Prof. DI Dr. Erich Gams

Datenbanken

Views MySQL

informationssysteme htl-wels

Übersicht ☛ Was lernen wir?

- Kurze Wiederholung
- Entity-Relationship-Modell und Erweiterungen
- Beziehungen mit den verschiedenen Kardinalitäten
- Notationen
- …und natürlich Übungen

Was ist eine View?

- Mit CREATE VIEW kann eine "Sicht" auf die Datenbank erstellt werden.
- **Eine View ist eine SQL SELECT Abfrage mit einem Namen.**
- Die View wird in der Datenbank gespeichert.

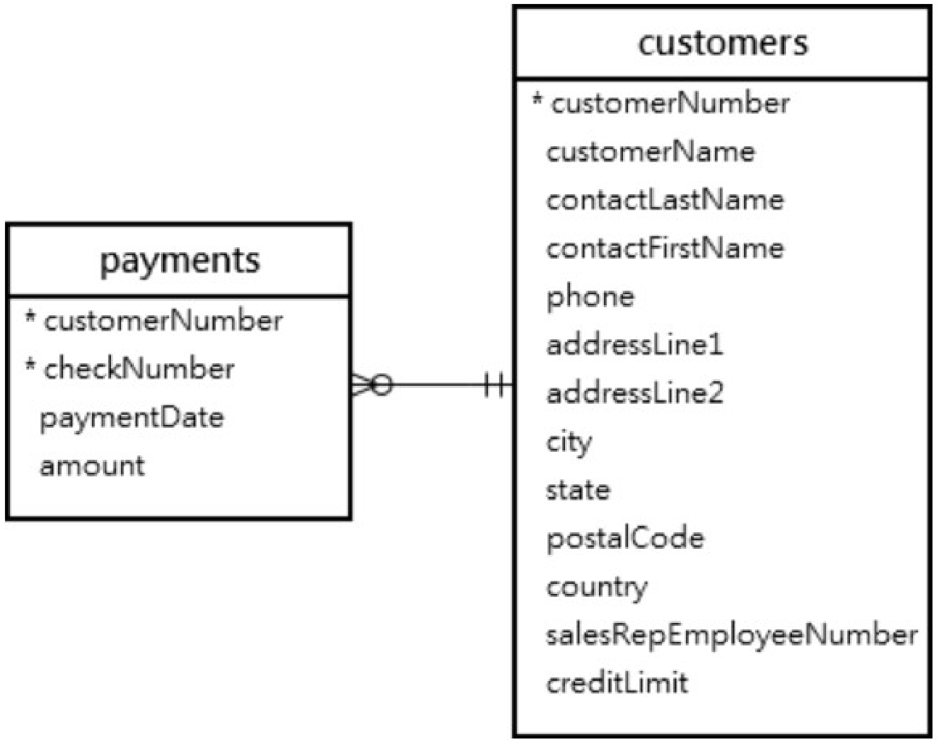
Pros und Cons

- Man erspart sich jedes Mal eine neue Abfrage zu machen.
- Man hat die Tabelle für Abfragen zur Verfügung wie man sie braucht.
- Aktualisierungen?

Characteristics

- A view does not physically store the data.
- When you issue the SELECT statement against the view, MySQL executes the underlying query specified in the view’s definition and returns the result set.
- A view is also referred to as a virtual table.

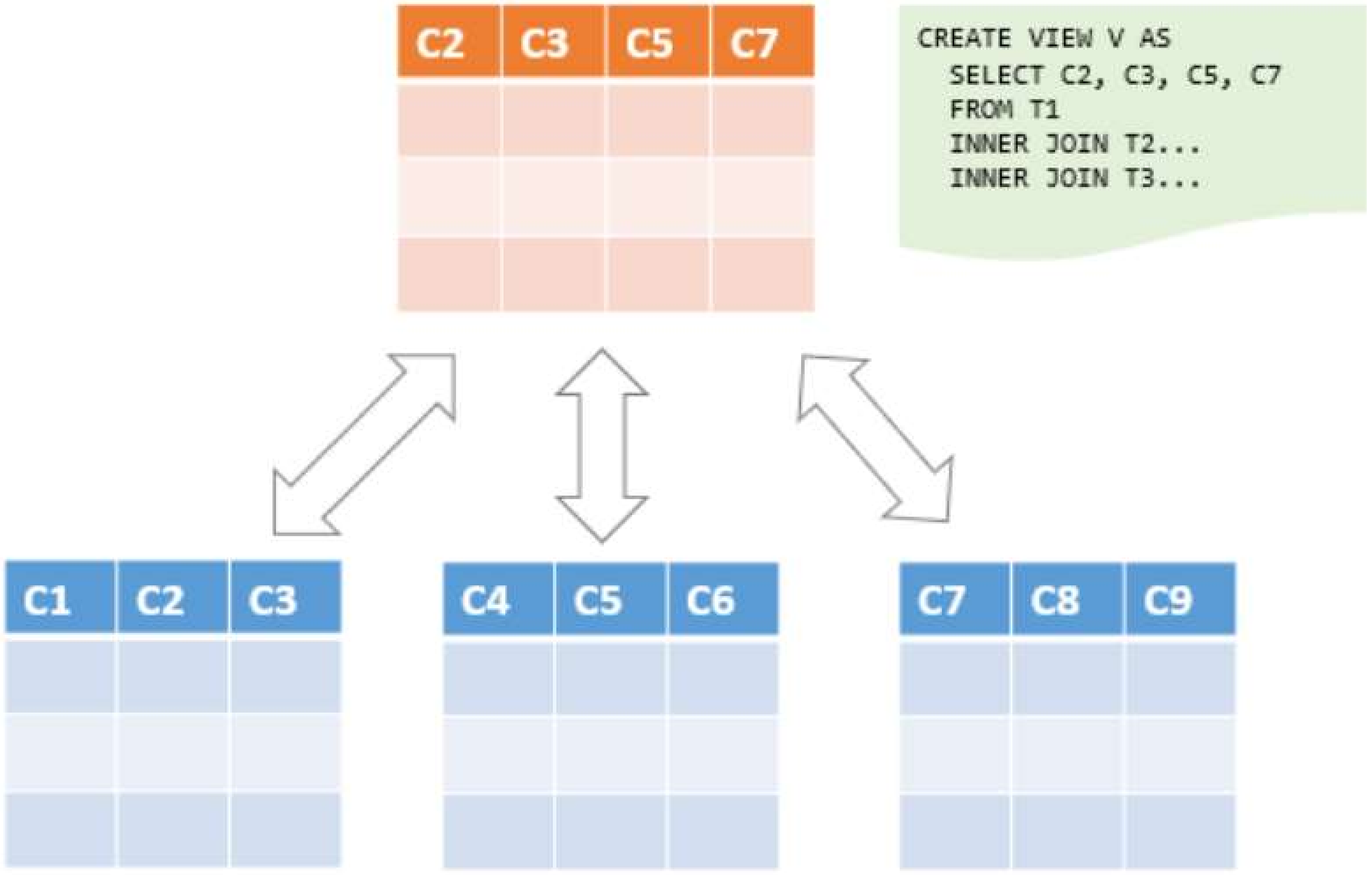
Beispiel


payments
* customerNumber
* checkNumber
paymentDate
amount
customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit

Join

```
SELECT
    customerName,
    checkNumber,
    paymentDate,
    amount
FROM
    customers
INNER JOIN
    payments USING (customerNumber);
```

MySQL: If the join condition uses the equal operator (=) and the column names in both tables used for matching are the same, you can use the USING clause instead



```
CREATE VIEW V AS
  SELECT C2, C3, C5, C7
  FROM T1
  INNER JOIN T2...
  INNER JOIN T3...
```

Sonderfall: View ohne Tabelle

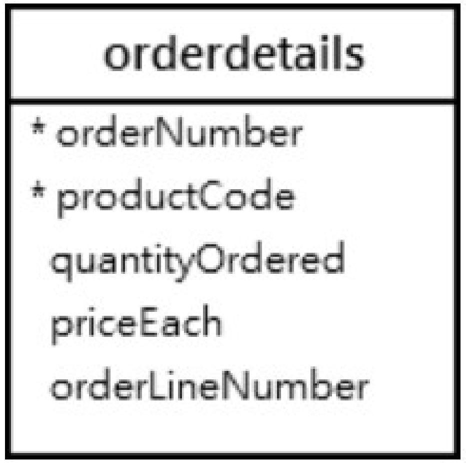
```
CREATE VIEW daysofweek (day) AS
    SELECT 'Mon'
    UNION
    SELECT 'Tue'
    UNION
    SELECT 'Web'
    UNION
    SELECT 'Thu'
    UNION
    SELECT 'Fri'
    UNION
    SELECT 'Sat'
    UNION
    SELECT 'Sun';
```

Advantages of MySQL Views

- Simplify complex queries
  - reference to the view by using a simple SELECT statement instead of typing the query all over again.
- Make the business logic consistent
  - query that has complex business logic.
  - logic consistent across queries, a view hides the complexity.
- Add extra security layers
  - e.g. grant privilegies of certain users the view, not the entire table employees.
- Enable backward compatibility
  - Suppose, you want to normalize a big table into many smaller ones. And you don't want to impact the current applications that reference the table.

Example

```
CREATE VIEW salePerOrder AS
    SELECT
        orderNumber,
        SUM(quantityOrdered * priceEach) total
    FROM
        orderDetails
    GROUP by orderNumber
    ORDER BY total DESC;
```



orderdetails

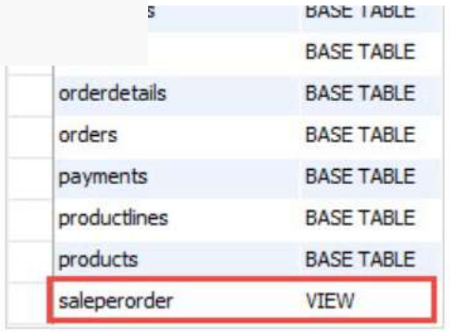
* orderNumber
* productCode

quantityOrdered

priceEach

orderLineNumber

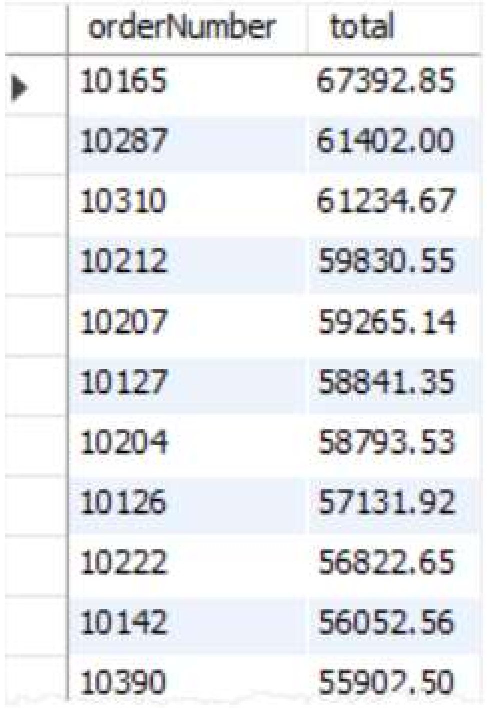
```
SHOW FULL TABLES;
```



s	BASE TABLE
	BASE TABLE
orderdetails	BASE TABLE
orders	BASE TABLE
payments	BASE TABLE
productlines	BASE TABLE
products	BASE TABLE
saleperorder	VIEW

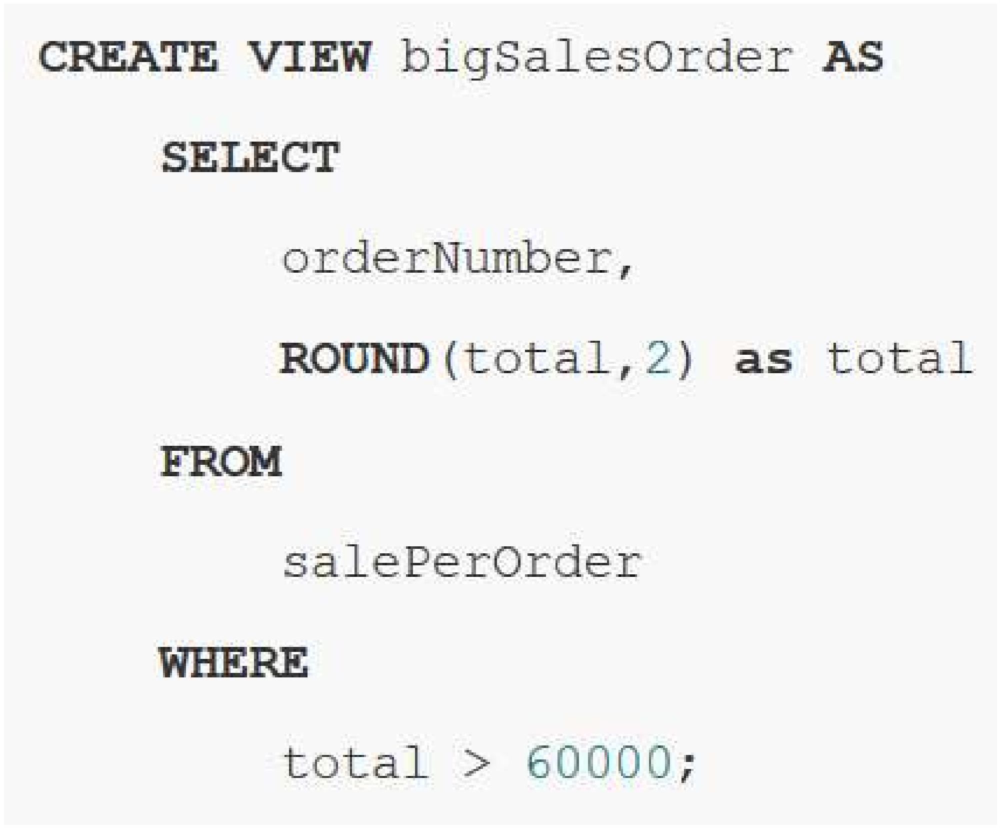
Simple Queries as a result….

```
SELECT * FROM salePerOrder;
```



orderNumber	total
10165	67392.85
10287	61402.00
10310	61234.67
10212	59830.55
10207	59265.14
10127	58841.35
10204	58793.53
10126	57131.92
10222	56822.65
10142	56052.56
10390	55902.50

Create a view on a view



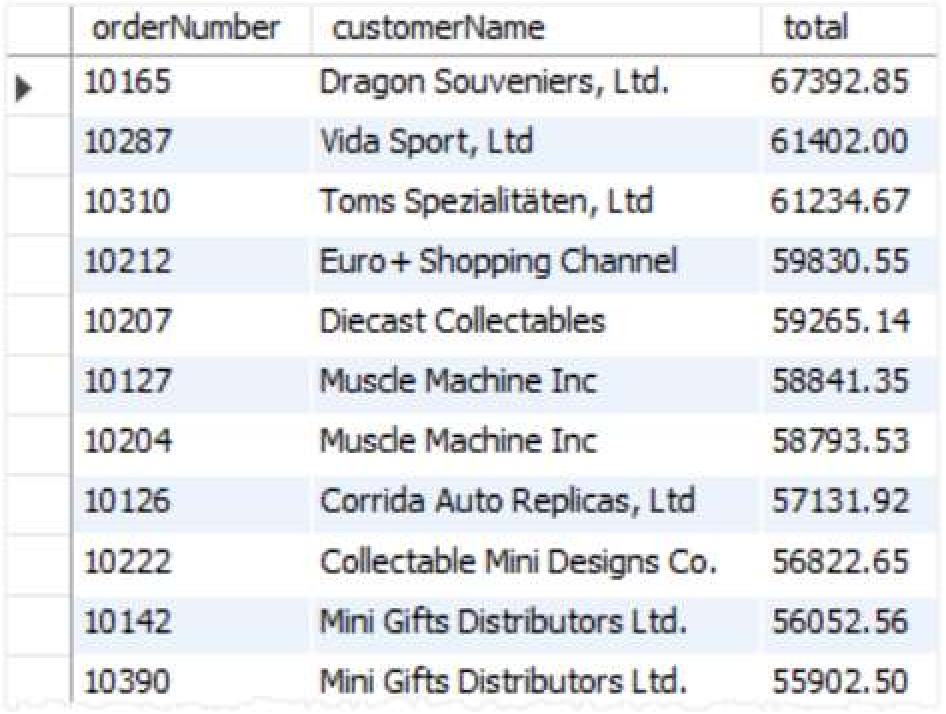
```
CREATE VIEW bigSalesOrder AS
    SELECT
        orderNumber,
        ROUND(total,2) as total
    FROM
        salePerOrder
    WHERE
        total > 60000;
```

View based on a Join

```
CREATE OR REPLACE VIEW customerOrders AS

SELECT

    orderNumber,

    customerName,

    SUM(quantityOrdered * priceEach) total

FROM

    orderDetails

INNER JOIN orders o USING (orderNumber)

INNER JOIN customers USING (customerNumber)

GROUP BY orderNumber;
```

Example query

```
SELECT * FROM customerOrders

ORDER BY total DESC;
```



orderNumber	customerName	total
10165	Dragon Souveniers, Ltd.	67392.85
10287	Vida Sport, Ltd	61402.00
10310	Toms Spezialitäten, Ltd	61234.67
10212	Euro+ Shopping Channel	59830.55
10207	Diecast Collectables	59265.14
10127	Muscle Machine Inc	58841.35
10204	Muscle Machine Inc	58793.53
10126	Corrida Auto Replicas, Ltd	57131.92
10222	Collectable Mini Designs Co.	56822.65
10142	Mini Gifts Distributors Ltd.	56052.56
10390	Mini Gifts Distributors Ltd.	55902.50

Create View with explicit columns

```
CREATE VIEW customerOrderStats (
   customerName ,
   orderCount
)
AS
    SELECT
        customerName,
        COUNT(orderNumber)
    FROM
        customers
            INNER JOIN
        orders USING (customerNumber)
    GROUP BY customerName;
```

Try with docker

- `docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag`
- `docker exec -i mymysql sh -c "exec mysql -u root -p mysql" < classicmodels.sql`

View Processing: ALGORITHM

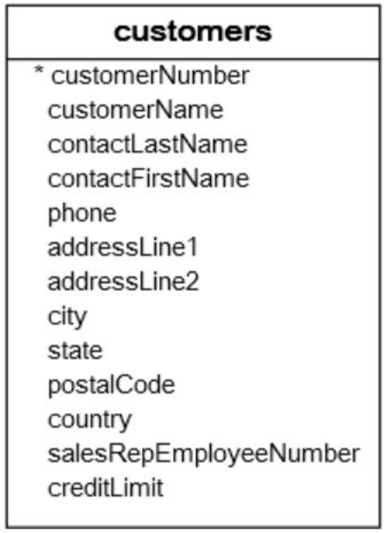
- The algorithm determines how MySQL process a view and can take one of three values MERGE, TEMPTABLE, and UNDEFINE.

MERGE

- When you query from a MERGE view, MySQL processes the following steps:
  - First, merge the input query with the SELECT statement in the view definition into a single query.
  - Then, execute the combined query to return the result set.

Merge Example

```
CREATE ALGORITHM=MERGE VIEW contactPersons(
    customerName,
    firstName,
    lastName,
    phone
) AS
SELECT
    customerName,
    contactFirstName,
    contactLastName,
    phone
FROM customers;
```



Merge Example

› You execute:

```
SELECT * FROM contactPersons
WHERE customerName LIKE '%Co%';
```

› MySQL executes:

```
SELECT
    customerName,
    contactFirstName,
    contactLastName,
    phone
FROM
    customers
WHERE
    customerName LIKE '%Co%';
```

TEMPTABLE and UNDEFINED

- TEMPTABLE view, MySQL performs these steps:
  - First, create a temporary table to store the result of the SELECT in the view definition.
  - Then, execute the input query against the temporary table.
- The UNDEFINED allows MySQL to choose either MERGE or TEMPTABLE.
  - MySQL prefers MERGE over TEMPTABLE if possible because MERGE is often more efficient than TEMPTABLE.

ALTER VIEW

› MySQL ALTER VIEW statement changes the definition of an existing view.

```
ALTER
    ALGORITHM=MERGE
VIEW salesOrders AS
    SELECT
        orderNumber,
        customerNumber,
        productCode,
        quantityOrdered,
        priceEach,
        status
    FROM
        orders
    INNER JOIN
        orderDetails USING (orderNumber);
```

```
SHOW CREATE VIEW salesorders;
```

MySQL Updatable Views

› INSERT or UPDATE statement to insert or update rows of the base table through the updatable view

› To create an updatable view, the SELECT statement that defines the view must not contain any of the following elements.

Exceptions

- Aggregate functions such as MIN, MAX, SUM, AVG, and COUNT.
- DISTINCT
- GROUP BY clause.
- HAVING clause.
- UNION or UNION ALL clause.
- Left join or outer join.
- Subquery in the SELECT clause or in the WHERE clause that refers to the table appeared in the FROM clause.
- Reference to non-updatable view in the FROM clause.
- Reference only to literal values.
- Multiple references to any column of the base table.

Example

```
CREATE VIEW officeInfo
 AS
   SELECT officeCode, phone, city
   FROM offices;
```

```
UPDATE officeInfo
SET
    phone = '+33 14 723 5555'
WHERE
    officeCode = 4;
```

Checking updatable view information

```
SELECT

    table_name,

    is_updatable

FROM

    information_schema.views

WHERE

    table_schema = 'classicmodels';
```

Removing rows through the view

```
CREATE VIEW LuxuryItems AS …..
```

```
DELETE FROM LuxuryItems
WHERE id = 3;
```

In the base table the tupel should also be deleted!

MySQL View & the WITH CHECK OPTION clause

- The WITH CHECK OPTION prevents a view from updating or inserting rows that are not visible through it.
- Whenever you update or insert a row of the base tables through a view, MySQL ensures that the insert or update operation is conformed with the definition of the view.

MySQL View & the WITH CHECK OPTION clause

CREATE OR REPLACE VIEW dummy AS

SELECT ….. WHERE name LIKE ‘%ei%’;

INSERT INTO dummy(name, …) VALUES (‘Wurst’,..);

› ‘Wurst’ is not visible through the view (because of the where clause)!

› Use: WHERE name LIKE ‘%ei%’ WITH CHECK OPTION;

Show all Views in the DB

```
SHOW FULL TABLES

WHERE table_type = 'VIEW';
```

```
SELECT

    table_name view_name

FROM

    information_schema.tables

WHERE

    table_type   = 'VIEW' AND

    table_schema = 'classicmodels';
```

Show create view statement

› SHOW CREATE VIEW [database_name].[view_name];

Rename/drop a view

```
RENAME TABLE <viewname>

TO <new_viewname>;


DROP VIEW [IF EXISTS] view_name1
[,view_name2]...;
```

Aufgabe

- Übungen in Moodle