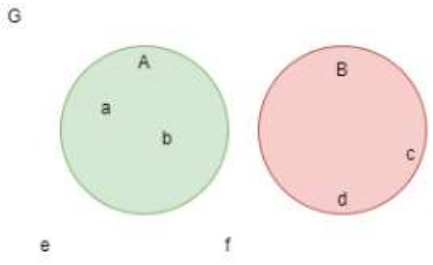
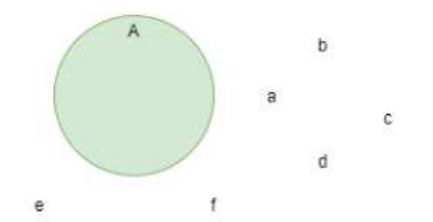
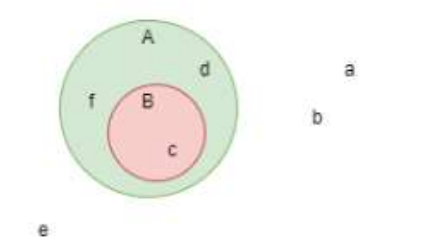
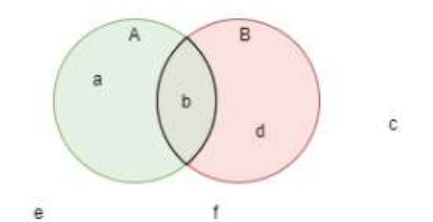
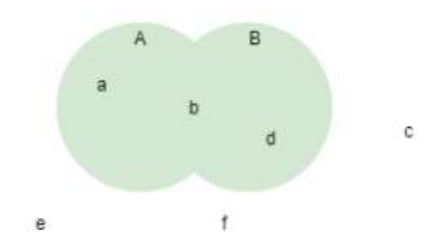
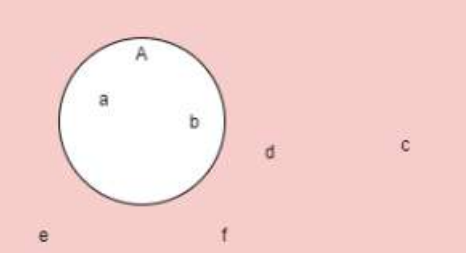
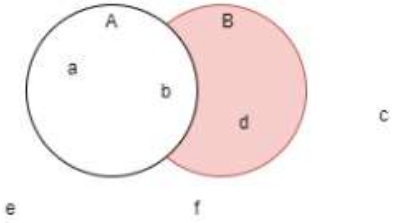


Zusammenfassung SQL 07.07.24

Mengenlehre

Gross- und Klein-Buchstaben und \in , \notin	Grossbuchstaben = Menge Kleinbuchstaben = Elemente welcher Menge zugewiesen sind	$G=\{a,b,c,d,e,f\}$, $A=\{a,b\}$, $B=\{c,d\}$	
$\{\}$ oder \emptyset	leere Menge	$A=\{\}$	
\subset , \subseteq	Teilmenge von	$B \subset A$, $x \in B$ auch $x \in A$	
\cap	Schnittmenge	$A \cap B$, $x \in B$ und $x \in A$	
\cup	Vereinigungsmenge	$A \cup B$, $x \in A$ oder $x \in B$	

X^c	Komplementärmenge e	alle Elemente, die nicht in der Menge X enthalten sind, $x \in G$ und $x \notin A$	
\setminus	Differenzmenge	$B \setminus A = x \in B$ und $x \notin A$	

Beziehungs-Constraints

Primary Key Constraint (Primärschlüssel):

- Sichert die Eindeutigkeit jeder Zeile in einer Tabelle.
- Eine Tabelle kann nur einen Primärschlüssel haben.

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100)
);
```

Foreign Key Constraint (Fremdschlüssel):

- Verbindet eine Spalte oder eine Gruppe von Spalten in einer Tabelle mit einer Spalte oder einer Gruppe von Spalten in einer anderen Tabelle.
- Sichert referentielle Integrität zwischen den Tabellen.

```
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Unique Constraint (Eindeutigkeitsbedingung):

- Gewährleistet, dass alle Werte in einer Spalte oder einer Gruppe von Spalten eindeutig sind.
- Unterschied zum Primärschlüssel: Eine Tabelle kann mehrere Unique Constraints haben und diese Spalten dürfen NULL-Werte enthalten.

```
CREATE TABLE employees (  
    employee_id INT PRIMARY KEY,  
    email VARCHAR(100) UNIQUE  
);
```

Check Constraint (Bedingung Constraint):

- Legt eine Bedingung fest, die alle Werte in einer Spalte erfüllen müssen.

```
CREATE TABLE products (  
    product_id INT PRIMARY KEY,  
    price DECIMAL(10, 2),  
    CHECK (price > 0)  
);
```

Not Null Constraint:

- Stellt sicher, dass eine Spalte keine NULL-Werte enthält.

```
CREATE TABLE orders (  
    order_id INT PRIMARY KEY,  
    order_date DATE NOT NULL  
);
```

Default Constraint:

- Legt einen Standardwert für eine Spalte fest, der verwendet wird, wenn kein Wert angegeben wird.

```
CREATE TABLE accounts (  
    account_id INT PRIMARY KEY,  
    balance DECIMAL(10, 2) DEFAULT 0.00  
);
```

1. Eindeutigkeit/Datenkonsistenz: Jeder Datensatz in der Datenbank sollte eindeutig identifizierbar sein, um zu verhindern, dass Daten doppelt eingegeben werden (Redundanzen).

2. Referenzielle Integrität: Wenn eine Tabelle Beziehungen zu anderen Tabellen hat, sollten die Beziehungen immer konsistent bleiben. Das bedeutet, dass z.B. eine Verknüpfung zwischen einer Bestellung und einem Kunden nur dann bestehen kann, wenn der Kunde auch tatsächlich in der Kundentabelle vorhanden ist.

3. Datentypen: Die Daten sollten in der Datenbank in den korrekten Datentypen gespeichert werden, um sicherzustellen, dass sie korrekt behandelt werden können. Beispielsweise sollte eine Telefonnummer in einem Feld vom Typ "String" (Zeichenfolge) und nicht vom Typ "Integer" (Ganzzahl) gespeichert werden.

4. Datenbeschränkungen: Datenbeschränkungen stellen sicher, dass die Daten in der Datenbank gültig sind. Zum Beispiel kann eine Datenbank so eingerichtet werden, dass nur positive Zahlen in einem bestimmten Feld eingegeben werden dürfen, oder dass eine E-Mail-Adresse nur in einem bestimmten Format eingegeben werden kann.

5. Validierung: Vor dem Einfügen von Daten in die Datenbank sollte eine Validierung durchgeführt werden, um sicherzustellen, dass die Daten den Anforderungen der Datenbank entsprechen.

DELETE	Regel	Bedeutung
ON DELETE	NO ACTION	Ein DELETE in der Primärtabelle kann nur ausgeführt werden, wenn in keiner Detailtabelle ein Fremdschlüssel mit dem entsprechenden Wert existiert. <i>Dies ist das Standardverfahren, wenn nichts angegeben wird.</i>
	CASCADE	Ein DELETE in der Primärtabelle führt auch zu einem Löschen der entsprechenden Datensätze in der Fremdschlüsseltabelle. Achtung mit dieser Regel, evtl. werden unbeabsichtigt Daten gelöscht!
	SET NULL	Bei einem DELETE in der Primärtabelle werden die entsprechenden Datensätze in der Fremdschlüsseltabelle auf NULL gesetzt. Hinweis: Diese Einstellung geht nur bei c:m(c) und c:c Beziehungen, wenn also beim FK NULL erlaubt ist!
	DEFAULT	Bei einem DELETE in der Primärtabelle werden die entsprechenden Datensätze in der Fremdschlüsseltabelle auf den DEFAULT-Wert gesetzt, falls einer definiert worden ist. Ansonsten werden sie auf NULL gesetzt (bei c:m(c) und c:c Beziehungen)

Damit NICHT das passiert:

Inkonsistente / konsistente Daten

kunden

<u>idKunde</u>	name	postleitzahl	ortName	arbeitgeber
1	Schmitt	10000	Musterhausen	Bäckerei Zimmermann GmbH
2	Müller	10000	Musterausen	Schlüter & Co KG
	Müller	79132	Coburg	Bäckerei Zimmermann

Sondern das:

kunden

<u>idKunde</u>	name	postleitzahl	arbeitgeberFK
1	Schmitt	10000	1
2	Müller	10000	14
3	Maier	79312	7

orte

<u>postleitzahl</u>	name
10000	Musterhausen
79098	Freiburg

arbeitgeber

<u>idArbeitgeber</u>	name
1	Bäckerei Zimmermann
7	Hug GmbH
14	ForSi

Referenzielle Integrität

Referenzielle Integrität stellt sicher, dass Beziehungen zwischen Tabellen in einer Datenbank konsistent bleiben. Das bedeutet, dass Fremdschlüssel in einer Tabelle immer auf gültige Primärschlüssel in einer anderen Tabelle verweisen müssen.

Aggregationsfunktionen

Aggregatsfunktionen sind spezielle Funktionen in SQL, die eine Berechnung über eine Menge von Werten durchführen und einen einzigen Wert zurückgeben. Zu den wichtigsten Aggregatsfunktionen gehören:

- **SUM():** Berechnet die Summe einer numerischen Spalte.
z.b :
SELECT SUM(*) FROM customers;
- **AVG():** Berechnet den Durchschnittswert einer numerischen Spalte.

- **COUNT():** Gibt die Anzahl der Zeilen zurück, die einer Bedingung entsprechen.
 - **MAX():** Gibt den größten Wert einer Spalte zurück.
 - **MIN():** Gibt den kleinsten Wert einer Spalte zurück. M employees;
-

Datensicherung & Bulk Import

Logische Backups:

- **Datenexport** (Exportiert Daten in einem lesbaren Format)
- **SQL-Dumps** (Erzeugt Skripte, die Datenbankstrukturen und -inhalte enthalten)
- **Plattformunabhängig** (Kann auf verschiedenen Datenbank Plattformen wiederhergestellt werden)
- **Einfach zu manipulieren** (Daten können vor der Wiederherstellung bearbeitet werden)
- **Langsamer** (Erstellung und Wiederherstellung können zeitaufwendig sein)

Physische Backups:

- Dateisystem-Sicherung (Kopiert physische Dateien der Datenbank)
- Schneller (Schnelle Erstellung und Wiederherstellung)
- Speicherplatz Intensiv (Erfordert mehr Speicherplatz für die Sicherung)
- Konsistenzsicherung (Erhält die physische Konsistenz der Datenbank)
- Plattformunabhängig (Kann nur auf derselben Datenbankplattform wiederhergestellt werden)

Möglichkeiten zur Sicherung von Datenbanken

Sicherheitskopien erstellen:

- Um Datenverluste zu verhindern, sollten Sicherheitskopien (Backups) der Datenbanken auf externen Medien erstellt werden. Diese Backups ermöglichen die Wiederherstellung des Datenbankzustands zum Zeitpunkt der Sicherung.

Online- und Offline-Backups:

- **Online-Backups:** Werden erstellt, ohne die Datenbank herunterzufahren. Änderungen während des Sicherungs Prozesses werden in einem separaten Bereich gespeichert und später integriert.
- **Offline-Backups:** Die Datenbank wird für die Sicherung heruntergefahren. Dieses Verfahren ist einfacher, aber während des Backups sind Anwendungen und Websites nicht verfügbar. Daher sollte es zu Zeiten geringen Datenverkehrs durchgeführt werden.

Arten der Datensicherung:

- **Voll-Backup:** Alle Daten werden vollständig gesichert. Erfordert viel Speicherplatz, aber nur das letzte Vollbackup wird zur Wiederherstellung benötigt.
 - **Differentielles Backup:** Nach einem Voll Backup werden nur die seitdem geänderten Daten gesichert. Spart Speicherplatz, da nur Änderungen gesichert werden. Zur Wiederherstellung benötigt man das letzte Voll- und das letzte differenzielle Backup.
 - **Inkrementelles Backup:** Sichert nur die seit der letzten Sicherung geänderten Daten. Spart noch mehr Speicherplatz, da nur die Änderungen seit dem letzten Backup gesichert werden. Zur Wiederherstellung sind alle inkrementellen Backups seit dem letzten Vollbackup notwendig.
-

Konsistenz und ref.Integrität

- *Welche Schwierigkeiten beim Einfügen von Daten ergeben sich, wenn z.B. der FK Vorgesetzter als Constraint definiert ist? (Tipp: ref. Integrität)*

Beim Einfügen von Daten in eine Tabelle mit dem FK Vorgesetzter als Constraint kann man Schwierigkeiten bekommen, wenn der referenzierte Vorgesetzte noch nicht existiert. Das verletzt die referenzielle Integrität und verhindert das Einfügen. Man muss sicher gehen, dass alle referenzierten Vorgesetzten vorher schon in der Tabelle existieren.

- *Warum ist der Wert NULL in der tbl_Hierarchie nicht zulässig? (Tipp: Kardinalität)*

Ein NULL-Wert in der tbl_Hierarchie ist nicht erlaubt, da die Kardinalität eine Hierarchie verlangt. Jede Position muss eindeutig einer anderen Position oder dem obersten zugeordnet sein, um die Struktur der Hierarchie zu bewahren. Ein NULL-Wert würde diese Zuordnung verstossen.

- *Wann muss eine Hierarchie-Tabelle anstelle einer rekursiven Beziehung eingesetzt werden? (Tipp: Chefs)*

Eine Hierarchie-Tabelle wird anstelle einer rekursiven Beziehung eingesetzt, wenn die Anzahl der Hierarchieebenen fest ist. Bei grossen Organisationen kann eine feste Struktur übersichtlicher und performanter sein als rekursive Abfragen.

- *Ref. Integrität: Was ist das? Machen Sie ein Beispiel dazu!* Die Definition von referentieller Integrität ist, dass Beziehungen zwischen Tabellen in einer relationalen Datenbank konsistent bleiben. Ein Fremdschlüssel in einer Tabelle verweist immer auf einen gültigen Eintrag in einer anderen Tabelle.
- *Welche Constraints kann eine Beziehung haben? (Tipp: Mehr als eine!)* Sie kann die Beziehung NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY und CHECK haben
- *Was ist der Unterschied zwischen LEFT JOIN und RIGHT JOIN?* Left Join gibt alle Datensätze aus der linken Tabelle und die Passenden aus der rechten zurück, wobei right join das gleiche macht, aber aus der rechten Tabelle.
- *Wie wird eine 1:1-Beziehung und eine c:m-Beziehung umgesetzt? Warum?* Durch einen Fremdschlüssel, der auch als UNIQUE und NOT NULL gilt. Das durch eine Zwischentabelle, welche beide PKs als FKs enthält.
- *Was ist der Nachteil, wenn eine Beziehung nur mit Primär- und Fremdschlüssel definiert werden, d.h. ohne die Constraint-Anweisung?* Es gibt dann keine weiteren Sicherungen bezüglich der Integrität. Es könnten also doppelte Einträge möglich sein oder Null Werte in Fremdschlüssel spalten.
- *Welche Folge hat ein Eintrag eines Fremdschlüssels Wertes, der als ID-Wert in der verbundenen Tabelle nicht vorhanden ist?* a) mit Constraint-Anweisung auf dem FS Es gibt einen Fehler aus b) ohne Constraint-Anweisung auf dem FS Er kann eingefügt werden auch wenn der FK nicht existiert