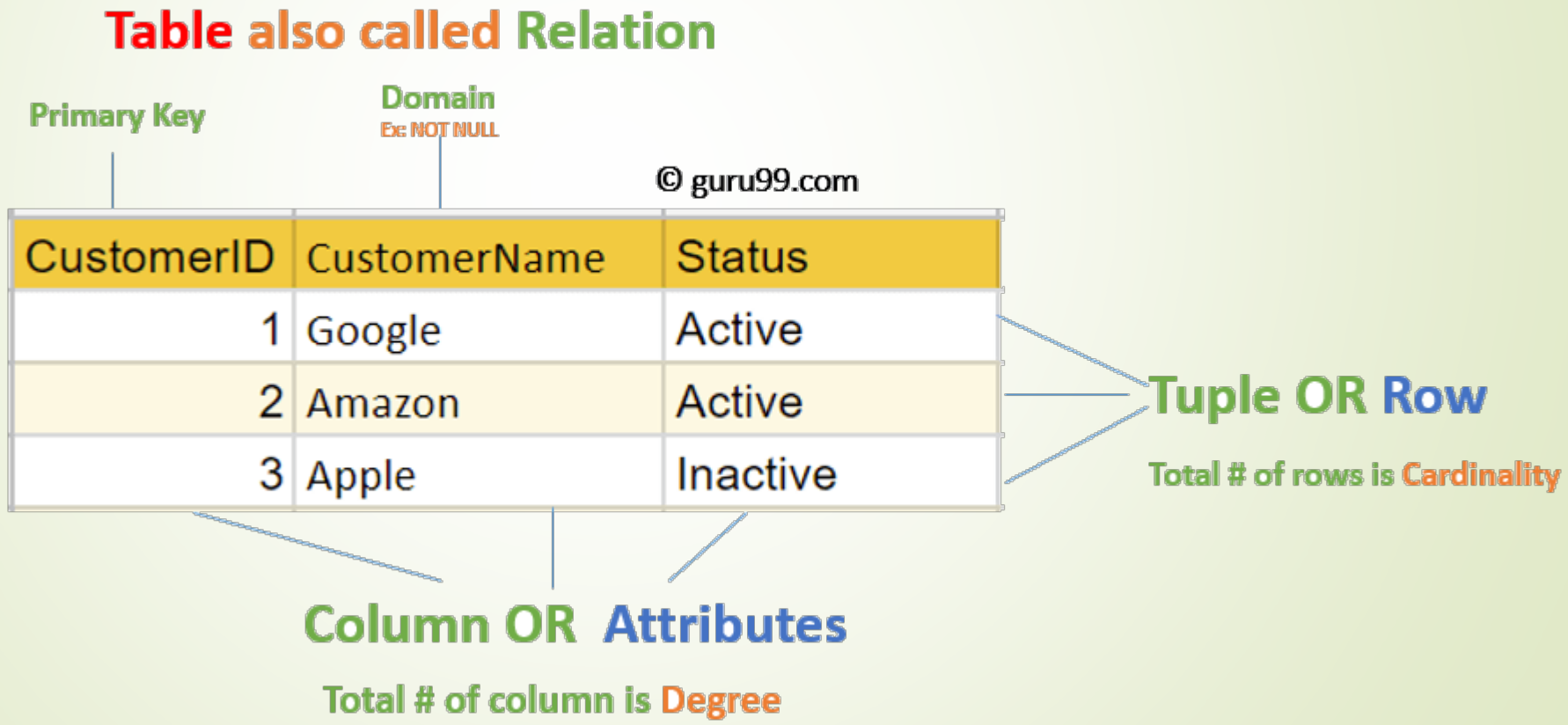
Fundamental Concepts

Definitions

- attribute
  - specific dimension of the tuple
  - attributes are the properties which define a relation
- attribute domain
  - every attribute has some pre-defined value and scope which is known as attribute domain
- tables
  - relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

Fundamental Concepts

Definitions



III.3

Fundamental Concepts

Definitions

- tuple
  - a single row of a table, which contains a single record
- column
  - represents the set of values for a specific attribute
- degree
  - total number of attributes which in the relation is called the degree of the relation
- cardinality
  - total number of rows present in the table

III.3

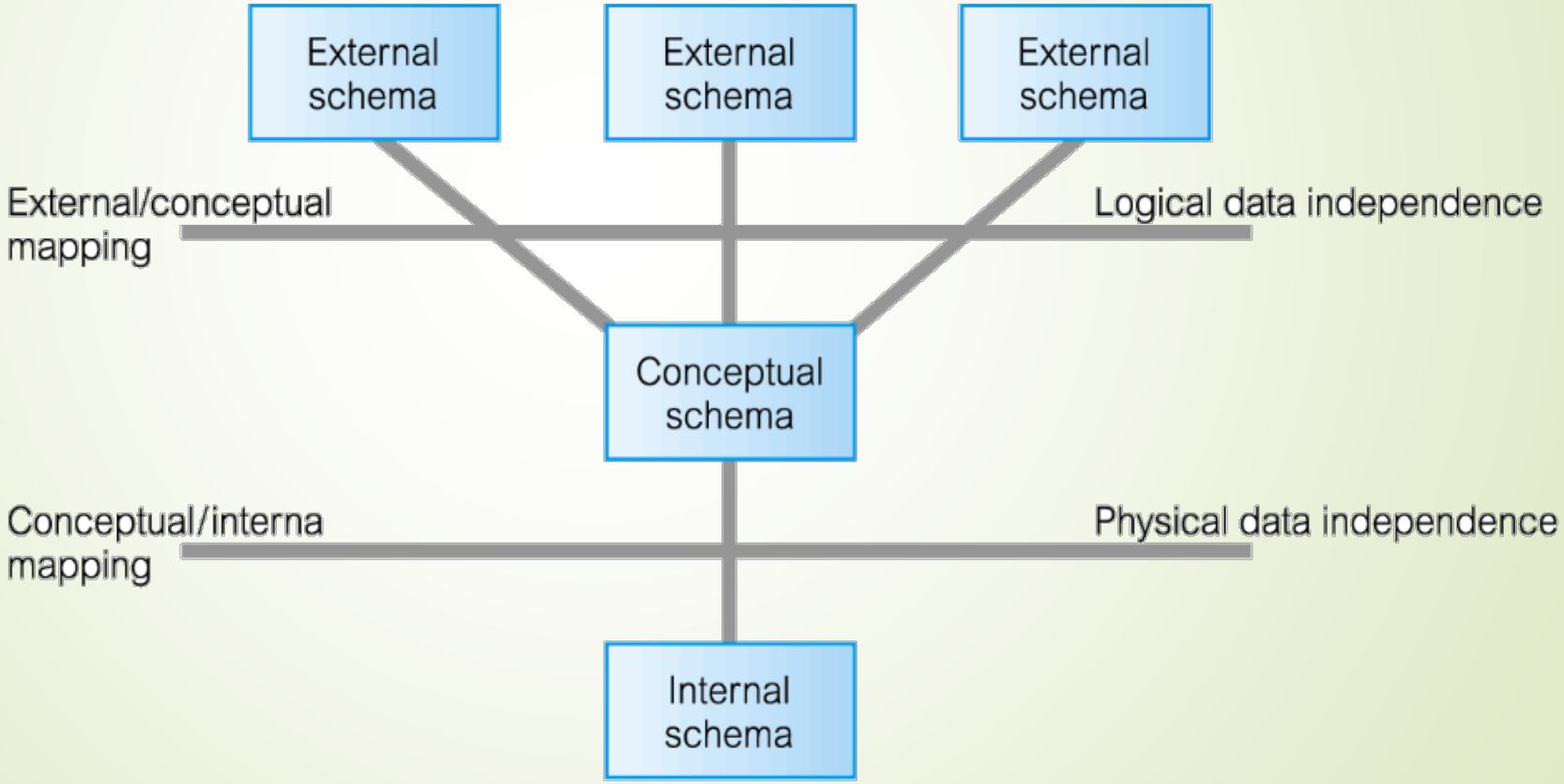
Fundamental Concepts

Definitions

- relation schema
  - a relation schema represents the name of the relation with its attributes
- relation instance
  - a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
- relation key
  - every row has one, two or multiple attributes, which is called relation key

Fundamental Concepts

ANSI-Sparc-Architecture



III.3

Fundamental Concepts

ANSI-Sparc-Architecture

- Conceptual scheme
  - model, which describes the relevant section of reality (also mini-world, discourse area) in structure and content
- Logical schema
  - model, which was derived paradigm-specifically from a conceptual schema, e.g. according to the relation approach
- physical schema
  - an implementation-specific model, which was derived from a logical schema

III.3

Database Design

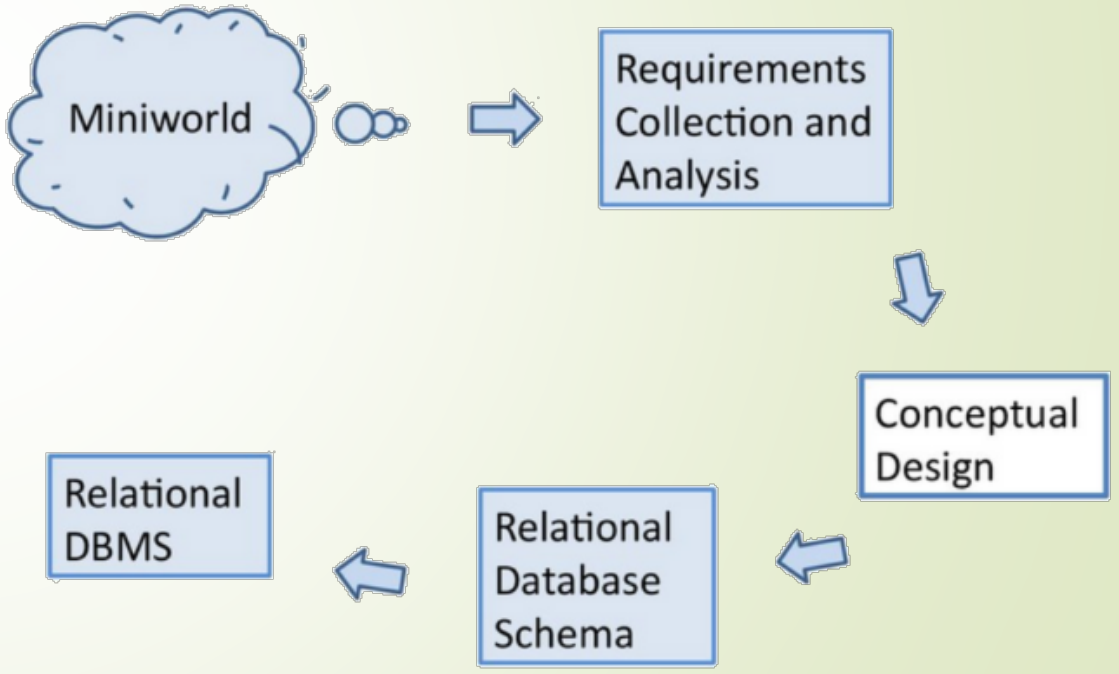
Database design

- objectives
  - modeling of an application-oriented section of the real world (mini-world)
  - design of the logical DB structure (DB design)
- constraints
  - completeness
  - correctness
  - minimality
  - modifiability

Database Design

Database design

- Display elements + rules
  - objects (entities) and relationships
  - classes of objects / relationships
  - properties (attributes)



III.3

Database Design

Abstraction concepts

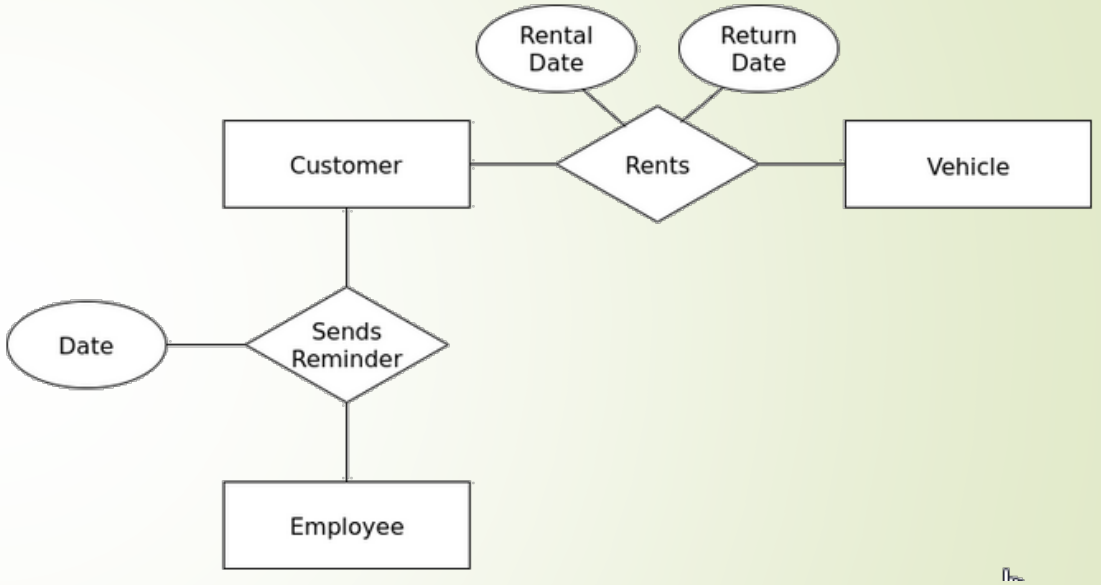
- information and data models are based on
  - classification
    - combines objects (entities, instances) with common properties to a new (set) object (entity set, class, object type)
    - instances/objects of a class are subject to the same structure (attributes), same integrity conditions, same operations
  - aggregation
    - aggregation of potentially different sub-objects (components) to a new object
  - generalization
    - subset relations between elements of different classes
    - essential: inheritance of properties to subsets

III.3

Entity-Relationship-Model

Concepts

- elements
  - entity sets
  - relationship sets
  - attributes
  - value ranges
  - primary keys



- supports the abstraction concepts of classification and aggregation
- graphical representation through diagrams

Entity-Relationship-Model

Entity sets

- entity
  - represents abstract or physical object of the real world
  - similar entities (i.e. entities with common properties) are grouped into entity sets (object types) (classification), i.e. entities are elements of a (homogeneous) set
- database contains finitely many entity sets, which do not necessarily have to be disjoint

III.3

Entity-Relationship-Model

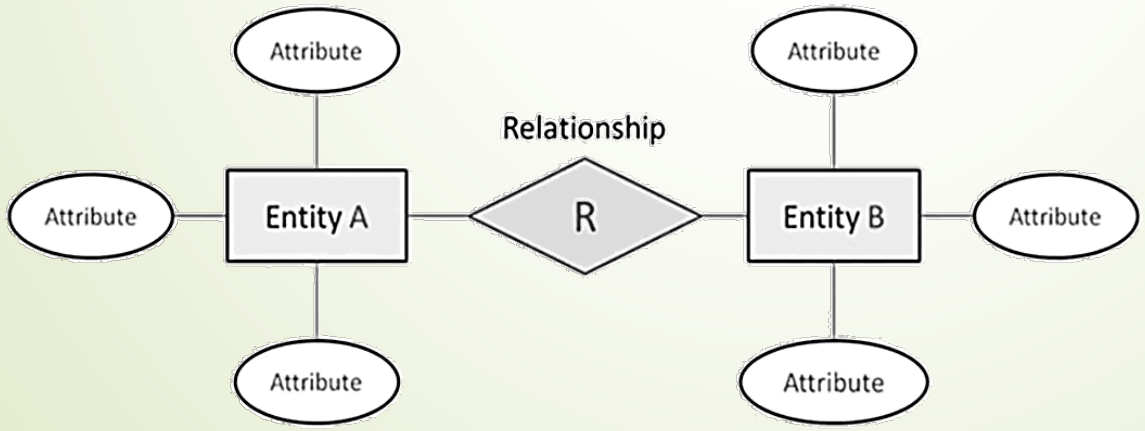
Keys

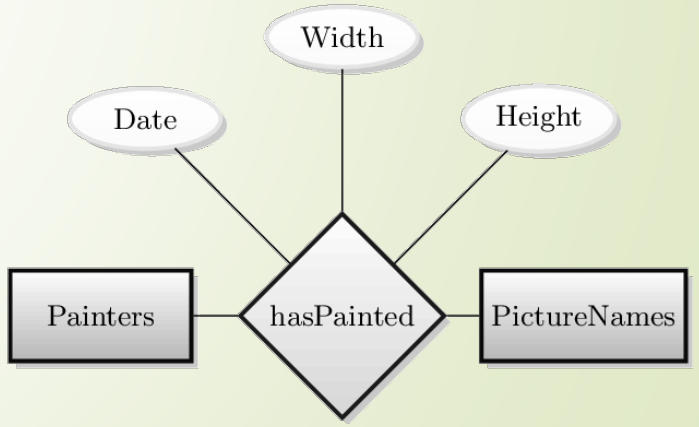
- key candidate or key for short (key)
  - single-valued attribute or attribute combination that uniquely identifies each entity of an entity set (no null values!)
- primary key
  - specific key candidate

Entity-Relationship-Model

Relationships

- relationship set
  - summary of similar relationships between entities that belong to the same entity sets
  - corresponds to a mathematical relation between n entity sets
  - may have attributes

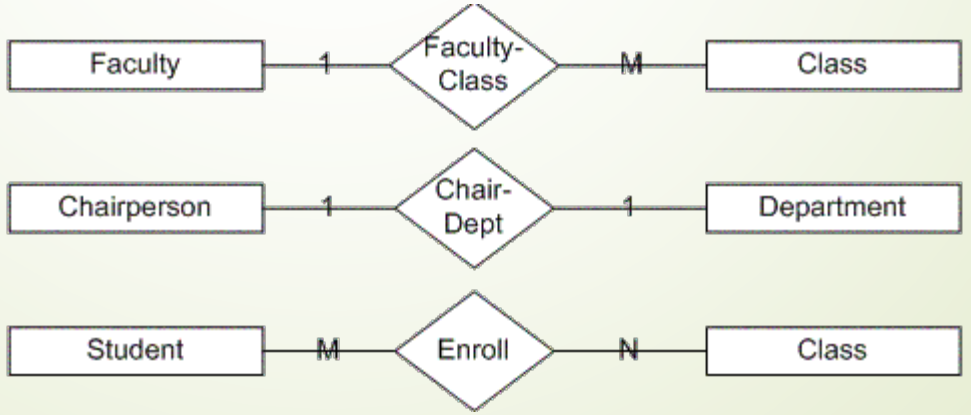




Entity-Relationship-Model

Cardinality of relations

- definition of important structural integrity conditions
- different mapping types for binary relationship between entity sets Ei and Ej
  - 1:1 one-to-one function (injective mapping)
  - n:1 mathematical function (functional mapping)
  - n:m mathematical relation (complex mapping)



III.3

Relational Data Model

Basic Concept

- data management structure: relation (table)
  - single data structure
  - all information represented exclusively by values
    - n not necessarily different value ranges $D1,\ldots,Dn$ (schema of relation)
    - values are atomic
    - relation defined by $R \subseteq D1 \times D2 \times \ldots \times Dn$
    - specification by sequence of identifier-value range pairs

III.3

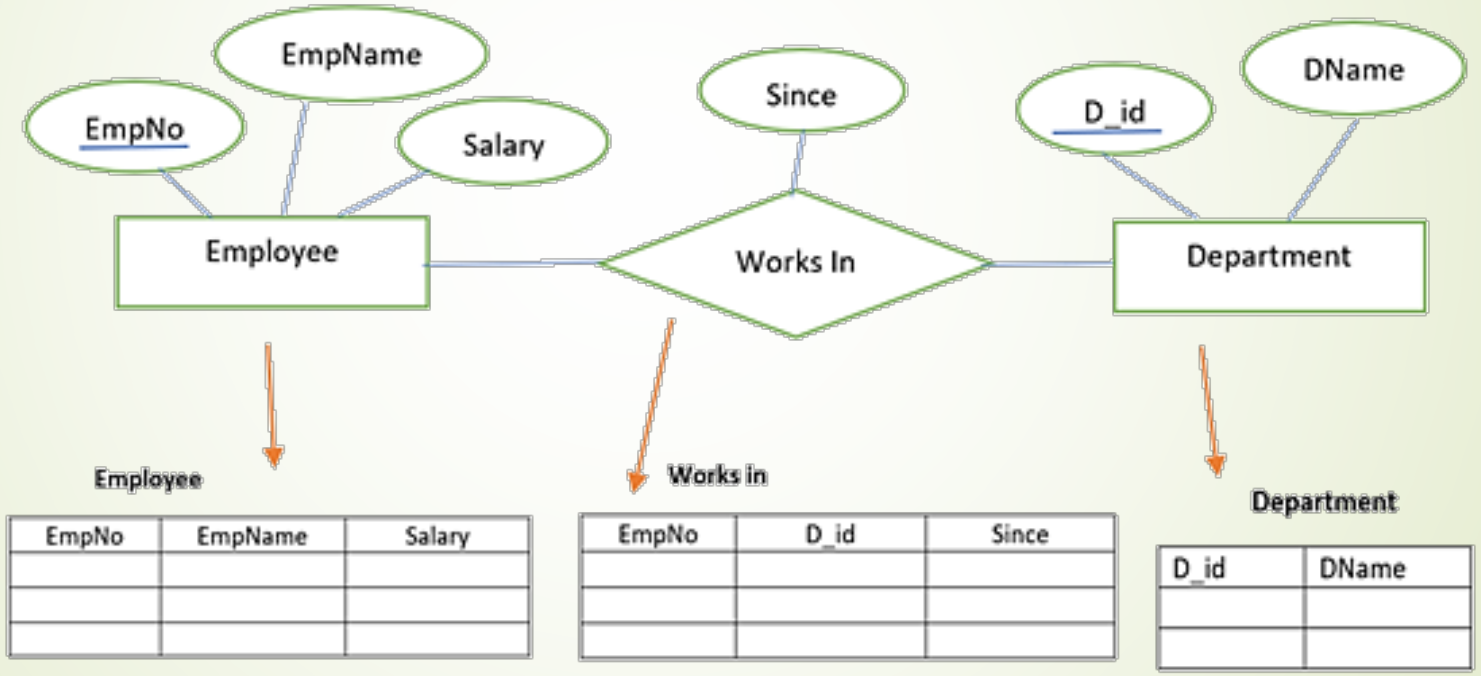
Relational Data Model

Basic Concept

- basic
  - relation for entity sets as for relationship sets
- supplementary
  - attributes with definition ranges
  - primary key
- n-column table for representation
  - parameters
    - number of columns = degree of relation
    - number of records / tuples = cardinality

Relational Data Model

Basic Concept



III.3

Structured Query Language

SQL

- unified language for all DB tasks
  - simple query capabilities for the casual user
  - powerful language constructs for the more educated user
- structured language based on English keywords
- basic building block

```
SELECT    attributes
FROM      relations
WHERE     conditions
```

  - input relations (FROM) are mapped into attributes of a result table (SELECT) under evaluation of conditions (WHERE)

Structured Query Language

SQL – Multi-Table Queries

- tables that share information about a single entity need to have a primary key that identifies that entity uniquely
- using the JOIN clause in a query, row data are combined across two separate tables using this unique key
  - INNER JOIN returns data that meets certain join conditions
  - LEFT/RIGHT JOIN is a join between tables that returns all records from the left/right table even if there is no match with the right/left table
  - FULL JOIN returns all records from two tables, even if there is no match with the other table

Structured Query Language

SQL – Multi-Table Queries

- [INNER] JOIN — no lossless relation, normal case
- LEFT JOIN — the left relation is lossless
- RIGHT JOIN — the right relation is lossless
- [FULL] OUTER JOIN — both relations lossless

L

A	B
1	2
2	3

R

B	C
3	4
4	5

inner

A	B	C
2	3	4

left

A	B	C
1	2	⊥
2	3	4

right

A	B	C
2	3	4
⊥	4	5

full

A	B	C
1	2	⊥
2	3	4
⊥	4	5

III.3

Django Models

Django's storage technology

- relational
  i.e. out-of-the-box it can connect to SQLite, Postgres, MySQL or Oracle
- Django model
  - is designed to map directly to a relational database table
  - instances of a Django model are stored as rows of a relational table named after the model
  - is usually defined in an application's models.py file
  - is implemented as subclasses of `django.db.models.Model`

III.3

Django Models

Defining a model

- simple table

id	name	age	city	state
12	Hans	23	Ingolstadt	Germany
22	John	....		

```
class People(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField
    city = models.CharField(max_length=30)
    state = models.CharField(max_length=2)
```

III.3

Django Models

Defining a model

- fields
  - can have an arbitrary number of fields, of any type
  - each one represents a column of data that we want to store in one of our database tables
  - field types are assigned using specific classes
    - determine the type of record that is used to store the data in the database
    - validation criteria to be used when values are received from an HTML form

III.3

Django Models

Defining a model

- fields
  - common arguments
    - can be used when declaring many/most of the different field types
    - help text, default, choices, ...
    - primary_key
      sets the current field as the primary key for the model
    - see full list of field options
  - field types
    - CharField, TextField, IntegerField, DateField, AutoField, ... (full list)
    - relationship specifications

Django Models

Defining a model

- metadata
  - model metadata is “anything that’s not a field”
    - ordering options (ordering)
    - database table name (db_table)
  - defined as inner class of the model class

```
class People(models.Model):

    class Meta:
        ordering = ['name', '-city']
```

III.3

Django Models

Defining a model

- methods
  - possible
  - minimally, in every model define the standard Python class method __str__() to return a human-readable string for each object

- attributes
  - Manager
    - the interface through which database query operations are provided to Django models and is used to retrieve the instances from the database
    - the default name is objects
    - only accessible via model classes, not the model instances

Django Models

Using a model

- creating and modifying records

```
# Create a new record using the model's constructor.
record = People(name="Emilia", ...)

# Save the object into the database.
record.save()

# Changing record by modifying a field.
record.city = "Munich"
record.save()

# Deleting a record.
record.delete()
```

Django Models

Using a model

- get a record

```
# Get a single record.
p = People.objects.get(id=12)
```

- searching for records
  - search for records that match certain criteria using the model's objects attribute

```
# Get all records.
queryset = People.objects.all()

# Filter records.
queryset = People.objects.filter(city="Ingolstadt")
```

Django Models

Using a model

- lookups
  - Field lookups are used to specify an SQL WHERE clause
  - SQL

```
SELECT * FROM people WHERE age <= 30;
```

  - Django

```
People.objects.get(age__lte=30
```

III.3

Django Models

Using a model

- queryset
  - a collection of data from a database, built up as a list of objects
  - makes it easier to get the data you actually need, by allowing you to filter and order the data at an early stage
  - can be evaluated in the following ways
    - iteration `for entity in querset`
    - slicing
    - pickle

Django Models

Using a model

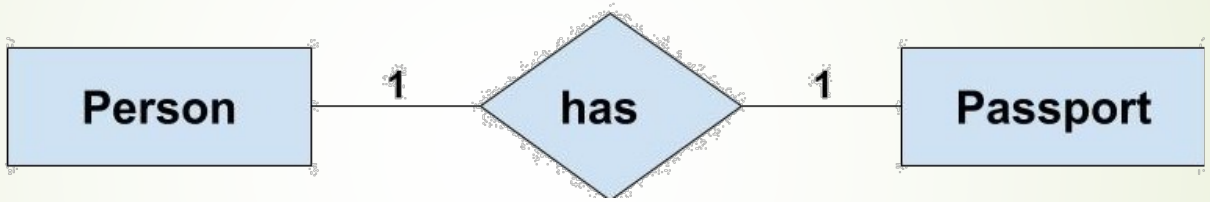
- formulating queries
  - Q() objects
    - represents an SQL condition that can be used in database-related operations
    - used to define and reuse conditions, and to combine them using operators such as | (OR) and & (AND)

```
queryset = People.objects.filter(
    Q(city__startswith='In') | Q(city__startswith='Mu'))
```

Django Models

Relationship Models

- One-to-One



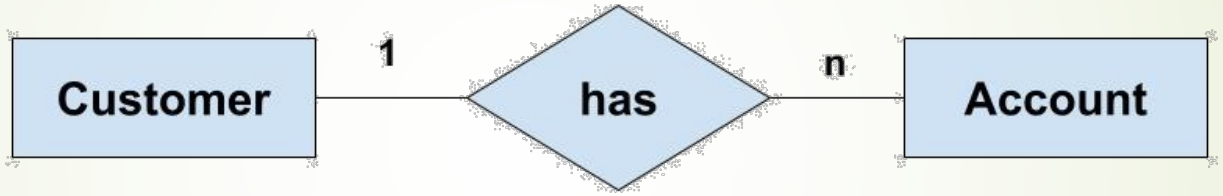

```
# The model Person
class Person(models.Model):
  name = models.CharField(max_length=50)

  def __str__(self):
        return str(self.name)
```

```
# The model Passport
class Passport(models.Model):
  person = models.OneToOneField(
        Person,
        on_delete=models.CASCADE,
        primary_key=True,
  )
  issuedate = models.DateField
```

Django Models

Relationship Models

- One-to-Many




```
# The model Customer
class Customer(models.Model):
  name = models.CharField(max_length=50)

  def __str__(self):
        return str(self.name)
```

```
# The model Account
class Account(models.Model):
  customer = models.ForeignKey(
        Customer,
        on_delete=models.CASCADE
  )
  issuedate = models.DateField
```

Django Models

Relationship Models

- Many-to-Many


```
# The model Customer
class Customer(models.Model):
  name = models.CharField(max_length=50)

  def __str__(self):
        return str(self.name)
```

```
# The model Account
class Account(models.Model):
  customer =
    models.ManyToManyField(Customer)
  issuedate = models.DateField
```