Relational Data Modelling

Table of Contents

[1 ERD – Entity-Relationship-Diagram 2](#_Toc91411691)

[1.1 (min, max) Notation 2](#_Toc91411692)

[1.2 Crow’s Foot Notation 3](#_Toc91411693)

[1.3 Example - Mitarbeiter 3](#_Toc91411694)

[2 ERD to Relational Model (Relational Schema) 5](#_Toc91411695)

[3 Important Concepts to Remember 6](#_Toc91411696)

[4 Joins 7](#_Toc91411697)

[4.1.1 INNER JOIN 8](#_Toc91411698)

[4.1.2 LEFT JOIN 8](#_Toc91411699)

[4.1.3 RIGHT JOIN 8](#_Toc91411700)

[4.1.4 OUTER JOIN 8](#_Toc91411701)

1. ERD – Entity-Relationship-Diagram

**Diagram

Description automatically generated**

* 1. Diagram

     Description automatically generatedText

     Description automatically generated(min, max) Notation

**Diagram

Description automatically generated**

Ein Mann heiratet **genau eine** Frau

Eine Frau heiratet **genau ein** Mann

Eine Abteilung hat mindestens **ein** Arbeitsplatz

Eine Abteilung hat höchstens **n (mehrere)** Arbeitsplätze

Eine Vorlesung wird von **genau einem** Prof gehalten

Ein Prof hält entweder **keine oder mehrere** Vorlesungen

Ein Stundent besucht keine oder mehrere Vorlesungen

Eine Vorlesung wird von keinem oder mehreren St. besucht

* 1. Crow’s Foot Notation

A little different than the (min, max) notation. Generally the representation of the entities are more like tables with attributes listed in the entity box.

* The **cardinalities** are read different than the (min, max) notation.
* The **relationship** texts are optional. Sometimes they are not written

Diagram

Description automatically generated

Eine Abteilung hat mindestens **ein** Arbeitsplatz

Eine Abteilung hat höchstens **n (mehrere)** Arbeitsplätze

* 1. Example - Mitarbeiter

Ein **Mitarbeiter** mit Name, Vorname, Adresse und Gehalt wird identifiziert durch seine Mitarbeiternummer.

Ein **Produkt** (mit Produktnummer und Name) wird von vielen, aber mindestens einem **Mitarbeiter** produziert und ein **Mitarbeiter** produziert viele **Produkte**.

In einem **Projekt**, identifiziert durch die Projektnummer und gekennzeichnet mit einer Beschreibung, arbeiten viele **Mitarbeiter** in unterschiedlichen **Rollen** mit einer eindeutigen RollenNr und einer Beschreibung und ein **Mitarbeiter** arbeitet in vielen **Projekten**.

Ein **Mitarbeiter** arbeitet genau in einer **Abteilung** und in einer **Abteilung** arbeiten mehrere **Mitarbeiter**.   
Eine **Abteilung** wird gekennzeichnet durch eine Abteilungsnummer (eindeutig) und durch einen Abteilungsnamen.

Eine **Abteilung** gehört genau zu einem (numerisch gekennzeichneten) **Unternehmen**; ein **Unternehmen** besteht aus mehreren **Abteilungen**.

Diagram

Description automatically generated

Asd

1. ERD to Relational Model (Relational Schema)

**TODO -** [**https://www.youtube.com/watch?v=OwdFzygGZqk**](https://www.youtube.com/watch?v=OwdFzygGZqk)

Diagram

Description automatically generated

Rules:

* For composite attributes: only include the most specific attributes (for **Car**, do not include **model**)
* **1:1** relationships are NOT modeled in databases. Meaning 2 of the entities will be modeled in one. You can put the columns of one entity into the other.
* **TODO** – in **1:N** relationships, the N side does not have any reference/information in it’s table about the 1 side. Only the 1 side will have the N sides primary key, saved as a foreign key
* In **N:M** relationships, a 3rd table must be created containing both primary keys as foreign keys (composite key?). There can also be 3 Entities with N:M relationships, the same applies. See the example in the lecture.
* **TODO** – **struktur, hierarchie, generalisierung und spezialisierung. How to map**?

1. Important Concepts to Remember

* In a table, there cannot be multiple rows with the same primary key. The whole point of a primary key is to uniquely identify a row/object/entity saved in a table. (When you do a join then you will see duplicated primary keys and that’s fine, works because TODO)
* A column cannot/should not contain any kind of list. If that’s the case, for example we want to save multiple cars a user owns in a column [“Ferrari”, “Opel”, …]. This is a typical case for a **one-to-many** relationship. AFAIK databases also does not have any collection types since it is not needed. All collections are then split up into tables, a references to the other.

A **one-to-many** example “a user can have many vehicles and a vehicle has one user”:

Table

Description automatically generatedTable

Description automatically generated

In a **one-to-many** relationship

* The primary key on the one side (user) is written to the many side (vehicle) as a foreign key. Because we cannot write it the other way around. Writing the vehicle ID to the user table as a foreign key is not possible because a user can have multiple vehicles and we cannot duplicate the entries, since then the primary keys would be duplicated.
* When we do pure SQL queries, we need to do a **JOIN** if we also want to get the vehicle data for each user. Since doing SELECT \* FROM User will only give us the userId and name. This is done automatically by Hibernate if we have a field which was annotated. If we fetch the user, hibernate will do a join and also fetch the vehicles for each user.
* The objects on the “many” side can always be mapped to **only one other entry**. The Ferrari with the ID 3 cannot also be owned by Ahmet because then it would be a many-to-many relationship.

In a **many-to-many** relationship

* We cannot use a **Join Column** on either table (like we did with one-to-many) therefore we have to create a separate **join table**.
* A **separate Join table** is created which contains both primary keys of the 2 tables. Those keys both define the primary key, which is called a composite primary key **TODO**? Is that always the case?
* The **many-to-many** tables are split up into **2 one-to-many** relationships.
* **TODO** show a different example and show how do we fetch all data, with joins??

Diagram

Description automatically generated

1. Joins

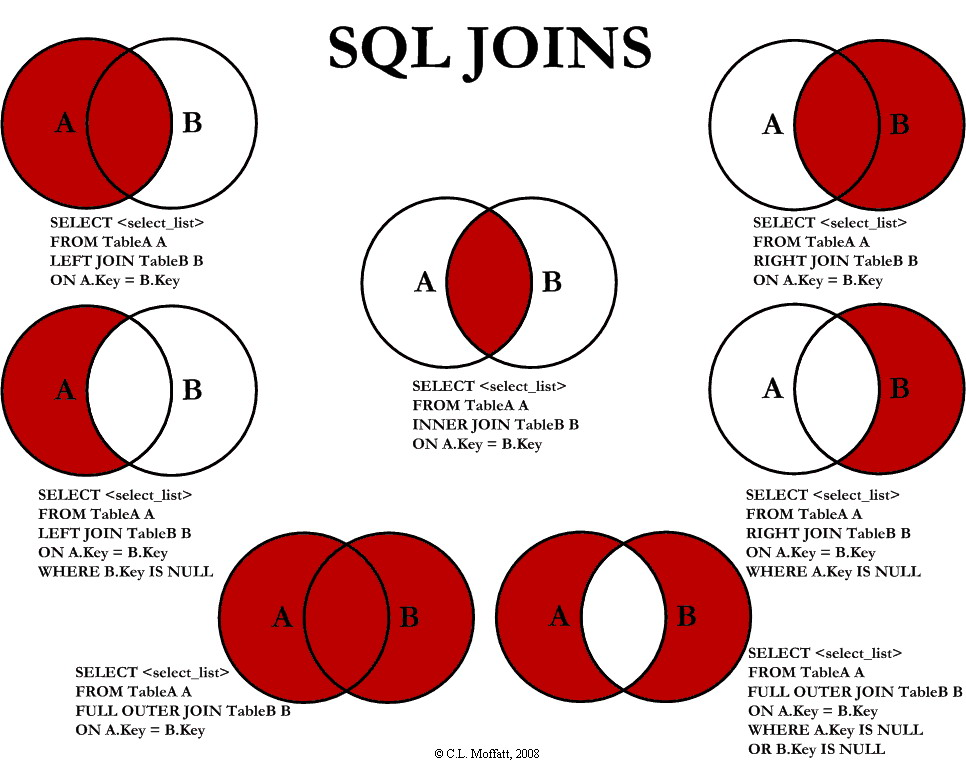
A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

* If you say **JOIN** it means by default **INNER JOIN**
* You define a column which exist (and is the same) in two or more tables and then you can join the rows of each table into one row.

Diagram

Description automatically generated with low confidence

* **(INNER) JOIN**: Returns records that have matching values in both tables. All other are dropped/not displayed.
* **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



Table

Description automatically generatedTable

Description automatically generated

* + 1. INNER JOIN

Only the the rows are displayed which has matches in both sides



Table

Description automatically generated

* + 1. LEFT JOIN

Now we take everything on the left side, regardless if the common column has any matches.  
Since Ahmet does not have any cars, all fields of the right table will be null.



Table

Description automatically generated

* + 1. RIGHT JOIN

Same now for Skoda, since it has no owner on the left side, all values will be null.



Table

Description automatically generated

* + 1. OUTER JOIN

There are no **FULL OUTER JOINS** in MySQL. A similar structure is **UNION** but using a union query will remove duplicates, and this is different than the behavior of full outer join that never removes any duplicates.

If we did this operation in another database which supports full joins, we would see all the results from all tables, both rows Ahmet and Skoda with their other columns being null.