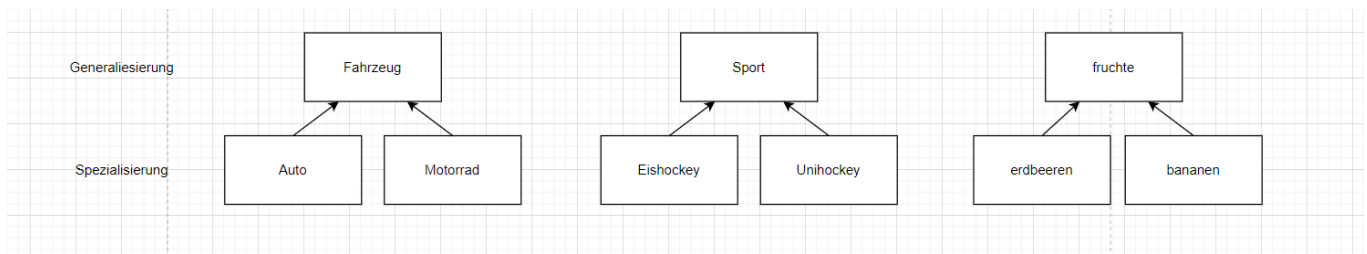


Module 164

Generalization / specialization (person with the role of driver or dispatcher).

The database modeling approach discussed here is based on the attribute concept, where attributes are defined with specific characteristics and assigned to entity types. A problem arises when multiple entity types share many attributes, leading to redundancy if a real-world object is described by several entity types. For instance, employees who also act as customers or drivers who also work as dispatchers illustrate this issue. According to Zehnder (1989), "local attributes" should only appear once in a database to avoid redundancy. The solution is to consolidate common attributes into a general entity type (generalization) while keeping non-common attributes within their respective specialized entity types (specialization). To prevent information loss, specialized tables should reference generalized tables through foreign keys, establishing an "is-a" relationship, similar to inheritance in object-oriented modeling.



Relationship Types: Identifying/ Non-Identifying relationship

In databases, relationships between tables can be categorized into identifying and non-identifying relationships. Here's a brief explanation of each:

1. Identifying Relationship:

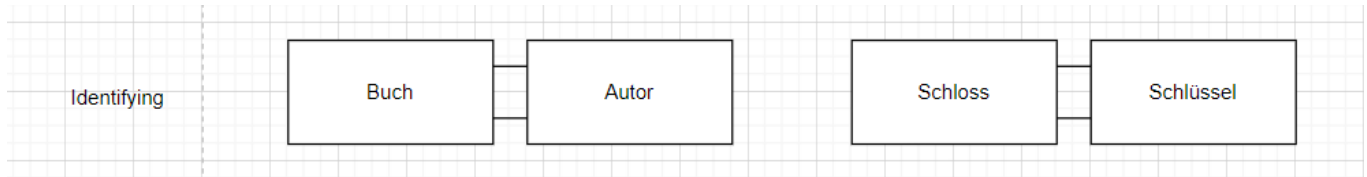
- The foreign key in the child table is part of its primary key.
- This forms a key combination of multiple attributes, with the child table's record partially identified by the parent table's record.
- Example: A room in a building where the room's ID includes the building's foreign key, making it crucial for identification.

2. Non-Identifying Relationship:

- The foreign key in the child table is not part of its primary key.
- This allows for the foreign key value to change without affecting the child table's identity.
- Example: An employee and department relationship where the department's foreign key is not part of the employee's primary key, allowing for changes in the department without altering the employee's identity.

own example

classmates examples



use cases made by chatGPT for identifying Relationship.

1. Public Policy

- **Social Program Evaluation:** Identifying relationships between program interventions, target populations, and outcomes helps in assessing program effectiveness.
- **Crime Analysis:** Understanding relationships between socio-economic factors, crime incidents, and law enforcement practices aids in developing effective crime prevention strategies.
- **Urban Planning:** Mapping relationships between urban infrastructure, population dynamics, and land use patterns supports sustainable urban development.

2. Environmental Studies

- **Ecosystem Analysis:** Identifying relationships between species, habitats, and environmental factors aids in ecosystem conservation and management.
- **Climate Change Research:** Understanding the relationships between human activities, climate variables, and environmental impacts helps in developing mitigation and adaptation strategies.
- **Resource Management:** Analyzing relationships between natural resources, consumption patterns, and sustainability practices supports effective resource management.

3. Research and Development

- **Scientific Research:** Identifying relationships between variables in experiments and studies helps in hypothesis testing and theory development.
- **Innovation Networks:** Mapping relationships between researchers, institutions, and innovations fosters collaboration and knowledge transfer.

- **Patent Analysis:** Understanding relationships between patents, inventors, and technological domains supports strategic intellectual property management.

4. Technology and Telecommunications

- **Network Analysis:** Identifying relationships between network components (e.g., servers, routers, devices) aids in network optimization and troubleshooting.
- **User Behavior Analysis:** Understanding relationships between user activities, service usage, and device interactions helps in improving user experience and service delivery.
- **Cybersecurity:** Recognizing relationships between different network activities and potential threats enhances threat detection and response strategies.

5. Social Networks

- **Friendship and Interaction Analysis:** Identifying relationships in social networks helps understand community structure, influence patterns, and information dissemination.
- **Influencer Identification:** Recognizing relationships between users and their content interactions helps in identifying key influencers within a network.
- **Behavioral Analysis:** Understanding relationships between user behaviors and network dynamics can improve content recommendations and user engagement.

DBMS (Database Management System)

A database system (DBS) is designed for efficient, consistent, and permanent electronic data management, providing data subsets in various formats for users and applications. A DBS comprises the database management system (DBMS) and the database (DB). The DBMS organizes data storage and controls access, offering a database language for querying and managing data. The most common database system is relational.

Features of a DBMS

A DBMS must provide:

- **Integrated data storage:** Unified management of all application data, allowing complex relationships and efficient data linking. Controlled redundancy can enhance processing efficiency.
- **Database language:** Includes Data Request (retrieval), Data Manipulation Language (DML), Data Definition Language (DDL), and Data Control Language (DCL).
- **User interfaces:** Various interfaces, such as query languages, application programming interfaces, graphical user interfaces (GUI), and web access.

- **Catalogue:** Access to metadata via a data dictionary.
- **User views:** Different views for different user classes, defined in the database's external schema.
- **Consistency control:** Ensures database correctness through integrity assurance, defined by user constraints, and physical integrity.
- **Data access control:** Prevents unauthorized data access through rules and defined rights.
- **Transactions:** Combines multiple changes into atomic transactions, ensuring durability if successful.
- **Multi-user capability:** Synchronizes competing transactions to avoid conflicts, maintaining data isolation for users.
- **Backup:** Restores the database to a correct state after errors.

Advantages of Using a Database

- **Standards:** Facilitates central data organization standards.
- **Efficient data access:** Uses advanced techniques for storing and retrieving large data volumes.
- **Shorter development times:** Offers common functions for faster application development.
- **Flexibility:** Allows database structure modifications without significant impact on existing data and applications.
- **High availability:** Supports high-availability applications through synchronization.
- **Cost-effectiveness:** Centralized investment in powerful hardware reduces overall costs.

Disadvantages of Database Systems

- **High initial investment:** Requires significant expenditure on hardware and software.
- **General-purpose software:** Less efficient for specialized applications.
- **Optimization limits:** Can only be optimized for some applications.
- **Additional costs:** Involves expenses for data security, synchronization, and consistency control.
- **Skilled personnel:** Needs experts like database designers and administrators.
- **Centralization vulnerability:** Risks associated with centralization.

LIST GIVEN BY THE EXERCISE

DBMS	\$?	Manufacturer	Model/Characteristics
Adabas	\$	Software AG	NF2 model (non-normalized)
Cache	\$	InterSystems	hierarchical, "postrelational"
DB2	\$	IBM	Object-relational
Firebird		-	relational, based on InterBase
IMS	\$	IBM	hierarchical, mainframe-DBMS
Informix	\$	IBM	Object-relational
InterBase	\$	Borland	relational
MS Access	\$	Microsoft	relational, desktop system
MS SQL Server	\$	Microsoft	Object-relational
MySQL		MySQL AB	relational
Oracle	\$	ORACLE	Object-relational
PostgreSQL		-	object-relational, emerged from Ingres and Postgres
Sybase ASE	\$	Sybase	relational
Versant	\$	Versant	Object-oriented
Visual FoxPro	\$	Microsoft	relational, desktop system
Teradata	\$	NCR Teradata	High-performance relational DBMS, especially for data warehouses

LIST FROM DB ENGINE RANKING

DBMS	\$?	Manufacturer	Model/Characteristics
Oracle	\$	ORACLE	Relational, Multi-model
MySQL		ORACLE	Relational, Multi-model
Microsoft SQL Server	\$	Microsoft	Relational, Multi-model
PostgreSQL		PostgresSQL Global Development Group	Relational, Multi-model
MongoDB		MongoDB, Inc	Relational, Multi-model
Redis		Redis project core team, inspired by Salvatore Sanfilippo	Relational, Multi-model
Elasticsearch		Elastic	Relational, Multi-model
IBM Db2	\$	IBM	Relational, Multi-model

DBMS	\$?	Manufacturer	Model/Characteristics
Snowflake	\$	Snowflake Computing Inc.	Relational
SQLite		Dwayne Richard Hipp	Relational
Microsoft Access	\$	Microdsoft	Relational
Cassandra		Apache Software Foundation	Wide Column, Multi-model
MariaDB		MariaDB Corporation ab (MariaDB Enterprise), Maria DB Founsation(community MariaDB Server)	Relational, Multi-model
Splunk	\$	Spluk Inc.	
Databricks	\$	Databricks	Multi-model
Microsoft Azure SQL Database	\$	Microsoft	Relational, Multi-model

COMPARISON OF THE TOP 3

Adabas VS Oracle

This is a rough comparison between both of them

Name	Adabas	Oracle
Primary database model	Multivalue DBMS	Relational DBMS
DB-Engines Ranking	Score 3.17 Rank 94 Overall Rank 1 Multivalue DBMS	Score 1236.29 Rank 1 Overall Rank 1 Relational
Developer	Software AG	Oracle
Initial release	1971	1980
License	Commercial	Commercial

MySQL Vs Cache

Sadly Cache isn't on the list of DB-Engines.com so I can't compare both of them.

Microsoft SQL Server Vs DB2

Sadly DB2 isn't on the list of DB-Engines.com so I can't compare both of them.

LB1- Teil 1 Ecolm 5Seiten 1/2h (1Tag-3Tag)

Teil2 Openbook Praxis 40min

Data type	MariaDB	Example	Remark/Setting
Integers	INT	INT 1254	from -32768 to 32767
Natural Numbers	doesn't exist		
Fixed-Point numbers (decimals)	Decimal (M[,D])	Decimal (6,2) 1234.56	M= Total number of digits D=Decimal places
Bulleled Types			
Boolean (logical values)	Boolean(TINYINT(1))		
Character (single character)			
Floats	FLOAT		
Fixed-length string	CHAR		
Variable length string	VARCHAR		
Date and/or time	Date (YYYY-MM-DD)	Date (2024.06.03)	
Timestamp	TIMESTAMP (YYYY-MM-DD HH:MM:SS)	TIMESTAMP (2024-06-03 17:14:42)	
Binary data objects of variable length (e.g. image)			
Compound	doesn't exist		
JSON	JSON Data Type		
Data type	mySQL	Example	Remark/Setting
Integers			
Natural Numbers			
Fixed-Point numbers (decimals)			
Bulleled Types			
Boolean (logical values)			
Character (single character)			
Floats			
Fixed-length string			
Variable length string			
Date and/or time	Date (YYYY-	Date	

Data type	mySQL	Example	Remark/Setting
	MM-DD)	(2024.06.03)	
Timestamp			
Binary data objects of variable length (e.g. image)			
Compound			
JSON			

Creating a constrain relationship

In the logical and physical data model, the relationships of the relational database are limited due to the possible settings (constraints) of the foreign keys!

The following relationships can be realized in the physical model:

Relationship <i>MasterTab.left : DetailTab.right</i>	possible	Not possible ¹ → becomes	Constraints FK
one-to-one	1:C C:C	1:1 → 1:c -	NN UQ -- UQ
One to many	1:mc c:mc	1:m → 1:mc c:m → c:mc	NN -- -- --
many too many only via transformation table		m:m, m:mc, mc:m, mc:mc → 1:mc-[TT]-mc:1	- NN -- & NN --

Supplement ALTER TABLE tbl ADD CONSTRAINT <...> FOREIGN KEY(...

We can insert foreign key constraints manually after the creation of the table with ALTER TABLE. An example would be the following:

```
ALTER TABLE <DetailTab>
    ADD CONSTRAINT <Constraint> FOREIGN KEY (<Foreign key>)
    REFERENCES <MasterTab> (Primary key);
```


To add an unique key we could use the following command:

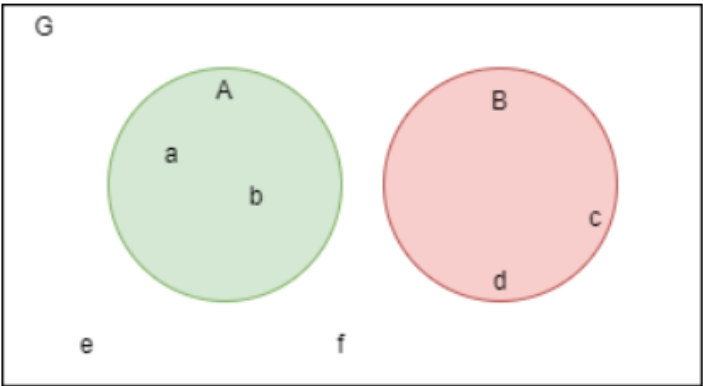
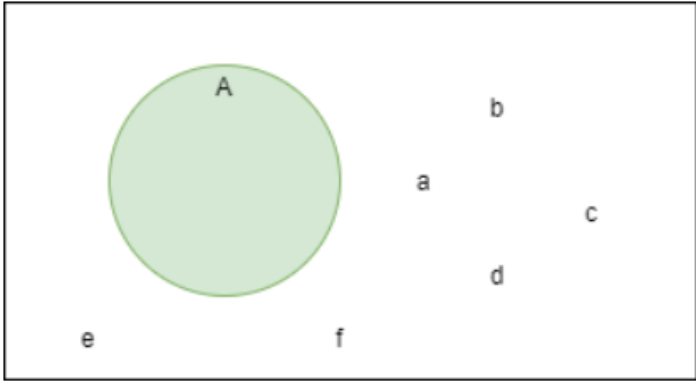
```
ALTER TABLE <Tabelle>  
    ADD UNIQUE(<FS_Name>);
```

It's important for us to know the meaning of every word we use so i will leave the following table that contains the explanation of some key words:

<Wert>	Explanation
<DetailTab>	Detail table name
<Constraint>	Freely definable name. Possible naming convention: FK_Detailtabelle_Mastertabelle, e.g. FK_Autoren_Ort.
<Fremdschlüssel>	Name of the foreign key attribute of the detail table
<MasterTab>	Master/Primary Table Name
<Primärschlüssel>	Name of the primary key attribute of the master/primary table

SET theory

For this i will leave the screenshots from the gitlab since i think they're self explanatory and easy to understand.

Symbol	Description
Uppercase and lowercase letters and \in , \notin	<p>Capital letters denote a lot. In the example below, there are three sets G (basic set), A and B.</p> <p>Lowercase letters denote elements that are assigned to a set (at least the basic set)</p> <p>$G=\{a,b,c,d,e,f\}$, $A=\{a,b\}$, $B=\{c,d\}$</p> <p>\in shows that an element is contained in a set, for example: $a \in A$, $b \in A$ or $d \in B$</p> <p>\notin indicates that an element is not contained in a set, for example: $a \notin B$ or $d \notin A$</p> 
$\{\}$ or \emptyset	<p>Denotes an empty set. In the following example, the set A is empty.</p> <p>$A=\{\}$</p> 

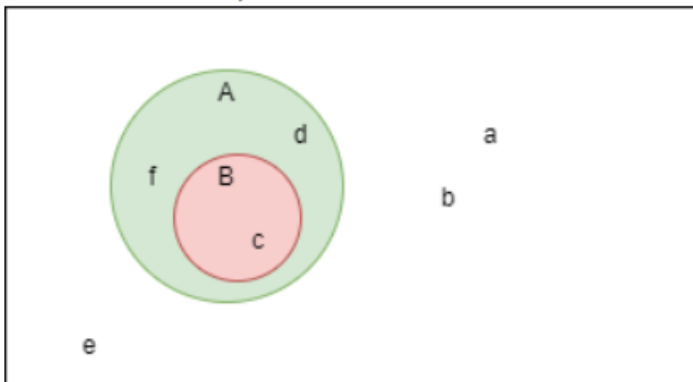
\subset, \subseteq

\subset or \subseteq means **subset** of. for example:

$B \subset A$

This is the case when a quantity is completely contained in another quantity.

For an element x , this means that if $x \in B$ is *also* $x \in A$.



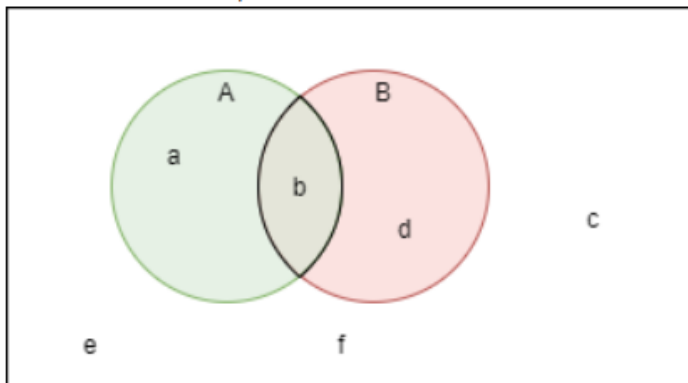
Attention: There is a difference between the two characters, but we ignore it here (real and fake subsets).

\cap

\cap refers to the **intersection** between two sets, for example:
 $A \cap B$.

If two sets do not overlap, the intersection is the empty set.

For an element x , this means that $x \in B$ **and** $x \in A$.



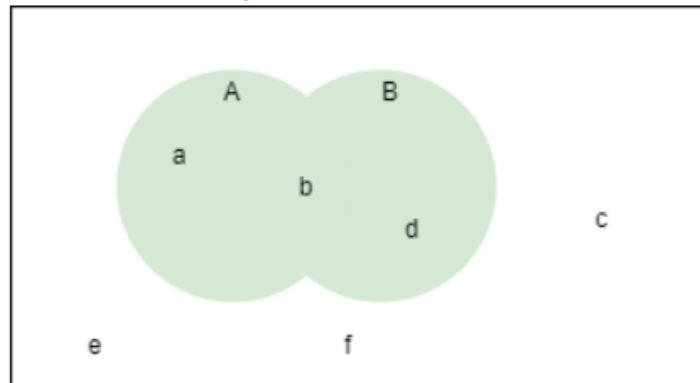
\cup

\cup denotes the **union set** of two sets, for example:

$A \cup B$.

The two quantities do not have to overlap.

For an element x , this means that x is $\in A$ **or** x is $\in B$.

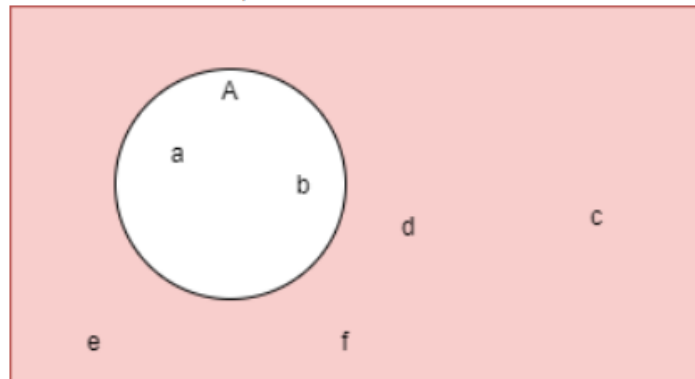


X^c

X^c denotes the **complementary set**, i.e. all elements that are not contained in the set X , for example:

A^c (all elements in the area marked in red).

For an element x , this means that x is $\in G$ **and** x is $\notin A$.

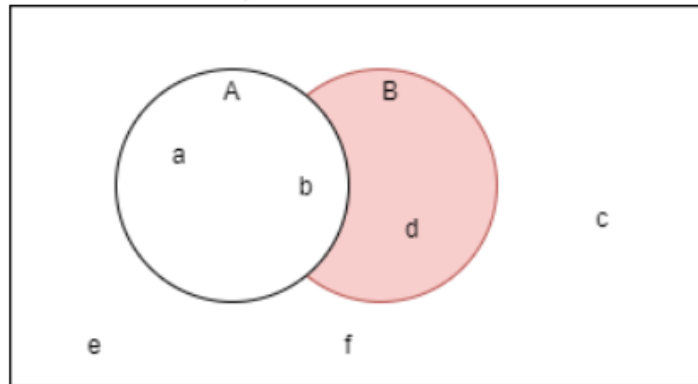


\

\ denotes the **difference** between two sets, for example $B \setminus A$.

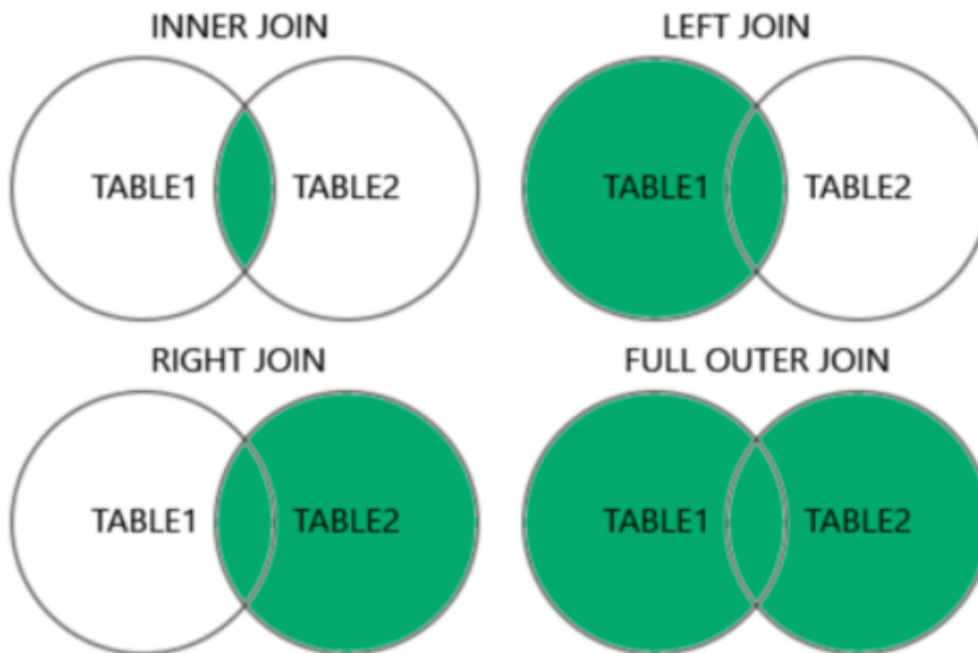
It includes all elements that are contained in the set B, but not in the set A.

For an element x , this means that x is $\in B$ **and** x is $\notin A$.



Now i'm going to leave a screenshot that explain the different types of JOIN

- **(INNER) JOIN** : Returns records that have matching values in both tables
- **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



Deletion in Professional Databases

In pro databases, we usually do data acquisition, modification, query, and deletion with SQL. But deleting data with the SQL `DELETE` command is usually a no-go because it leads to losing info, which is bad for many reasons.

Example: If an employee leaves and we delete them from the personnel table, we lose all the info related to them. So we can't see what tasks they did before. Instead, we shouldn't delete but mark the employee as inactive or add exit info to keep the data intact.

Also, updating (overwriting) entries doesn't keep history. For example, if an object is lent to different people multiple times, we need extra tables to track each lending period. This way, we can still analyze past data (like how often an item was borrowed).

In POS systems, if a purchase record could just be deleted, it might be misused, leading to extra cash in the register. Instead of deleting, we log cancellations with details like time and reason to keep things transparent and prevent fraud.

Referential integrity managed by the DBMS using foreign key constraints means we can't delete data without also deleting related records. For example, on Wikipedia, user edits are historized, so we can't just delete user info. Problematic users get blocked instead.

Data Integrity

Data integrity makes sure the database info is accurate, consistent, and complete. Key points are:

1. **Uniqueness and Consistency:** Each record should be unique to prevent duplicates and accidental changes.
2. **Referential Integrity:** Relationships between tables must be consistent, like ensuring an order only links to existing customers.
3. **Data Types:** Use the correct data types to handle info properly, like storing phone numbers as strings, not integers.
4. **Data Restrictions:** Set limits to ensure valid data, like allowing only positive numbers in some fields or enforcing correct email formats.
5. **Validation:** Check data before inserting to ensure it meets the database requirements.

Foreign key rules when deleting

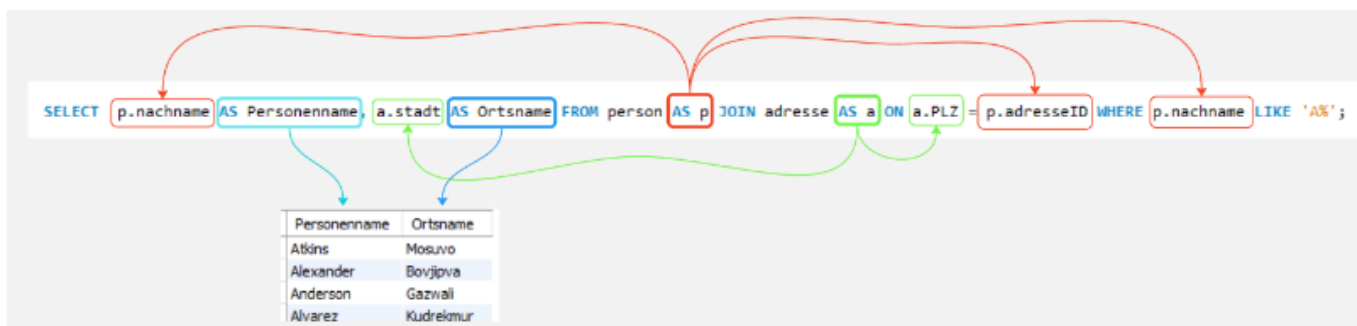
There's ON DELETE rules for deleting a primary table that has a constrain.

DELETE	Rule	Meaning
ON DELETE	NO ACTION	A DELETE in the primary table can only be executed if there is no foreign key with the corresponding value in any detail table. <i>This is the standard procedure if nothing is specified.</i>
	CASCADE	A DELETE in the primary table also leads to a deletion of the corresponding records in the foreign key table. Be careful with this rule, data may be deleted unintentionally!
	SET ZERO	If there is a DELETE in the primary table, the corresponding records in the foreign key table are set to NULL. Note: This setting only works for c:m(c) and c:c relationships, i.e. if NULL is allowed for FK!
	DEFAULT	In the event of a DELETE in the primary table, the corresponding records in the foreign key table are set to the DEFAULT value, if one has been defined. Otherwise they are set to (for c:m(c) and c:c relationships) NULL

There's not only ON DELETE rules but ON UPDATE rules too. These come into play when a primary key value changes. However, we let the system manage the primary key values (*without any further information*) (AUTO INCREMENT). So the ID values never change, so the ON UPDATE rules are irrelevant to us.

SELECT ALIAS

Aliases are used to give temporary names to columns in a table or a table itself. The aliases only exist for the duration of the query in order to make names of the columns more readable. Table aliases must be used throughout the statement. So in the example under it can't stand - will be replaced everywhere! `SELECT person.nachname ... ``person``p`



Aggregate functions

These functions are used to summarize or calculate data from a column. These functions are typically used in conjunction with the GROUP BY statement to group data into groups and then calculate or summarize on those groups. Some of the most common aggregate functions in MySQL are the following.

These aggregate functions can also be used in combination to execute complex queries and obtain summarized results.

COUNT

This function returns the number of rows in a table or group. It can also be used to count the number of values in a column. `NULL`

```
SELECT COUNT(*) FROM customers;
```

This returns the **number of all records** in the table. `customers`

```
SELECT COUNT(salary) FROM customers;
```

This returns the number of all rows of the column 'salary' **without** `NULL` in the table. `customers`

SUM

This function calculates the sum of the values in a column or group.

```
SELECT SUM(salary) FROM employees;
```

This returns the sum of the values in the column of the table. `salary` `employees`

AVG

This function calculates the average value of the values in a column or group.

```
SELECT AVG(salary) FROM employees;
```

This returns the average value of the values in the column of the table. `salary` `employees`

MIN

This function returns the smallest value in a column or group.

```
SELECT MIN(salary) FROM employees;
```

This returns the smallest value in the column of the table. `salary`employees`

MAX

This function returns the largest value in a column or group.

```
SELECT MAX(salary) FROM employees;
```

This returns the largest value in the column of the table. `salary`employee`

GROUP BY

In MySQL, the GROUP BY statement is used to group and summarize data into groups. It is usually used along with aggregate functions such as the ones we saw before.

The statement groups data based on the contents of one or more columns. This means that all rows with the same value in the grouping column will be grouped together. The GROUP BY statement then returns a result set that contains the results of the aggregate functions for each group.

EXAMPLE: We have a table named "orders" that contains the following columns:

- "order_id"
- "customer_id"
- "order_date"
- "order_total"
- We want to calculate the total value of orders for each customer and group the results by customer.

We can do this with the statement as follows: `GROUP BY`

```
SELECT customer_id, SUM(order_total)
FROM orders
```

```
GROUP BY customer_id;
```

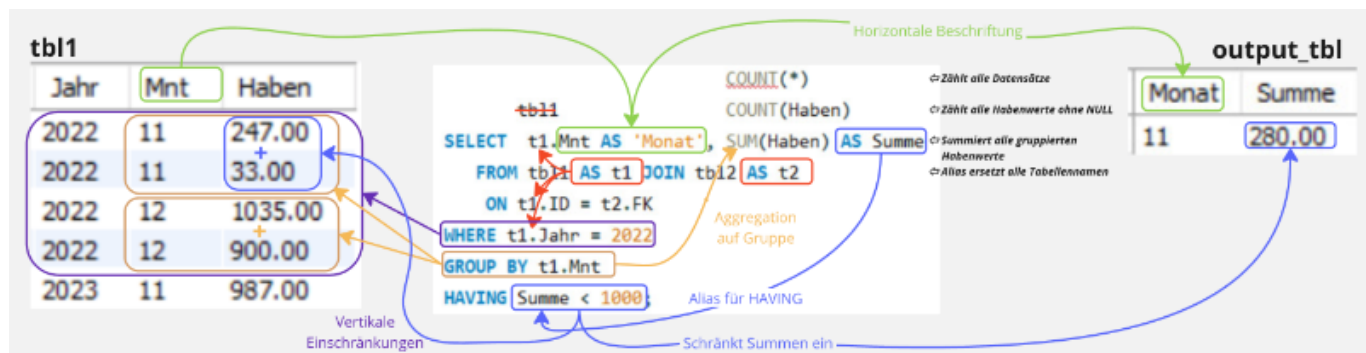
In this example, the data in the table is grouped by column. The function is used to calculate the total value of orders for each group. The result shows the customer ID and the total value of the orders for each customer. `orders``customer_id``SUM`

The result of the query will look like this:

customer_id	SUM(order_total)
1	350.50
2	210.00
3	450.25

HAVING Condition

In MySQL, this statement is used to filter the results of grouping operations performed by the statement. The clause is used to apply conditions to the aggregated results returned by the aggregate functions used in the statement.



EXAMPLE: We have a table named "orders" that contains the following columns:

- "order_id"
- "customer_id"
- "order_date"
- "order_total"

We want to calculate the total value of orders for each customer and select only those customers whose order total is greater than 500.

We can do this with the - and statement as follows: `GROUP BY``HAVING`

```
SELECT customer_id, SUM(order_total)
FROM orders
GROUP BY customer_id
HAVING SUM(order_total) > 500;
```

In this example, the data in the table is grouped by column. The function is used to calculate the total value of orders for each group. Then, the HAVING statement is used to select only those customers whose total order total is greater than 500. `orders` `customer_id` `SUM``

The result of the query will look like this:

customer_id	SUM(order_total)
1	650.50
3	900.25

SUBQUERY (SUBSELECT)

A query in a MySQL database is a query **within another query**. It is used to retrieve data from one table that meets the conditions that are determined based on data in another table.

If we using subqueries:

- We must always enclose the subquery in parentheses.
- Pay attention to the operator we are using to compare the result of the subquery.

Skalare Unterabfrage

```
SELECT * FROM artikel
WHERE einkaufspreis > (
    SELECT AVG(einkaufspreis) FROM artikel
);
```

=, >, <, >=, <=

Gibt nur einen Wert zurück!

Nicht-skalare Unterabfrage

```
SELECT Name FROM Mitarbeiter
WHERE AbteilungsID IN (
    SELECT AbteilungsID FROM Abteilungen
    WHERE Standort = 'Zürich'
);
```

NOT, EXISTS

Gibt temp. Tabelle zurück!

In SELECT, FROM, JOIN, WHERE, HAVING Klauseln möglich!

Scalar Subquery

A *scalar* subquery returns only **one column** with **only one row**. There are many operators that we can use to compare a column to a subquery. However, some of them can only be used with scalar subqueries: `, , , ,` and `.` If we use one of these operators, our subquery must be *scalar*. `=`

`> >= < <=`

EXAMPLE: Let's say we have a customer who wants to travel from Paris to Bariloche. Before he buys the ticket, he wants to see if there are any places where the ticket is cheaper. The following query finds these cities:

```
SELECT city_destination, ticket_price, travel_time, transportation FROM
one_way_ticket
WHERE ticket_price < (
    SELECT ticket_price FROM one_way_ticket
    WHERE city_destination = 'Bariloche' AND city_origin = 'Paris'
)
AND city_origin = 'Paris';
```

Here, too, the subquery is executed first; their result (the price of a ticket Paris-Bariloche, i.e. \$970) is compared with the column in the outer query. So we get all records in with a value less than \$970. The result of the query is shown below:

ticket_price one_way_ticket

City_Destination	Ticket_Price	Travel_Time	Transportation
Florianopolis	830.00	11hr 30min	Air

Non-scalar subquery

Here's an example of a *non-scalar* query with a Subselect: SELECT

```
use subselect;
SELECT name, age, country
FROM users
WHERE country IN
( -- hier beginnt Subquery:
    SELECT name FROM country WHERE region = 'Europa'
)
```

This example retrieves the names and ages of users who live in European countries from the table. The subselect is used to find the countries in Europe by running a query on the table and looking for the "" entry in the column. users countries Europe region

The result of the *non-scalar* subquery is a **list of countries**, which are then used as a condition for the clause in the main query to filter the records that satisfy the condition: **IN** or **NOT IN** for all others.

DB Backup

Databases are crucial for web hosting and business software because websites and company operations rely on the availability and completeness of the data stored in them. Websites pull necessary info from databases to display pages correctly, often using scripting languages. Similarly, a company's IT infrastructure uses databases to store and retrieve info, like personnel, financial, or customer data. If a database fails or loses data, websites can go offline, applications can stop working, and customer data can become incomplete or lost. This can lead to extra work and loss of customer trust.

Most data loss isn't due to external attacks but hardware failures or user errors. Even top-notch security software can't prevent this, so regular data backups are essential to prevent irreversible data loss.

Ways to Back Up Databases

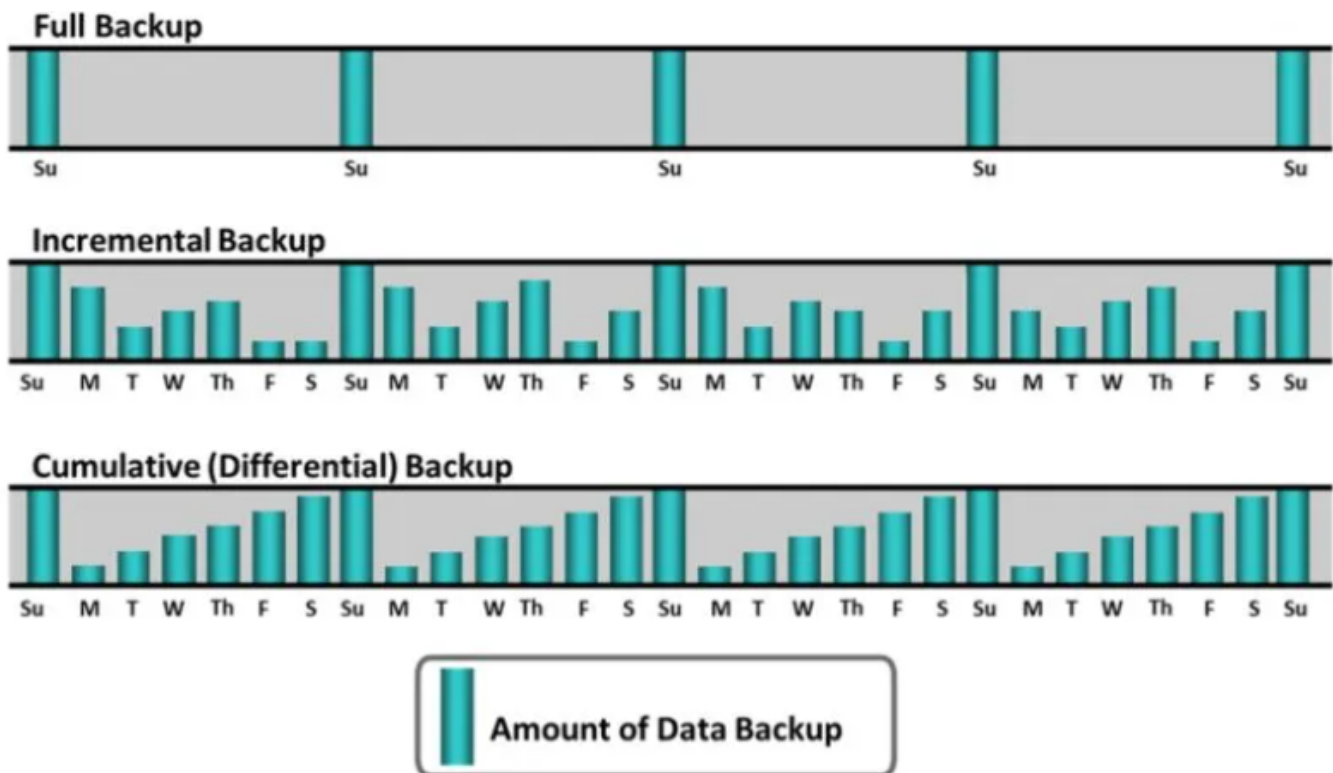
To prevent data loss, we should back up databases to external storage. These backups can restore the database to its previous state if data is lost.

Types of Backups

1. **Online Backups:** Created without shutting down the database. Changes are recorded in a separate area during the backup and then added to the database file after the backup is done.
2. **Offline Backups:** The database is shut down during the backup. This method is simple but makes the database unavailable during the backup. Best done at night or during low traffic times.

Backup Methods

1. **Full Backup:** Backs up all data and structures every time. It provides a complete snapshot but requires a lot of storage space.
2. **Differential Backup:** Starts with a full backup, then only backs up files that have changed or been added since the last full backup. It saves storage space compared to full backups but still backs up changed files each time. To restore, we need the last full backup plus the latest differential backup.
3. **Incremental Backup:** Also starts with a full backup, then only backs up files changed since the last backup (full or incremental). It saves the most storage space but requires all backups (from the full backup to the desired version) to restore files.



There are various ways to back up database systems like SQL databases or Microsoft Access. The best method depends on the specific needs of the user or company. However, doing backups infrequently to save storage space is a bad idea. Backup copies should be stored safely on external media like hard drives, ideally in a separate location to protect against theft and fire. Also, encrypt the backups so that, if stolen, they can't be used by unauthorized people.

Create Backups

Once we've chosen a backup method, the next step is deciding how to execute it. Here are some tools and options for backing up databases like SQL:

1. **MySQLDump:** If we have shell access, we can use MySQL's built-in backup function with the `mysqldump` command. This is the fastest backup method, but not all hosts allow it.
2. **phpMyAdmin:** This tool lets we easily export the desired database in formats like SQL. However, large databases might cause the PHP script to time out, and importing backups larger than 2 MB might not work.
3. **BigDump:** Complements phpMyAdmin by restoring backups of any size. However, it doesn't have its own backup function.
4. **HeidiSQL:** A Windows backup tool that doesn't rely on PHP, so it handles large backups well. It's similar to phpMyAdmin but doesn't automate the backup process.
5. **Mariabackup:** An open-source tool from MariaDB for online physical backups of InnoDB, Aria, and MyISAM tables. Available for Linux and Windows.

Protect Our Databases

Databases are crucial for running companies smoothly and displaying websites correctly. Web servers use database info to show websites, and network applications often depend on databases. Databases also store sensitive info like addresses, account details, and phone numbers.

Because of their importance, databases need proper security measures. They must be protected from external attacks, but also from internal issues like hardware failures or user errors. Regular backups are essential to prevent data loss and ensure long-term data security.

```
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO  
'backupuser'@'localhost' IDENTIFIED BY 'backup123';`
```