Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CHAI
Humanities-Centered AI

Databases

Malte Luttermann – Institute for Humanities-Centered Artificial Intelligence (CHAI)

November 27, 2025

Version from November 27, 2025

Topics for Today

- Structured Query Language (SQL)

Acknowledgement

- Some of the upcoming content is taken from the following lecture (and corresponding exercises) and has been translated and partially modified:
  - PD Dr. habil. Özgür L. Özçep: »Datenbanken«

SQL – Comprehension Questions I

Are the following statements correct?

- »When using the `SUM` aggregate function in the `SELECT` clause, the output is always exactly one row.«
- »Views in SQL are shortcuts for queries.«
- »A `HAVING` clause must always be specified with `GROUP BY`.«

SQL – Comprehension Questions I

Are the following statements correct?

- »When using the `SUM` aggregate function in the `SELECT` clause, the output is always exactly one row.«
  - Incorrect. When grouping the data, it is possible to calculate a sum for each group from the aggregated data.
- »Views in SQL are shortcuts for queries.«
- »A `HAVING` clause must always be specified with `GROUP BY`.«

SQL – Comprehension Questions I

Are the following statements correct?

- »When using the `SUM` aggregate function in the `SELECT` clause, the output is always exactly one row.«
  - Incorrect. When grouping the data, it is possible to calculate a sum for each group from the aggregated data.
- »Views in SQL are shortcuts for queries.«
  - Correct. Views enable you to assign names to query modules, allowing you to access them more quickly.
- »A `HAVING` clause must always be specified with `GROUP BY`.«

SQL – Comprehension Questions I

Are the following statements correct?

- »When using the `SUM` aggregate function in the `SELECT` clause, the output is always exactly one row.«
  - Incorrect. When grouping the data, it is possible to calculate a sum for each group from the aggregated data.
- »Views in SQL are shortcuts for queries.«
  - Correct. Views enable you to assign names to query modules, allowing you to access them more quickly.
- »A `HAVING` clause must always be specified with `GROUP BY`.«
  - Incorrect. An unspecified `HAVING` clause means »true«.

SQL Queries

Let the following relational schemas be given:

- $Supplier = \{\underline{SupplierId}, Company, ContactPerson, Street, City, Telephone\}$
- $Category = \{\underline{CategoryId}, CategoryName, Description\}$
- $Order = \{\underline{OrderId}, OrderDate, CustomerId\}$
- $OrderDetails = \{\underline{OrderId}, \underline{ItemId}, Quantity\}$
- $Customer = \{\underline{CustomerId}, Company, ContactPerson, Street, City, Telephone\}$
- $Item = \{\underline{ItemId}, ItemName, SupplierId, CategoryId, DeliveryUnit\}$
- $ItemPrice = \{\underline{ItemId}, Price\}$

Formulate a query in SQL to determine the total order volume (i.e., the value of all orders placed) for each customer. Only the *CustomerId* and the order volume are of interest here. Sort the results in descending order by *CustomerId*.

SQL Queries

Formulate a query in SQL to determine the total order volume (i.e., the value of all orders placed) for each customer. Only the *CustomerId* and the order volume are of interest here. Sort the results in descending order by *CustomerId*.

```sql
SELECT CustomerId, SUM(Price * Quantity) AS OrderVolume
FROM ItemPrice I, Order O, OrderDetails D
WHERE O.OrderId = D.OrderId
  AND I.ItemId = D.ItemId
GROUP BY CustomerId
ORDER BY CustomerId DESC;
```

SQL – Comprehension Questions II

Are the following statements correct?

- »Using a view, the result of an SQL query can be saved.«
- »All SQL views may be modified.«
- »`DELETE ... FROM` may result in a violation of integrity.«

SQL – Comprehension Questions II

Are the following statements correct?

- »Using a view, the result of an SQL query can be saved.«
  - Incorrect. A view just gives an SQL query a name so it can be used again later. The query is re-evaluated each time it is used.
- »All SQL views may be modified.«
- »`DELETE ... FROM` may result in a violation of integrity.«

SQL – Comprehension Questions II

Are the following statements correct?

- »Using a view, the result of an SQL query can be saved.«
  - Incorrect. A view just gives an SQL query a name so it can be used again later. The query is re-evaluated each time it is used.
- »All SQL views may be modified.«
  - Incorrect. Some views (e.g., with a join) must not be changed.
- »`DELETE ... FROM` may result in a violation of integrity.«

SQL – Comprehension Questions II

Are the following statements correct?

- »Using a view, the result of an SQL query can be saved.«
  - Incorrect. A view just gives an SQL query a name so it can be used again later. The query is re-evaluated each time it is used.
- »All SQL views may be modified.«
  - Incorrect. Some views (e.g., with a join) must not be changed.
- »`DELETE ... FROM` may result in a violation of integrity.«
  - Correct. A change occurs in the database, which may result in an integrity violation.

SQL Queries

Let the following relational schemas be given:

- $Supplier = \{\underline{SupplierId}, Company, ContactPerson, Street, City, Telephone\}$
- $Category = \{\underline{CategoryId}, CategoryName, Description\}$
- $Order = \{\underline{OrderId}, OrderDate, CustomerId\}$
- $OrderDetails = \{\underline{OrderId}, \underline{ItemId}, Quantity\}$
- $Customer = \{\underline{CustomerId}, Company, ContactPerson, Street, City, Telephone\}$
- $Item = \{\underline{ItemId}, ItemName, SupplierId, CategoryId, DeliveryUnit\}$
- $ItemPrice = \{\underline{ItemId}, Price\}$

Formulate a query in SQL to determine all suppliers (wanted: *Company*) located in Munich and store the result in a persistent table.

SQL Queries

Formulate a query in SQL to determine all suppliers (wanted: *Company*) located in Munich and store the result in a persistent table.

```sql
CREATE TABLE MunichSuppliers AS
SELECT Company
FROM Supplier
WHERE City = 'Munich';
```

SQL Queries

Let the following relational schemas be given:

- $Customer = \{\underline{CustomerId}, FirstName, LastName\}$
- $Drink = \{\underline{DrinkId}, DrinkName, Price, Type\}$
- $Order = \{\underline{OrderId}, CustomerId\}$
- $OrderDetails = \{\underline{OrderId}, \underline{DrinkId}, Quantity\}$

Formulate a query in SQL to create a view called *Beers* that specifies the instances of the attributes *DrinkName* and *Price* for all drinks of the type »Beer«.

SQL Queries

Formulate a query in SQL to create a view called *Beers* that specifies the instances of the attributes *DrinkName* and *Price* for all drinks of the type »Beer«.

```sql
CREATE VIEW Beers AS
SELECT DrinkName, Price
FROM Drink
WHERE Type = 'Beer';
```

SQL Queries

Let the following relational schemas be given:

- $Customer = \{\underline{CustomerId}, FirstName, LastName\}$
- $Drink = \{\underline{DrinkId}, DrinkName, Price, Type\}$
- $Order = \{\underline{OrderId}, CustomerId\}$
- $OrderDetails = \{\underline{OrderId}, \underline{DrinkId}, Quantity\}$

Formulate a query in SQL to create a view called *DrinkRevenue* that specifies the instances for the *DrinkId* attribute and instances for the output attribute *TotalRevenue* (i.e., how much money was earned with each drink).

SQL Queries

Formulate a query in SQL to create a view called *DrinkRevenue* that specifies the instances for the *DrinkId* attribute and instances for the output attribute *TotalRevenue* (i.e., how much money was earned with each drink).

```sql
CREATE VIEW DrinkRevenue AS
SELECT DrinkId, SUM(Price * Quantity) AS TotalRevenue
FROM Drink D, OrderDetails O
WHERE D.DrinkId = O.DrinkId
GROUP BY DrinkId;
```

SQL Queries

Let the following relational schemas be given:

- $Customer = \{\underline{CustomerId}, FirstName, LastName\}$
- $Drink = \{\underline{DrinkId}, DrinkName, Price, Type\}$
- $Order = \{\underline{OrderId}, CustomerId\}$
- $OrderDetails = \{\underline{OrderId}, \underline{DrinkId}, Quantity\}$

Adjust the previous query such that only drinks that generated at least 20 euros in sales are displayed.

SQL Queries

Adjust the previous query such that only drinks that generated at least 20 euros in sales are displayed.

```sql
CREATE VIEW DrinkRevenue AS
SELECT DrinkId, SUM(Price * Quantity) AS TotalRevenue
FROM Drink D, OrderDetails O
WHERE D.DrinkId = O.DrinkId
GROUP BY DrinkId
HAVING TotalRevenue >= 20;
```

SQL Queries

Let the following relational schemas be given:

- $Customer = \{\underline{CustomerId}, FirstName, LastName\}$
- $Drink = \{\underline{DrinkId}, DrinkName, Price, Type\}$
- $Order = \{\underline{OrderId}, CustomerId\}$
- $OrderDetails = \{\underline{OrderId}, \underline{DrinkId}, Quantity\}$

Add a constraint called *PriceCap* to the *Price* column from the table *Drink* such that no drink is allowed to cost more than 7 euros.

SQL Queries

Add a constraint called *PriceCap* to the *Price* column from the table *Drink* such that no drink is allowed to cost more than 7 euros.

```
1 ALTER TABLE Drink
2 ADD CONSTRAINT PriceCap
3 CHECK (Drink.Price <= 7);
```

SQL Queries

Let the following relational schemas be given:

- $Customer = \{\underline{CustomerId}, FirstName, LastName\}$
- $Drink = \{\underline{DrinkId}, DrinkName, Price, Type\}$
- $Order = \{\underline{OrderId}, CustomerId\}$
- $OrderDetails = \{\underline{OrderId}, \underline{DrinkId}, Quantity\}$

Add a constraint that ensures that no drink named »Becks Lemon« is assigned to the type »Beer«.

SQL Queries

Add a constraint that ensures that no drink named »Becks Lemon« is assigned to the type »Beer«.

```sql
ALTER TABLE Drink
ADD CONSTRAINT BecksLemon
CHECK (
  NOT (DrinkName, Type) in VALUES(('Becks Lemon', 'Beer'))
);
```