



The lecture will start soon at
10:15



Aalto University
School of Business

MySQL for Data Analytics

Lecturer: Yong Liu

Contact me at: Yong.liu@aalto.fi

Objectives for class 3

- **HeidiSQL: Import csv file to MySQL table**
- **Export data to a database file (sql file)**
- **Key and index: foreign key**
- **Understanding the basics of Entity-Relationship Diagram (ERD)**
- **MySQL Keyword: Select**

Import CSV File Into MySQL Table

1. Download ‘Chile.csv’ from MyCourse

2. Open the csv file to check its structure.

What if the csv file is very large? R?

3. Create a new table in the DB with a structure that is consistent with the structure of the csv file.

4. Import the csv file to the new table:

Tools → Import CSV file

Import text file

Input file

Filename: \\home.org.aalto.fi\\liuy13\\data\\Desktop\\MySQL course csv data file.csv

Encoding: Let server/database decide (utf8)

Options

Ignore first 1 lines

☐ Low priority, avoid high server load

☐ Input file contains local formatted numbers, e.g. 1.234,56 in Germany

☐ Truncate destination table before import

Control characters

Fields terminated by ;

Fields enclosed by " ☒ optionally

Fields escaped by

Lines terminated by \r\n

Handling of duplicate rows

☒ INSERT (may throw errors)

☐ INSERT IGNORE (duplicates)

☐ REPLACE (duplicates)

Method

☒ Server parses file contents (LOAD DATA)

☐ Client parses file contents

Destination

Database: temp

Table: chile

Columns:

☒ ID

☒ region

☒ population

☒ sex

☒ age

☒ education

☒ income

Import! Cancel

ENCLOSED BY "

One; two; "three; tres; trois" ; four ; five

**ESCAPED BY **

**One; two; "In France, one would say
\"trois\""; four ; five**

Clean the data of a table

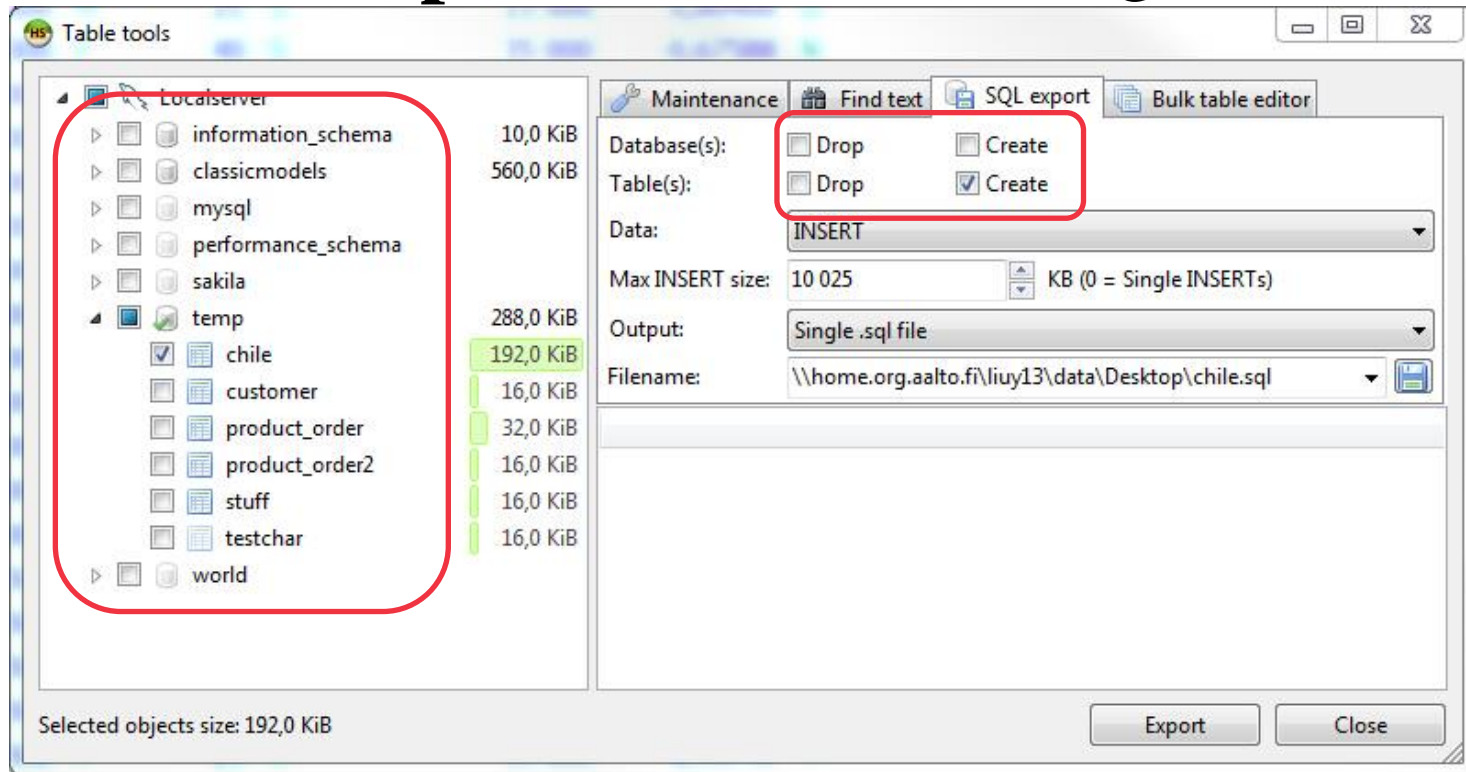
- If the data was not imported correctly, you can drop the problematic data. Please using the following comment to clean the table

```
delete from table_name;
```

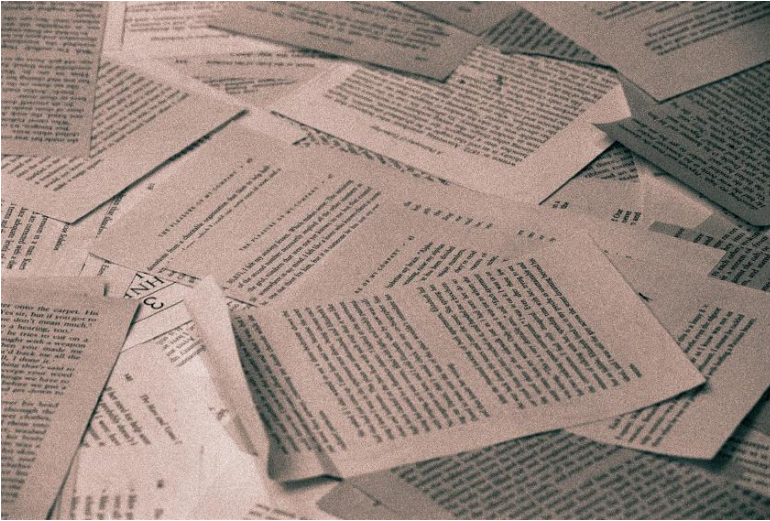
- **Output:** an empty table will be returned

Export data to be a database file

- **Tools → Export database as SQL**



Section 2: Key and Index



A database without index!

“In database systems, an index (IDX) is a data structure defined on columns in a database table to significantly speed up data retrieval operations. An index is a small copy of a database table sorted by key values. Without an index, query languages like SQL may have to scan the entire table from top to bottom to choose relevant rows.”

Index of MySQL is similar to the index of a dictionary, or the address of a person in Finland when you want to find the person.

Index

- Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs.
- If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.

Index of MySQL

Localserver\world\city - HeidiSQL 9.1.0.4904

File Edit Search Tools Help

Database filter Table filter Host: 127.0.0.1 Database: world Table: city Data Query* Query #2* x Query #3* x

Basic Options **Indexes** Foreign keys Partitions CREATE code ALTER code

Localserver

- information_schema
- classicmodels
- mysql
- performance_schema
- sakila
- temp 592,0 KiB
- world 430,3 KiB
 - city 308,9 KiB
 - country 65,9 KiB
 - countrylanguage 55,5 KiB

Add **Remove** **Clear**

Name

- PRIMARY KEY
- ID

Up Down

Columns: **Add** **Remove** Up Down

#	Name	Datatype	Length/Set	Unsign...	Allow NULL	Zerofill	Default
1	ID	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREMENT
2	Name	CHAR	35	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
3	CountryCode	CHAR	3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
4	District	CHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5	Population	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0

14.09.2022

Localserver(world/city) - HeidiSQL 9.1.0.4904

File Edit Search Tools Help

Database filter Table filter

Host: 127.0.0.1 Database: world Table: city Data Query# Query#2* Query#3*

Basic Options Indexes Foreign keys Partitions CREATE code ALTER code

Localserver

- information_schema
- classicmodels
- mysql
- performance_schema
- sakila
- temp 592,0 KiB
- world 430,3 KiB
 - city 308,9 KiB
 - country 65,9 KiB
 - countrylanguage 55,5 KiB

Columns: Add Remove Up Down

#	Name	Datatype	Length/Set	Unsign...	Allow NULL	Zerofill	Default
1	ID	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREMENT
2	NAME	CHAR	35	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
3	POPULATION	INT	3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
4	AREA	INT	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
5	POPULATION	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0

Context menu for column #1 ID:

- Copy Ctrl+C
- Copy selected columns
- Paste columns
- Add column Ctrl+Ins
- Remove column Ctrl+Del
- Move up Ctrl+U
- Move down Ctrl+D
- Create new index
 - PRIMARY KEY
 - UNIQUE
 - FULLTEXT
 - SPATIAL
- Add to index

852 select * from city order by ID asc;

853 /* Affected rows: 0 Found rows: 4 079 Warnings: 0 Duration for 1 query: 0,000 sec. */

Columns: Add Remove Up Down

#	Name	Datatype	Length/Set	Unsign
1	ID	INT	11	<input type="checkbox"/>
2	NAME	CHAR	35	<input type="checkbox"/>
3	POPULATION	INT	3	<input type="checkbox"/>
4	AREA	INT	20	<input type="checkbox"/>
5	POPULATION	INT	11	<input type="checkbox"/>

Context menu for column #1 ID:

- Copy Ctrl+C
- Copy selected columns
- Paste columns
- Add column Ctrl+Ins
- Remove column Ctrl+Del
- Move up Ctrl+U
- Move down Ctrl+D
- Create new index
 - PRIMARY KEY
 - UNIQUE
 - FULLTEXT
 - SPATIAL
- Add to index

Advantages vs. Disadvantages of index

Advantages

- Speed up relevant queries like select.

Disadvantages

- more disk space,
- degrade insert/updates/delete speed
- Building index itself takes time

Operation on big data



- Performing a command in a table with 384,243 rows of data

Commands	Query duration
<code>select * from MyTable where id = 7278409</code>	Duration for 1 query: 0,000 sec.
<code>select * from MyTable where title like '%This is a new product%'</code>	Duration for 1 query: 10,765 sec.
<code>select * from MyTable where title like '%This is a new product%' and id = 7278409</code>	Duration for 1 query: 0,000 sec.
<code>select * from MyTable where title like '%This is a new product%' and via_mobile = 'TRUE'</code>	Duration for 1 query: 2,918 sec.

Columns 'id' and 'via_mobile' are indexed, but column 'title' is not indexed₃
"Via_mobile" only has two different values: TRUE or FALSE

Building index in MySQL (Attention!)

- **To operate on a big table (e.g. 10 million rows), you must build an index before your perform basic command like ‘select’.**
- **Building index also takes time.**

Building index also takes time!



- “I have a table with 1.4 billion records. The table structure is as follows:

```
CREATE TABLE text_page ( text VARCHAR(255) ,  
                           page_id INT UNSIGNED )  
ENGINE=MYISAM DEFAULT CHARSET=ascii
```

The requirement is to create an index over the column **text**.

- The table size is about 34G.
- I have tried to create the index by the following statement:

```
ALTER TABLE text_page ADD KEY ix_text (text)
```

- After 10 hours' waiting I finally give up this approach.
- Is there any workable solution on this problem?”

Tips



- **Indexed the columns referenced in the WHERE clause and columns used in JOIN clauses.**
- **Indexing columns in abundance will result in some disadvantages. However, many times these disadvantages are negligible.**
- **Use the NOT NULL attribute for those columns in which you consider the indexing, so that NULL values will never be stored.**

**Where and join commands
will be introduced in the
future sessions.**

Key and Index

- **KEY is normally a synonym for INDEX.**
- **Columns defined as primary keys or unique keys are automatically indexed in MySQL.**

Primary key

- **As a principle, there should be no duplicated rows co-existing in a table.**
- **A PRIMARY KEY is a unique index where all key columns must be defined as NOT NULL.**
- **A table can have only one PRIMARY KEY.**

Unique key

- **A unique key creates a constraint such that all values in the index must be distinct.**
- **A unique key permits multiple NULL values for columns that can contain NULL.**
- **A table can have multiple unique keys**

Duplicating a table

- Copy the structure and indexes, but not the data:
 - `create table new_table like old_table;`
- Copy the structure, **indexes** and the data
 - `Create table new_table like old_table;`
 - `Insert new_table select * from old_table;`
- Copy the data and the structure, but not the **indexes**:
 - `create table new_table as select * from old_table;`

Purpose of foreign key?

- To delete the information of the students who are graduated from the university from the database, you need to, e.g. :
 - **Delete** students' IDs from the table of **university student list**
 - **Delete** students' IDs from the table of **department student list**
 - **Delete** students' IDs from the table of **course management**
 - **Delete** students' IDs from the table of **library**
 - **Delete** students' IDs from the table of **health care**
 - **Etc.**

Foreign key

- A foreign key is a column in a table (**table A**).
 - But this column is a primary key in the other table (**table B**).
 - Any data in a foreign key column of **table A** must have corresponding data in the other table (**table B**).
- Note: foreign-key column need not be unique in **table A**.

Example

Primary key

Table customer

🔑 Person_Id	name	address	telephone
1	David	Helsinki street 50	3 580 000
2	Jorge	Espoo street 50	3 580 001
3	John	Turku street 50	3 580 002
4	Richard	Oulu street 50	3 580 003
5	Jason	Helsinki street 25	3 580 005

Table product_order

Product	Order_ID	🔑 Person_ID	Price
ABC washing machine	2	1	550
ABC TV	2	1	600
ABC vacuum cleaner	2	2	100

Foreign key

Create foreign key (1)



Product	Order_ID	Person_ID	Price
ABC washing machine	2	1	550
ABC TV	2	1	600
ABC vacuum cleaner	2	2	100

```
CREATE TABLE product_order
```

```
(
```

```
Product char(50) NOT NULL,
```

```
Order_ID int NOT NULL,
```

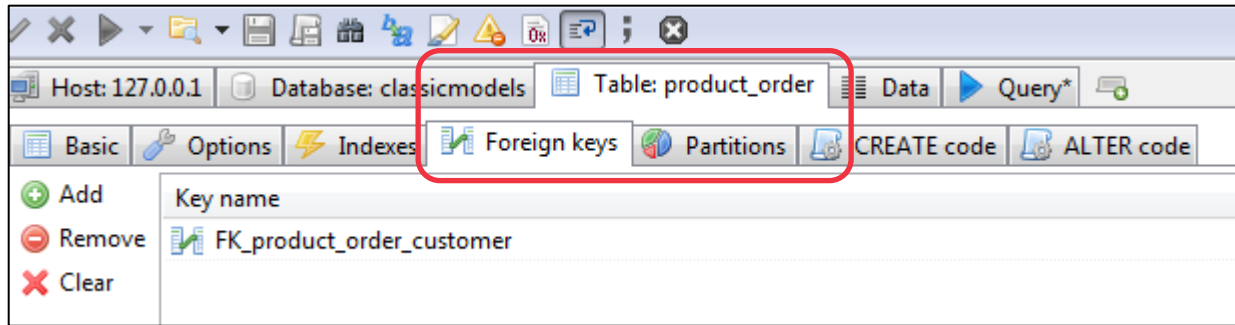
```
Person_ID int NOT NULL,
```

```
Price int NOT NULL,
```

```
Constraint `key_1` foreign key (Person_ID) references  
customer(Person_ID)
```

```
);
```

Create foreign key via HeidiSQL



Columns	Reference table	Foreign colu...	On UPDATE	On DELETE
Person_ID	customer	Person_Id	SET NULL	NO ACTION

Using FOREIGN KEY Constraints

- Delete and update record using foreign key

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
REFERENCES tbl_name (index_col_name, ...)
[ON DELETE reference_option]
[ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

default

<https://www.sitepoint.com/mysql-foreign-keys-quicker-database-development/>
<https://dev.mysql.com/doc/refman/5.6/en/create-table-foreign-keys.html>

FOREIGN KEY Constraints

For both update and delete :

if you try to update / delete the parent row :

- **Restrict** : Nothing gonna be delete if there is a child row. Rejects the delete or update operation for the parent table [Equivalent to **No Action**].
- **Cascade** : the child row will be **delete / update** too
- **Set Null** : the child column will be set to null if you **delete** the parent [make sure that you have **NOT** declared the columns in the child table as **NOT NULL**.]



Create foreign key (2)

If you want to add a foreign key after the related tables have been built:

```
ALTER TABLE product_order ADD `key_1` FOREIGN KEY  
(Person_ID) REFERENCES customer(Person_ID)
```

Host: 127.0.0.1 Database: class models Table: orderdetails Data chile.sql* Query #2* x Query #3* x

Basic Options Indexes **Foreign keys** Partitions CREATE code ALTER code

Key name

orderdetails_ibfk_1

orderdetails_ibfk_2

Columns	Reference table	Foreign col...	On UPDATE	On DELETE
orderNum...	orders	orderNum...	RESTRICT	RESTRICT
productCo...	products	productCo...	RESTRICT	RESTRICT

Function of foreign key

- Inserting an order from a customer that does not exist in the table customer?

```
insert into product_order values ('ABC washing  
machine', '02', '11', 550);
```

/ SQL Error (1452): Cannot add or update a child row: a foreign key constraint fails
(`temp`.`product_order`, CONSTRAINT `key_1` FOREIGN KEY (`Person_ID`)
REFERENCES `customer` (`Person_Id`)) */*

Product	Order_ID	Person_ID	Price
ABC washing machine	2	1	550
ABC TV	2	1	600
ABC vacuum cleaner	2	2	100

Referential integrity

- **Foreign key values must exist in another table**
 - If not, those records cannot be joined
- **Can be enforced when data is added**
 - Associate a primary key with each foreign key
- **Helps avoid erroneous data**
 - Only need to ensure data quality for primary keys

Example (again)

Primary key

Table customer

🔑 Person_Id	name	address	telephone
1	David	Helsinki street 50	3 580 000
2	Jorge	Espoo street 50	3 580 001
3	John	Turku street 50	3 580 002
4	Richard	Oulu street 50	3 580 003
5	Jason	Helsinki street 25	3 580 005

Table product_order

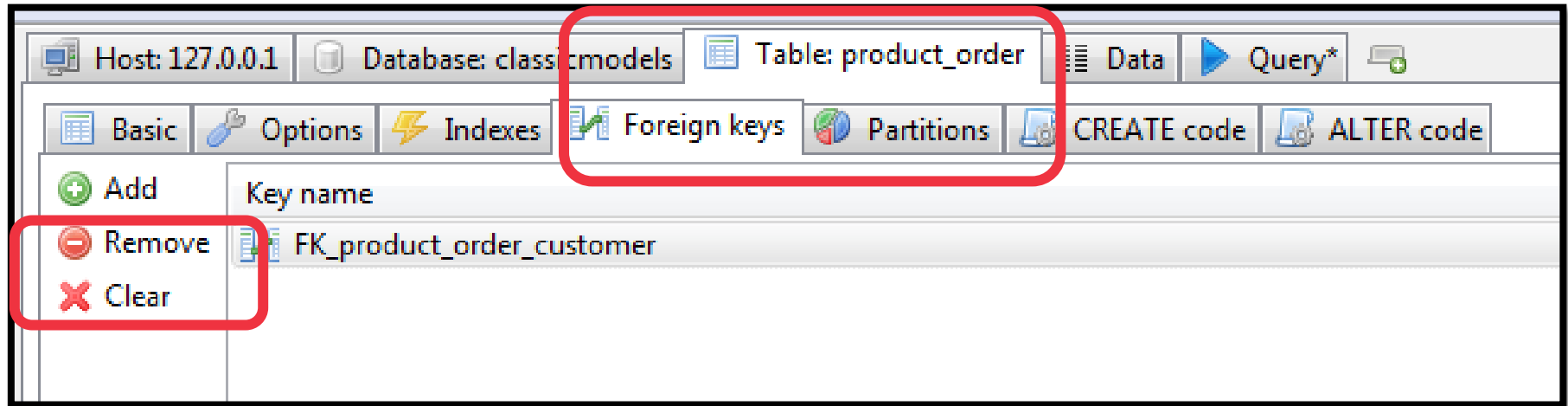
Product	Order_ID	🔑 Person_ID	Price
ABC washing machine	2	1	550
ABC TV	2	1	600
ABC vacuum cleaner	2	2	100

Foreign key

Reflection

How to use foreign key in your research project?

Drop foreign key via HeidiSQL



Drop foreign key via commands



- Foreign key tends to have a different name that is not intuitively available.
- Step 1: Obtain the name of foreign key or **constraint name**
 - **Two substep2**
- Step 2: drop the foreign key or constraint

Step 1:

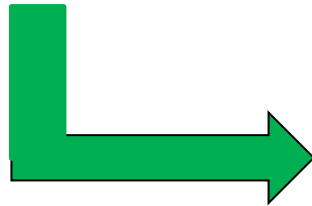
“**show create table**
Table_Name”



Step 2:

Obtain the name of a
foreign key:

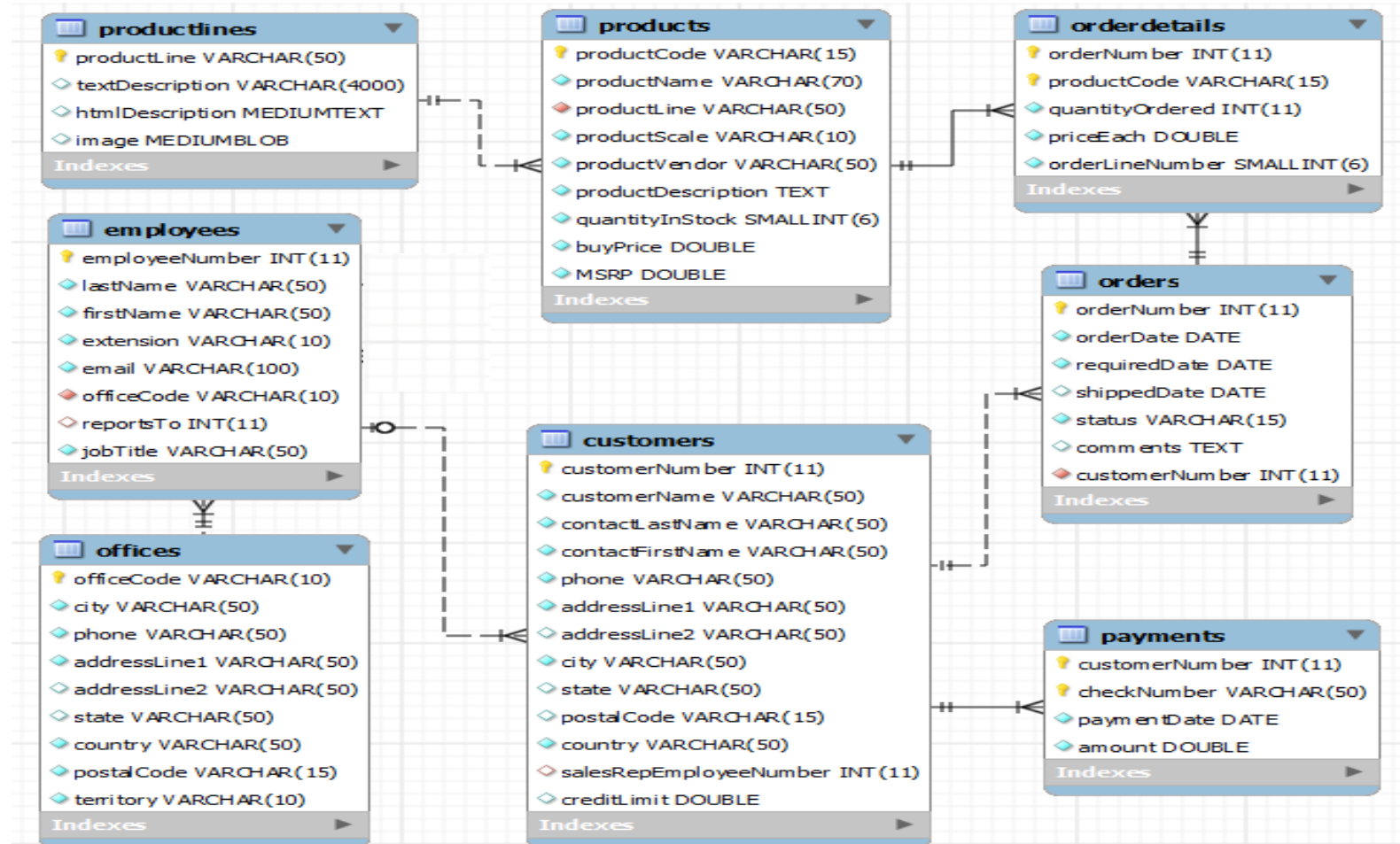
```
CREATE TABLE `orderdetails` (  
  `orderNumber` int(11) NOT NULL,  
  `productCode` varchar(15) NOT NULL,  
  `quantityOrdered` int(11) NOT NULL,  
  `priceEach` double NOT NULL,  
  `orderLineNumber` smallint(6) NOT NULL,  
  PRIMARY KEY (`orderNumber`,`productCode`),  
  KEY `productCode` (`productCode`),  
  CONSTRAINT `orderdetails_ibfk_1` FOREIGN KEY (`orderNumber`) REFERENCES `orders` (`orderNumber`),  
  CONSTRAINT `orderdetails_ibfk_2` FOREIGN KEY (`productCode`) REFERENCES `products` (`productCode`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```



Step 3:

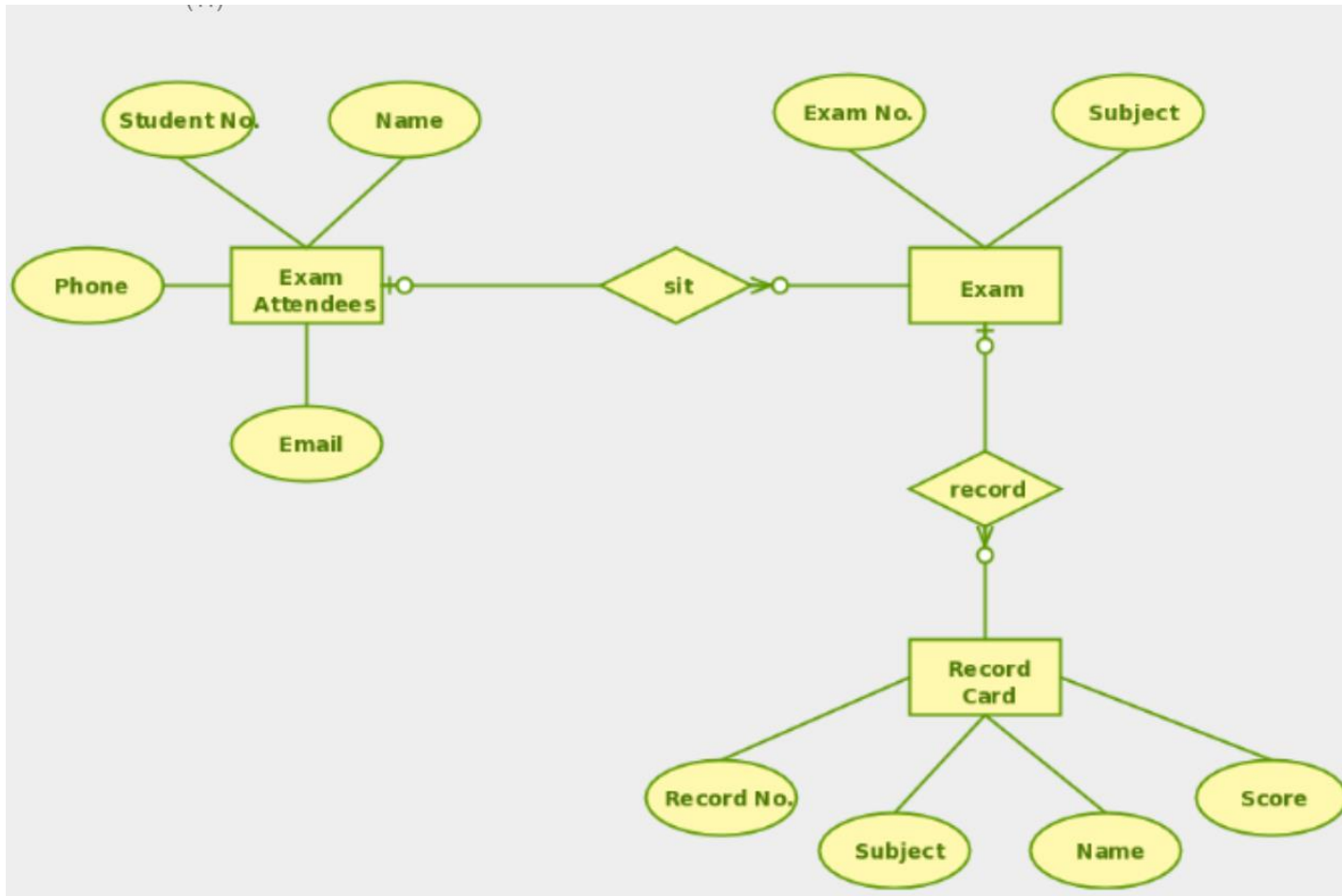
“**alter table Table_Name drop**
foreign key `constraint_name`”

Section 3: Entity-relationship diagram (ERD)



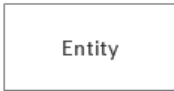
Entity-relationship diagram

- A graphical representation of the structure of a database.
- When a relational database is to be designed, an **entity-relationship diagram** is drawn at an early stage and developed as the requirements of the database and its processing become better understood.



Crow's Foot ERD

There are different notations, the Crow's Foot ERD is a popular one.



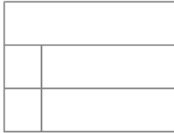
Entity
(with no attributes)



Entity
(with attributes field)



Entity
(attributes field with columns)

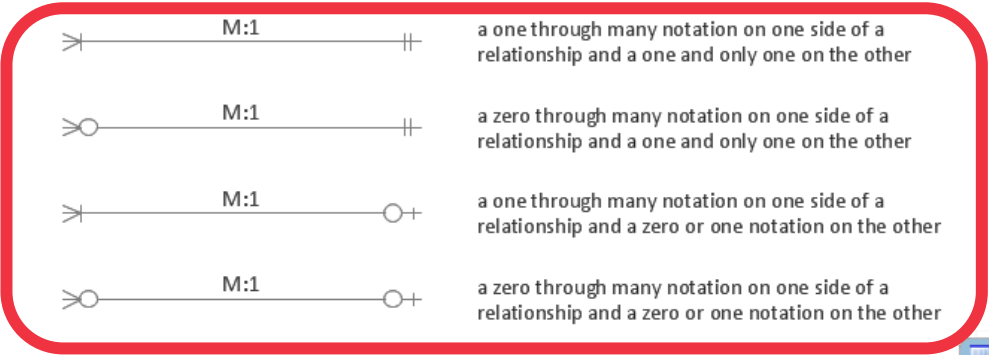


Entity
(attributes field with columns and variable number of rows)

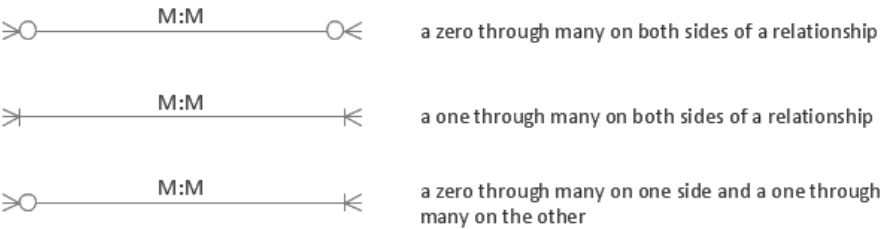
Relationships
(Cardinality and Modality)



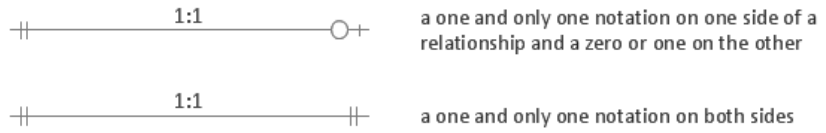
Many - to - One



Many-to-Many



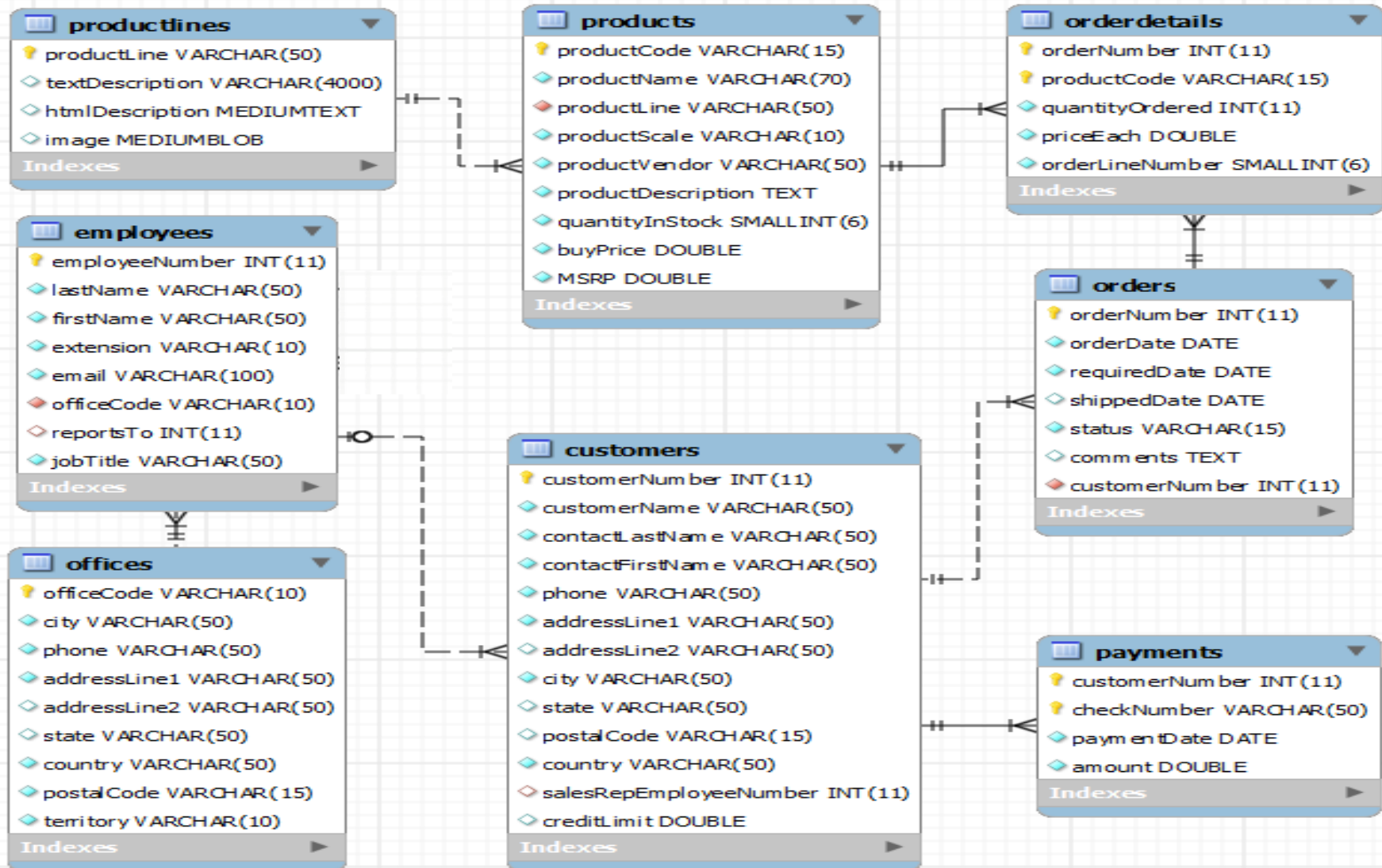
Many-to-Many



customers

- customerNumber INT (11)
- customerName VARCHAR(50)
- contactLastName VARCHAR(50)
- contactFirstName VARCHAR(50)
- phone VARCHAR(50)
- addressLine1 VARCHAR(50)
- addressLine2 VARCHAR(50)
- city VARCHAR(50)
- state VARCHAR(50)
- postalCode VARCHAR(15)
- country VARCHAR(50)
- salesRepEmployeeNumber INT(11)
- creditLimit DOUBLE

Indexes



Section 4: Select

- **Select** command is used to retrieve data from a table
- Template of a “select” query:

Select attributes

from table or **view**

[**Where** conditions]

[**Group by** attributes [**Having** condition]]

[**Order by** attributes [**asc** | **desc**]]

[**Limit**]

Select for calculation

- In command window:

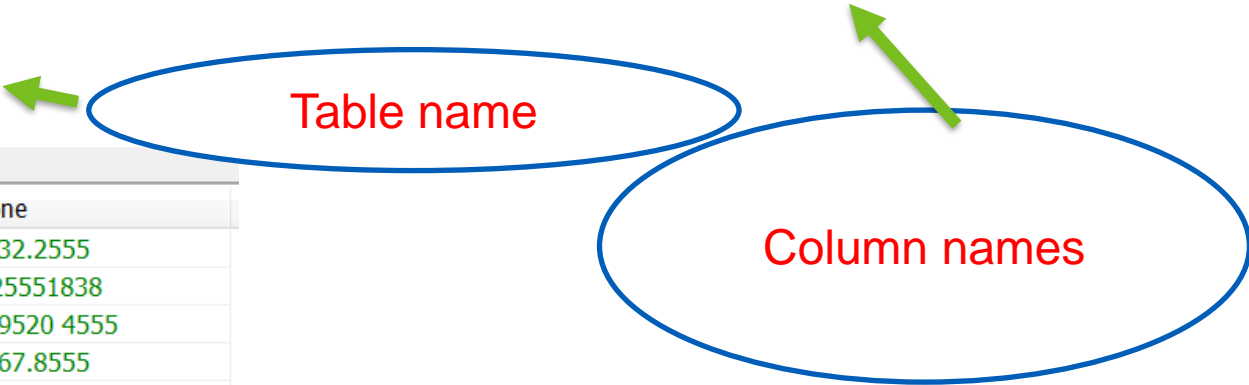
The diagram illustrates the correct syntax for performing a calculation in a SQL command window. It shows three lines of input:

- Line 1: `5+5;` This is labeled "This does not work!" with a green arrow pointing to the semicolon.
- Line 2: (blank)
- Line 3: `select 5+5;` This is labeled "This works!" with a green arrow pointing to the semicolon.

Select: Retrieving certain columns and all rows

- If you have a data table with 20+ variables and half a million rows, but you just want to use a few variables.

```
select contactLastName, contactFirstName, phone  
from customers;
```



contactLastName	contactFirstName	phone
Schmitt	Carine	40.32.2555
King	Jean	7025551838
Ferguson	Peter	03 9520 4555
Labrune	Janine	40.67.8555
Bergulfsen	Jonas	07-98 9555
Nelson	Susan	4155551450
Piestrzeniewicz	Zbyszek	(26) 642-7555
Keitel	Roland	+49 69 66 90 2555

If a reserved word is used as column name

- For instance, a column is named as **select**.

```
SELECT `select` from table_name
```

Please note that it is ` (back quote). It is not '

Where you can find



Select all columns via *

- An asterisk (*) indicates an inclusion of all the columns of a table

```
select * from customers;
```

Select...Where...(1)

- A customer of your company is found to not pay the bill by the deadline. Your boss know the ID of the customer (**customerNumber**) is **103** and ask you to provide the contact details of the customer.

Your task: quickly detecting the contact details of the customer whose cusomterNumber is 103.

Solution

```
Select contactLastName, contactFirstName  
from customers  
where customerNumber = 103;
```


Select...Where...(2)

- An important customer contacted your company asking for some information on his deal with the company, but he lost the name card of sales representative who assisted him with the deal. He remember the first name of the person is **Leslie**. Please find the contact information of Leslie from the database.

“ = ” is MySQL equal operator

Remember to use ' to specify a string

**Select lastName, firstName, email
from employees
where firstName = 'Leslie'**

Is it possible to compare strings with 'greater than' and 'less than' ?

select **lastName**, **firstName**, **email**
from **employees**
where **firstName** > 'L'

lastName	firstName	email ▲
Bondur	Loui	lbondur@classicmodelcars.com
Bott	Larry	lbott@classicmodelcars.com
Jennings	Leslie	ljennings@classicmodelcars.com
Thompson	Leslie	lthompson@classicmodelcars.com
Gerard	Martin	mgerard@classicmodelcars.com
Nishi	Mami	mnishi@classicmodelcars.com
Patterson	Mary	mpatterso@classicmodelcars.com
Castillo	Pamela	pcastillo@classicmodelcars.com
Marsh	Peter	pmarsh@classicmodelcars.com
Patterson	Steve	spatterson@classicmodelcars.com
King	Tom	tking@classicmodelcars.com
Patterson	William	wpatterson@classicmodelcars.com
Kato	Yoshimi	ykato@classicmodelcars.com

Comparison operator

Comparison operator	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
!=	Not equal to

Don't get confused with other programming languages!

Comparison operator: Date

- **Your company did a one-day marketing campaign on June 21, 2004.**
- **Your boss wants to know the contact details of the customers who were motivated to make an order on that day.**

Comparison operator: Date

Comparison operator can also be applied to date type column, e.g.:

**Select customerNumber from payments where
paymentDate = '2004-06-21'**

Compound conditions: **and** / **or** / **not**

- **And:** all the simple condition must be true in order for the compound condition to be true.
- **Or:** the compound condition will be true whenever any of the simple conditions is true.
- **NOT:** the compound condition will be true if simple condition is false.

Question (1)

- The marketing group of your company found that customers who are living in Paris with a credit limit over 50,000 are most profitable. They ask you to provide contact details of those customers.


```
SELECT  contactLastName,contactFirstName,  
phone, city, addressLine1  
from customers where city = Paris and creditLimit >  
50000
```



Single quotes are
needed.

Tips: quick way of counting rows

customers (5×85)				
contactLastName	contactFirstName	phone	city	addressLine1
King	Jean	7025551838	Las Vegas	8489 Strong St.
Ferguson	Peter	03 9520 4555	Melbourne	636 St Kilda Road
Labrune	Janine	40.67.8555	Nantes	67, rue des Cinquante Otages
Bergulfsen	Jonas	07-98 9555	Stavern	Erling Skakkes gate 78
Nelson	Susan	4155551450	San Rafael	5677 Strong St.
Keitel	Roland	+49 69 66 90 2555	Frankfurt	Lyonerstr. 34

How many customers have a credit limit over 5000? `SELECT * from customers creditLimit > 5000`

- This approach can also be used for big tables.**

Question (2)

- The marketing group also found that customers living in **city Madrid** with a **creditLimit** over **10,000** are very profitable, in addition to those living in **city Paris** with a **creditLimit** over **5000**.
- Can you retrieve their contact details via one query?

- **Make a proper use of ()**

**Select contactLastName, contactFirstName, phone, city
from customers
where (city = 'Paris' and CreditLimit > 5000)
or (city = 'Madrid' and CreditLimit > 10000)**

Example for NOT

Select contactLastName, contactFirstName, phone, city
from customers
where NOT
(city = 'Paris' and CreditLimit > 5000)
or (city = 'Madrid' and CreditLimit > 10000)

- What does the above command mean?

What does the above query mean?

Select contactLastName, contactFirstName, phone, city
from customers
where NOT
(city = 'Paris' **and** CreditLimit > 5000)
or (city = 'Madrid' **and** CreditLimit > 10000)

Question: Is the above command equal to:

Select contactLastName, contactFirstName, phone, city
from customers
where NOT (city = 'Paris' **and** CreditLimit > 5000)

<http://premo.aalto.fi/drm>

Between...and...

- In the table customers, please retrieve the records of customers with **creditLimit** that are **i)** greater than or equal to 60,000 and **ii)** less than or equal to 70,000

Alternative 1:

```
Select * from customers  
where creditLimit >= 60000 and creditLimit <= 70000
```

Alternative 2:

```
Select * from customers  
where creditLimit between 60000 and 70000
```

Challenge!



- Please retrieve the records from table “**orders**” in which **status** is not “**Shipped**” and the **orderDate** is between '2005-05-09' and '2005-05-31' and **requiredDate** is between '2005-06-01' and '2005-06-10' by customers whose **customersNumber** are 124 or 119.

🔑 orderNumber	orderDate	requiredDate	shippedDate	▼ status	comments	📍 customerNumber
10 240	2004-04-13	2004-04-20	2004-04-20	Shipped	(NULL)	177
10 241	2004-04-13	2004-04-20	2004-04-19	Shipped	(NULL)	209
10 242	2004-04-20	2004-04-28	2004-04-25	Shipped	Customer is interested in buying more...	456
10 243	2004-04-26	2004-05-03	2004-04-28	Shipped	(NULL)	495
10 244	2004-04-29	2004-05-09	2004-05-04	Shipped	(NULL)	141


Table **orders**

Answer

```
SELECT *  
FROM orders  
WHERE status != 'Shipped' AND  
orderDate BETWEEN '2005-05-09' AND '2005-05-31' AND  
requiredDate BETWEEN '2005-06-01' AND '2005-06-10' AND  
(customerNumber = 124 or customerNumber = 119)
```

Reflect and Question

```
SELECT *  
FROM orders  
WHERE `status` != 'Shipped' AND  
orderDate BETWEEN '2005-05-09' AND '2005-05-31' AND  
requiredDate BETWEEN '2005-06-01' AND '2005-06-10' AND  
(customerNumber = 124 or 119)
```



What will happen if
“customerNumber = ” is removed
from the “customerNumber = 119 ”
condition

Select limit

- ***Limit*** is used to limit your MySQL query results to those that fall within a specified range.
- **select * from products Limit 0,10;**
Retrieve first ten rows or using [Limit 10]
- **select * from products Limit 5,10;**
Retrieve rows 6-15

Select limit

Limit function:

- Enable a quick check of the validity of the result.
- Save lots of time when a huge number of records will be returned [**a big-data issue**].

Using computed columns (1)

- Your boss asks you to provide information on the **values** of different products in stock.
- **Value = quantityInStock * buyPrice**

🔑 productCode	productName	quantityInStock	buyPrice	▼ MSRP
S10_1949	1952 Alpine Renault 1300	7 305	98,58	214,3
S12_1108	2001 Ferrari Enzo	3 619	95,59	207,8
S12_1099	1968 Ford Mustang	68	95,34	194,57
S10_4698	2003 Harley-Davidson Eagle Drag Bike	5 582	91,02	193,66
S12_3891	1969 Ford Falcon	1 049	83,05	173,02

Using computed columns (2)

Select **productCode**, **quantityInStock*buyPrice**
from **products**

select 5/2 → 2,5000

select 5%2 → 1

products (2×110)	
productCode	quantityInStock*buyPrice
S10_1678	387,209.73
S10_1949	720,126.9
S10_2016	457,058.75
S10_4698	508,073.64
S10_4757	278,631.36
S10_4962	702,325.22

+

-

*

/

%

Using **As** for Aliases

- **SELECT** column_name **AS** alias_name
FROM table_name;

Select productCode,
quantityInStock*buyPrice **as** productValue
from products

products (2×110)	
productCode	productValue
S10_1678	387,209.73
S10_1949	720,126.9
S10_2016	457,058.75
S10_4698	508,073.64

Make the result more readable

- **Select** productCode,
round(quantityInStock*buyPrice, 1) **as** productValue
from products

products (2×110)	
productCode	productValue
S10_1678	387,209.7
S10_1949	720,126.9
S10_2016	457,058.7
S10_4698	508,073.6
S10_4757	278,631.4
S10_4962	702,325.2
S12_1099	6,483.1

Save results as a new table

Duplicate tables

- `create table new_table as select * from old_table;`
- `create table temp as (
 select productCode,
 round(quantityInStock*buyPrice , 2)
 as productValue
from products)`

Select: sorting rows

- Your boss is asking you to provide information on the top 3 most valuable products in stock.

🔑 productCode	productName	quantityInStock	buyPrice	▼ MSRP
S10_1949	1952 Alpine Renault 1300	7 305	98,58	214,3
S12_1108	2001 Ferrari Enzo	3 619	95,59	207,8
S12_1099	1968 Ford Mustang	68	95,34	194,57
S10_4698	2003 Harley-Davidson Eagle Drag Bike	5 582	91,02	193,66
S12_3891	1969 Ford Falcon	1 049	83,05	173,02

Order by... asc/desc

- Problem of unsorted results
- Order by column asc/desc
 - Asc : ascending (default option)
 - Desc: descending

products (2×110)	
productCode	productValue
S10_1678	387 209,73
S10_1949	720 126,90
S10_2016	457 058,74
S10_4698	508 073,63
S10_4757	278 631,36
S10_4962	702 325,22
S12_1099	6 483,12
S12_1108	345 940,21
S12_1666	123 004,10
S12_2823	662 501,18
S12_3148	615 600,84
S12_3380	685 684,68
S12_3891	87 119,45
S12_3990	180 762,96

Example

- **select** productCode,
quantityInStock*buyPrice **as** productValue
from products

order by productValue **desc**

limit 3

More complex sort

- Sort several columns at the same time

Order by column 1, column 2...

Primary sort key

Secondary sort key

- If you want to change direction of sorting among columns

Order by column 1 desc,
column 2 asc

🔑 orderNumber	🔑 productCode	quantityOrdered	priceEach
10 100	S18_1749	30	136
10 100	S18_2248	50	55,09
10 100	S18_4409	22	75,46
10 100	S24_3969	49	35,29
10 101	S18_2325	25	108,06
10 101	S18_2795	26	167,06
10 101	S24_1937	45	32,53
10 101	S24_2022	46	44,35

Question

temp.payments: 273 rows total (approximately)

👉 customerNumber	👉 checkNumber	paymentDate	▲ amount
398	JPMR4544	2005-05-18	615,45
381	MS154481	2003-08-22	1 128,2
121	FD317790	2003-10-28	1 491,38
381	CC475233	2003-04-19	1 627,56
103	OM314933	2004-12-18	1 676,14
456	MO743231	2004-04-30	1 679,92
350	OB648482	2005-01-29	1 834,56
172	AD832091	2004-09-09	1 960,8
161	BR352384	2004-11-14	2 434,25
148	DD635282	2004-08-11	2 611,84
323	HG738664	2003-07-05	2 880
216	BG407567	2003-05-09	3 101,4
219	BN17870	2005-03-02	3 452,75
484	GK294076	2004-10-26	3 474,66
148	ME497970	2005-03-27	3 516,04
205	GL756480	2003-12-04	3 879,96
204	IS150005	2004-09-24	4 424,4
219	BR941480	2003-10-18	4 465,85
209	ED520529	2004-06-21	4 632,31
145	CN328545	2004-07-03	4 710,73
181	GQ132144	2003-01-30	5 494,78

- Your boss wants to identify ‘big’ payments to the company **in 2004**
- **Amounts** of ‘big’ payments should be **over 10000**.
- Please exhibit results in both an ascending order for **customerNumber** and a descending order for **amount**

Table ‘payments’

Answer

```
Select * from payments
      where amount > 10000 and
            paymentDate between '2004-01-01'
            and '2004-12-31'
      order by customerNumber asc,
            amount desc
```

Instructions on Hands-on session 3

- “chile” data and “classicmodels” database will be used for the hands-on training 3. The tables belonging to “classicmodels” database should have already been imported at the first hands-on session via importing the “classicmodels (using PCs in the lab or at home).sql” file.
- The Chile data has 2400 rows and 8 columns. This data was derived from a national survey conducted in April and May of 1988 by FLACSO/Chile. Missing data are removed.
- Please read description of the dataset (Description of Chile election 1988.docx), downloadable from MyCourse [Data and database files folder]