Logical database design

- The learning objectives for this week are:
 - Knowing what is the objective logical database design
 - Knowing how to derive relations from simple ER diagrams
 - Knowing how to determine natural primary keys and foreign keys

Logical database design

- The typical main phases in a database design process are:
 - i. Conceptual database design
 - ii. Logical database design
 - iii. Physical database design
- Each phase from top to bottom adds more detail to the design
- We have familiarized ourselves with the *concetual database design* by defining entity types and their relationship based on the requirements
- The *logical database design* is the process of refining and translating the conceptual schema into a logical database schema based on a specific data model, e.g. the relational model

Logical database design

- Entity types, attributes, and relationship types can be directly transformed into relations with some simple rules
- Typically, the logical database design process includes the following types of activities:
 - i. Deriving relations for the logical data model
 - ii. Validating relations using normalisation
 - iii. Validating relations against user transaction
 - iv. Double-checking integrity constraints
 - v. Reviewing logical data model with user

Deriving relations for the logical data model

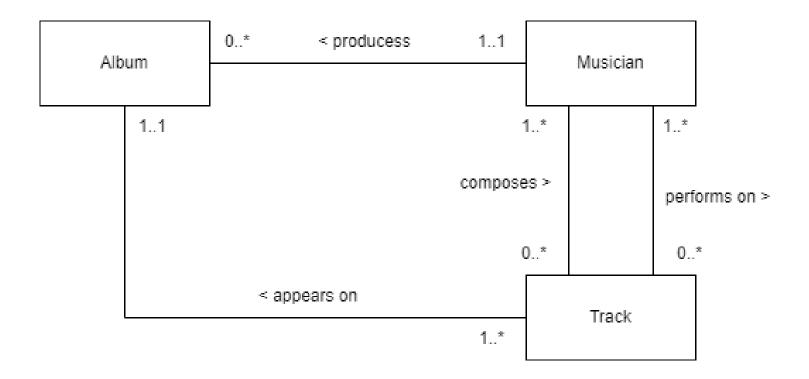
- The process starts by *deriving relations for the logical data model*, which includes:
 - i. Creating the relations
 - ii. Refining the attributes
 - iii. Determing primary and foreign keys
 - iv. Determing other types of integrity constraints

Creating relations

- We create the relations in the following manner:
 - For each entity type, create a relation that includes all simple attributes of the entity
 - For M:N (many-to-many) relationship types, create a "bridge relation" to represent the relationship
 - For multi-valued attributes, create a new relation to represent the multi-valued attribute. For example, a person may have several phone numbers, but multivalued attributes are not allow in relations

Example of creating relations

• Let's consider creating relations for the following conceptual model:



Example of creating relations

- In the example there's two many-to-many relationships:
 - A musician performs on multiple tracks and tracks have multiple performers
 - A musician composes multiple tracks and tracks are composed by multiple musicians
- In this case we create two bridge relations: *Track_Performer* and *Track_Composer*
- The are no multi-valued attributes, which leaves with the following relations: Album, Musician, Track, Track_Performer, and Track_Composer

Refining the attributes

- Once we have created the relations we need to refine the attributes in the following manner:
 - Divide a non-atomic attribute into smaller (atomic) attributes. For example person's address can be divided into, city, postal code and street address
 - Refine attribute domains: data types and lengths, requiredness, validation rules etc.
 - Define which attributes can have NULL values. Allow NULL in an attribute only based on strong arguments

Determing primary keys

- There should be exactly one primary key in each relation
- The primary key can be either a *simple key* (single column) or a *composite key* (several columns)
- By definition, the primary key should always satisfy the properties of requiredness, uniqueness and minimality
- The primary key *should remain stable*. That is, primary key values should not need to be updated later
- The primary key should be *reasonably short*
- The primary key should have *no privacy issues*. For example social security number has privacy issues

Determing primary key for a weak entity type

- A weak entity type is an entity type that is dependent on the existence of another entity type
- For example CourseGrade is existence-dependent on Student and CourseOffering
- When a relation derived from a weak entity type, the natural primary key is partially or fully derived from the weak entity type's owner entity type
- For example, the natural primary key of the *CourseGrade* relation is a composite key that *includes columns from two foreign keys*, one referencing *Student* and other referencing *CourseOffering*
- The primary key of the weak entity's relation cannot be made until after the foreign keys have been determined for the relation

Surrogate keys

- If there is initially no suitable candidate key for a relation, then we cannot determine a natural primary key
- We have to take care of the situation by including an extra attribute in the relation to act as the primary key
- This kind of primary key is a *surrogate key*
- Surrogate keys are commonly generated values, such as incrementing or random numbers

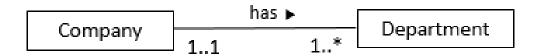
Alternate keys

- Candidate keys that are not selected to be primary the key are called *alternate keys*
- We should consider the use of the *unique constraint* on alternate keys to make sure that their values remain unique:

```
Student (<u>studentnumber</u>, ssn, familyname, givenname)
UNIQUE (ssn)
```

• Especially, when we are using a surrogate primary key, a unique constraint on at least one natural alternate key improves data quality

- In a relational database, *relationships* are represented by the *primary key/foreign key mechanism*
- Before deciding where to place the foreign key we have to identify the *parent* entity type and the child entity type involved in the relationship



- For example, in the ER diagram above Company is the parent entity type and Department is the child entity type
- When we translate this diagram to relation schemas we must do the following:
 - We create the two relations: *Company* and *Department*
 - We determine primary keys for both relations
 - We represent the relationship by placing a copy of the parent relation's (Company) primary key into the child relation (Department), to act as the foreign key

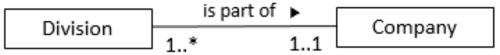
• The result could be for example the following relation schema:

```
Company (<a href="mailto:company_id">company_id</a>, name)

Department (<a href="mailto:department_id">department_id</a>, company_id, name)

FK (<a href="mailto:company_id">company_id</a>)
```

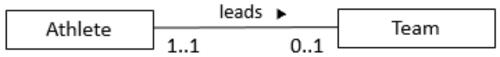
- To know in which relation to place the foreign key we need to *identify the* relationship type
- Most often the structure falls in one of these categories:
 - N:1 (many-to-one) binary relationship



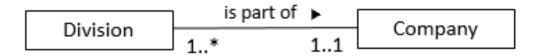
M:N (many-to-many) binary relationship



1:1 (one-to-one) binary relationship



N:1 (many-to-one) binary relationship



- Place a copy of the parent relation's ("..1" side) primary key into the child relation ("..*" side), to act as a foreign key
- If the child relation is derived from a weak entity type, then the primary key of the child relation is typically a composite key
- In the example above, we would get the following relation schema:

```
Company (<a href="mailto:company_id">company_id</a>, name)

Division (<a href="mailto:division_id">division_id</a>, company_id, name)

FK (<a href="mailto:company_id">company_id</a>)
```

M:N (many-to-many) binary relationship

```
Athlete participates ► Race
```

- Create a bridge relation to represent the relationship
- Place a copy of the primary key from each of the parent relations into the bridge relation to act as foreign keys
- Typically, the bridge relation's primary key is a composite key that includes the both foreign keys
- In the example above, we would get the following relation schema:

```
Athlete (athlete_id, first_name, family_name)
Race (race_id, name, date)
Race_Participation(athlete_id, race_id)
FK (athlete_id), REFERENCES Athlete(athlete_id),
FK (race_id), REFERENCES Race(race_id)
```

1:1 (one-to-one) binary relationship

```
Athlete leads ► Team

1..1 0..1
```

- In case of mandatory participation ("1..1") on one side only, place a copy of the primary key from the relation on the "1..1" side into the relation on the "0..1" side to act as the foreign key
- In case of mandatory participation on both sides we can usually combine the two relations into one relation
- In case *optional participation* ("0..1") *on both sides* the foreign key can be placed in either relation
- In the example above, we would get the following relation schema:

```
Athlete (<u>athlete_id</u>, first_name, family_name)
Team (<u>team_id</u>, athlete_id, name)
FK (athlete_id), REFERENCES Athlete(athlete_id)
```

Multi-value attributes

Employee

empno family name given name email[0...*]

- A relation can't have attributes with *multiple values*, such as the *email* attribute of the *Employee* entity type above
- In such case, we must create a *new relation* to represent the multi-valued attribute
- We move the attribute from the original relation and place it to the new relation
- We Place a copy of the parent relation's primary key into the child relation, to act as the foreign key

Multi-value attributes

Employee

empno family name given name email[0...*]

• In the example above, we would get the following relation schema:

```
Employee (empno, first_name, family_name)
Email (email, empno)
FK (empno), REFERENCES Employee(empno)
```

Summary

- The objective of logical database design is to translate the conceptual schema into a logical database schema based on a specific data model
- When we derive relations from entity types, we create a relation for each entity type
- Many-to-many relationship requires an additional bridge relation
- There should be exactly one primary key in each relation
- The foreign key placement depends on the relationship type (many-to-one, many-to-many or one-to-one)