The Standard Query Language (MySQL Implementation)

General Aspects

SQL (or “sequel”) means Standard Query Language.
It is designed for creating queries to a **Relational Database**.
A Database is a collection of relations instances called **Tables**.
It is a declarative (not imperative) programming language.
Aspects of SQL are:

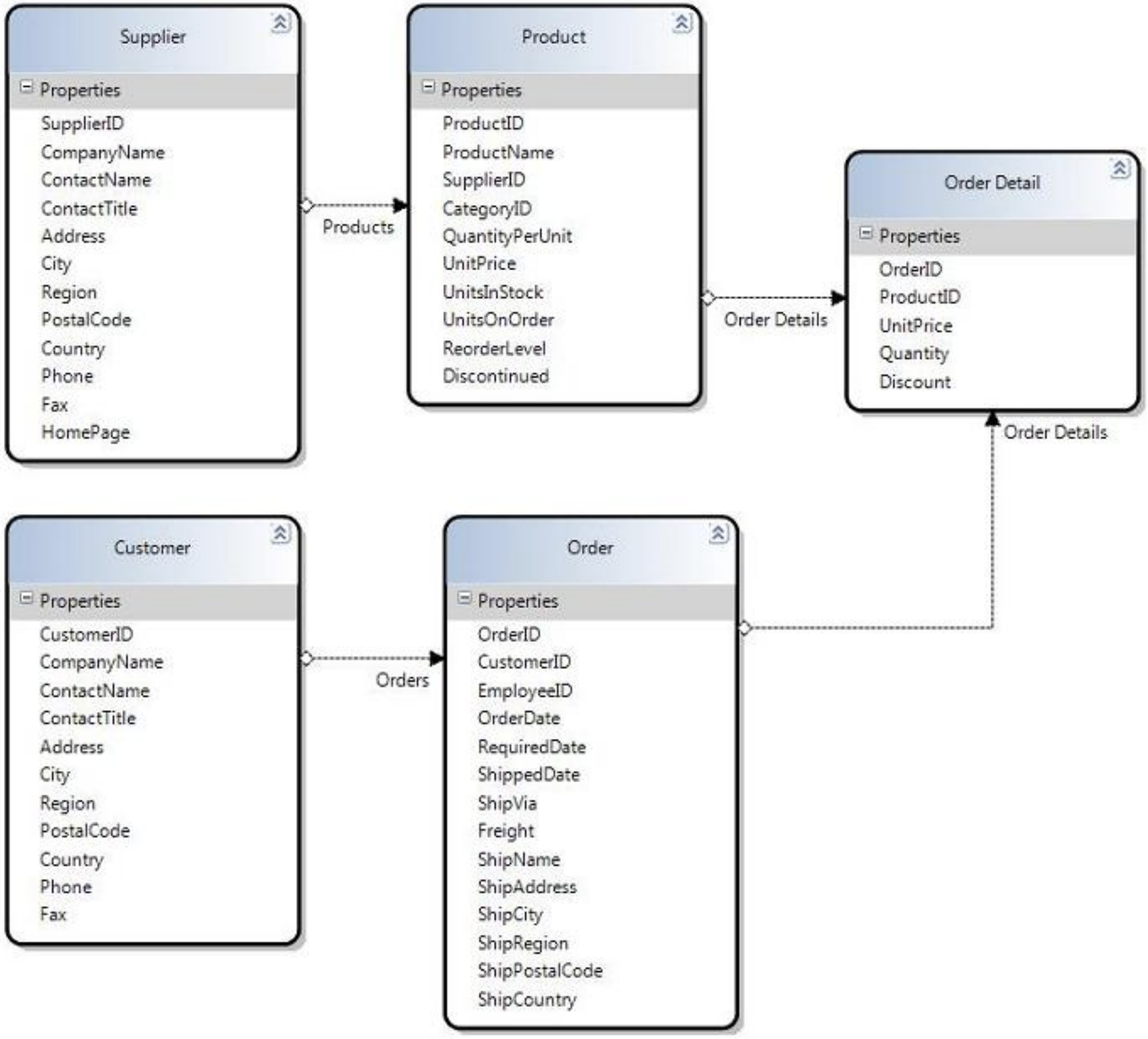
DML (Data Manipulation Language): SQL allows the creation and submission of queries to a database.

DDL (Data Definition Language): SQL provides commands for the creation and modification of tables.

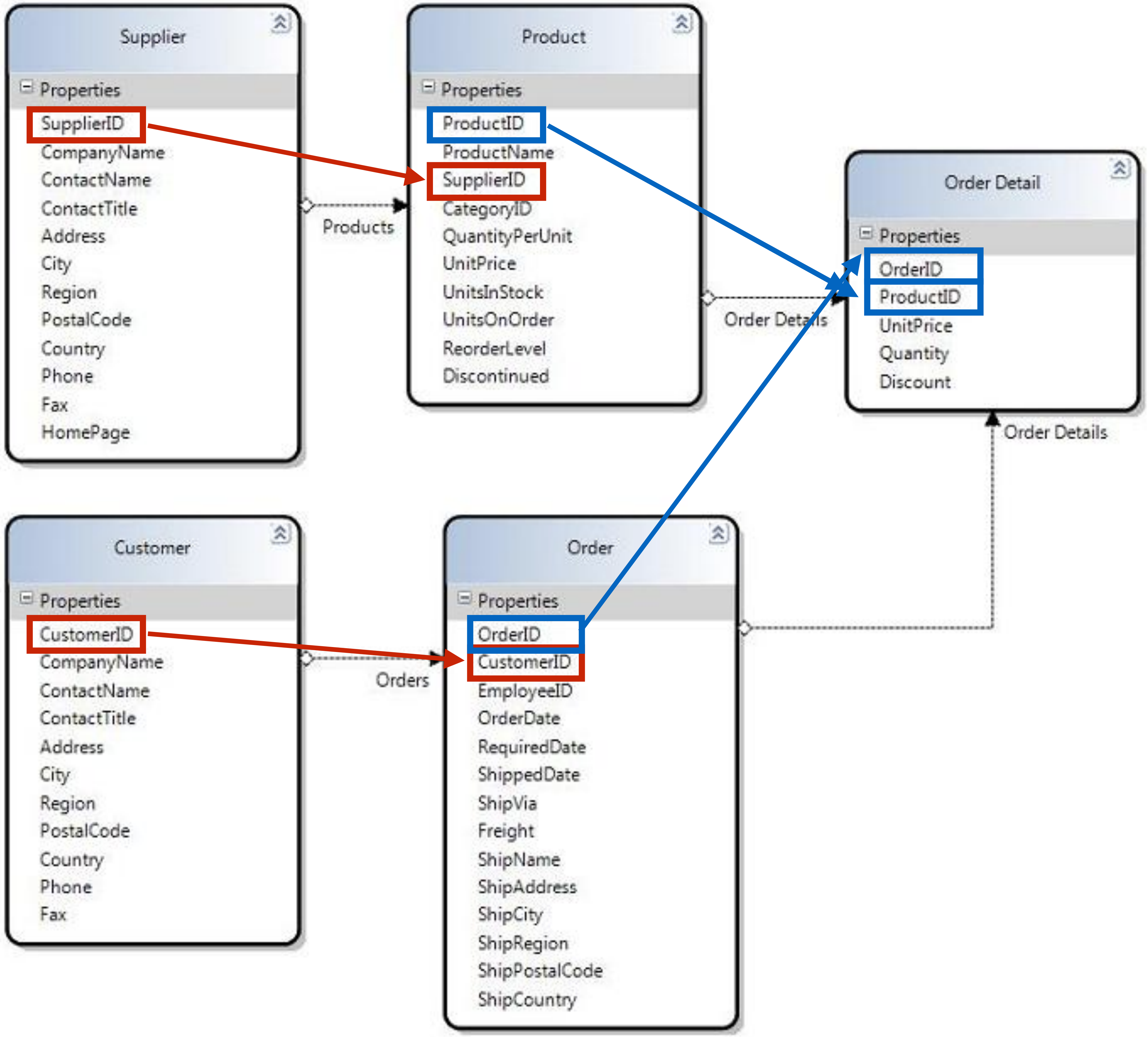
Triggers: It is possible to define automatic procedures in response to operations on the database, for ensuring the integrity of the DB’s structure.

Client-Server Features: SQL supports communication to a remote database, the management of the transactions, their security and concurrency.

Working with more tables



Primary and Foreign Keys



Relational Integrity

What does it happen when we modify/insert/delete records in a table belonging to a multi-table Database?

These operations, carried out on one table, might leave records in other tables unreferenced (“orphaned”).

Such problems break the “relational integrity” of the database.

Relational Integrity must be enforced either by functions or automatic procedures (“triggers”).

Relational integrity is a property of databases which requires every value of one attribute (column) of a table to exist as a value of another attribute in a different (or the same) table.

“CRUD”, or the functions of Persistence Storage

CREATE

RETRIEVE

UPDATE

DELETE

- Origins date back to about 1994.
- It is an **open-source** Database Management System.
- The source code is available under the GNU licence agreement.
- Supports the client-server architecture.
- Very popular for internet applications.
- Now owned by Oracle.
- We are now (2015-2016) to version 5.
- No native graphical interface: we will use phpmyAdmin (another free software).

The structure of a simple query

SELECT [DISTINCT] <fields> $\longrightarrow$ Projection π
FROM <tables> $\longrightarrow$ Product $\times$
WHERE <conditions> $\longrightarrow$ Selection σ

- Upper case is not mandatory (but recommended).
- SELECT and FROM clauses are the only mandatory commands in a query.
- DISTINCT is optional and avoids duplicated records.
- The wildcard “ * ” is available (e.g. SELECT *).
- The WHERE clause contains a logical expression (AND,OR,NOT,=,<,>)
- SQL implements closely the relational algebra.

Example

Table T

ID	Name	Age	City
1	John Smith	44	Kansas City
2	Steven Miller	32	Portland
3	Simon Chan	21	Vancouver
4	Yuri Kirillov	57	Novosibirsk
5	Antonio Esposito	19	Napoli
6	Pierre Daveaux	76	Marseille
7	Antonia Caballero	33	Madrid
8	Alexa Fredrikson	37	Oslo
9	Jack Post	22	New York
10	Shintaro Ito	25	Osaka

```
SELECT Name,City
FROM T
WHERE Age<23
```

Name	City
Simon Chan	Vancouver
Antonio Esposito	Napoli
Jack Post	New York

Or using a range variable:

```
SELECT V.Name,V.City
FROM T as V
WHERE V.Age <= 23
```

Example

- Open MySQL
- Open the example database:

```
source U:\\lucad\testdatabase.sql
```

Check if the database is there and use it:

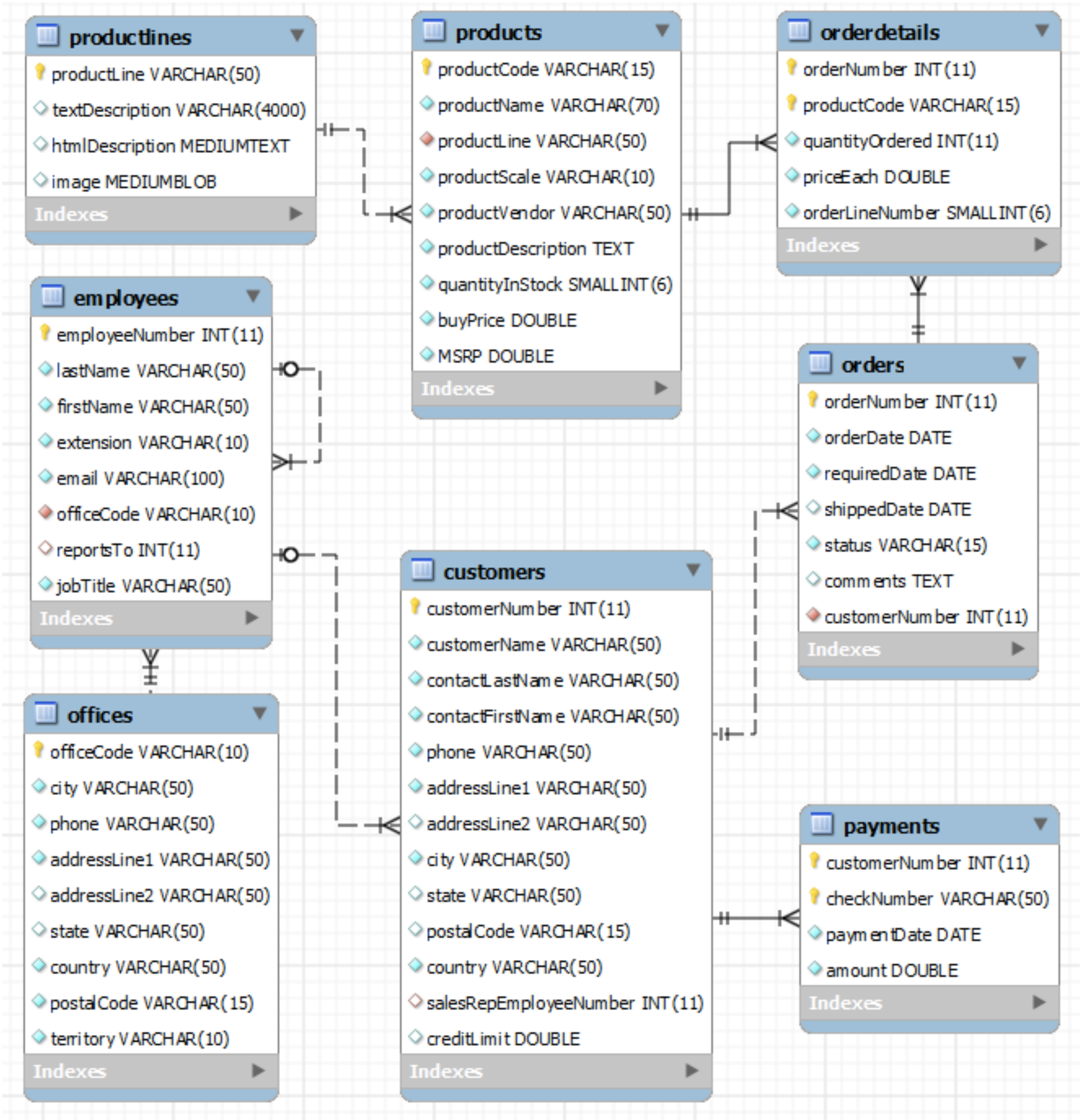
show databases: `use <dbname>;`

Look at the tables in it: `show tables;`

Look at the properties of a table:

```
describe <tablename>;
```

ClassicModels Database

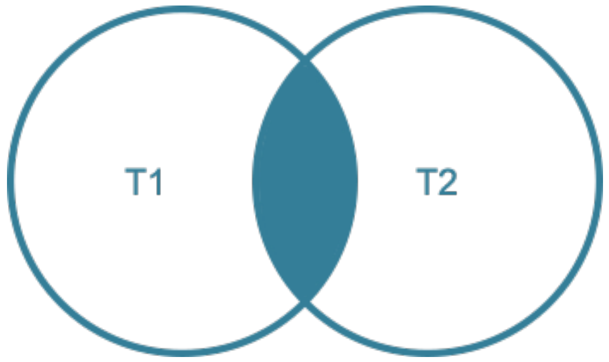


More complex queries

```
SELECT
    column_1, column_2, ...
FROM
    table_1
[INNER | LEFT |RIGHT] JOIN table_2 ON conditions
WHERE
    conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1
LIMIT offset, length;
```

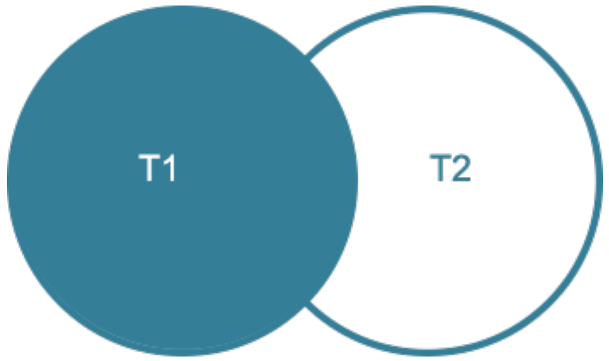
INNER JOIN

For each row in the T1 table, the MySQL INNER JOIN clause compares it with each row of the T2 table to check if both of them satisfy the join condition. When the join condition is matched, it will return the row that combine columns in both T1 and T2 tables.



LEFT JOIN

When we join the T1 table to the T2 table using the LEFT JOIN clause, if a row from the left table T1 matches a row from the right table T2 based on the join condition (T1.c1 = T2.c1), this row is included in the result set. In case the row in the left table does not match the row in the right table, the row in the left table is also selected and combined with a “fake” row from the right table. The fake row contains NULL values for all corresponding columns in the SELECT clause.



RIGHT JOIN

LEFT <-> RIGHT

Set Operations

SQL “set operations” are indeed quite similar to operations usually carried out on general sets. In SQL, set operations are applied to tables.

UNION:

Combines the results of two SQL queries into a single table of all matching rows. The two queries must result in the same number of columns and compatible data types in order to unite. Any duplicate records are automatically removed unless UNION ALL is used. UNION does not respect ordering and ORDER BY should be included if needed.

INTERSECT

The SQL INTERSECT operator takes the results of two queries and returns only rows that appear in both result sets. For purposes of duplicate removal the INTERSECT operator does not distinguish between NULLs. The INTERSECT operator removes duplicate rows from the final result set. The INTERSECT ALL operator does not remove duplicate rows from the final result set.

MINUS / EXCEPT

The SQL EXCEPT operator takes the distinct rows of one query and returns the rows that do not appear in a second result set. The EXCEPT ALL operator does not remove duplicates. For purposes of row elimination and duplicate removal, the EXCEPT operator does not distinguish between NULLs.

Some DBMSs provide a MINUS operator which is functionally equivalent to the SQL standard EXCEPT DISTINCT operator

UNION in MySQL

SELECT column1, column2
UNION [DISTINCT | ALL]
SELECT column1, column2
UNION [DISTINCT | ALL]

UNION combines the results of the two SELECT clauses in one single result.

<u>Try this in our example DB:</u>

```
SELECT customerNumber id, contactLastname name
FROM customers
UNION
SELECT employeeNumber id,firstName name
FROM employees
```

Set Operations in MySQL

MySQL doesn't support the **INTERSECT** and **MINUS** set operators.
We can rewrite these queries by using the JOIN operator:

```
SELECT x, y FROM table_a
MINUS
SELECT x, y FROM table_b;
```

$\longrightarrow$

```
SELECT a.x, a.y
FROM table_a a LEFT JOIN table_b b
ON a.x = b.x AND a.y = b.y
WHERE b.x IS NULL;
```

```
SELECT x, y FROM table_a
INTERSECT
SELECT x, y FROM table_b;
```

$\longrightarrow$

```
SELECT a.x, a.y
FROM table_a a JOIN table_b b
ON a.x = b.x AND a.y = b.y;
```

DIVIDE in MySQL

This operation is not directly supported in MySQL.
DIVIDE is not a basic relational algebra operator, so it should be possible to construct it.
Recall the algebraic definition:

$$T1/T2 = \pi_x(T1) - \pi_x((\pi_x(T1) \times T2) - T1)$$

This kind of operator answers the questions with an "ALL" into them.
Example:
"Which employees of a certain kind work in **ALL** the projects listed in table T ?"
"Which boys are registered on those courses that are taken by **ALL** the girls?"
"Which girls are registered on **ALL** the courses taken by student nr. 999?"

The logical operator behind division is the implication. We can rephrase one of the above
sentences as:

For which girls is it true that:
IF 999 is registered in a course THEN the girl is registered in the same course?

OR, in a "double negation" logic:

Find all the "girls" rows where is NOT true that the student 999 is NOT registered in the same course.

DIVIDE in MySQL

“Which girls are registered on **ALL** the courses taken by student nr. 999?”

OR:

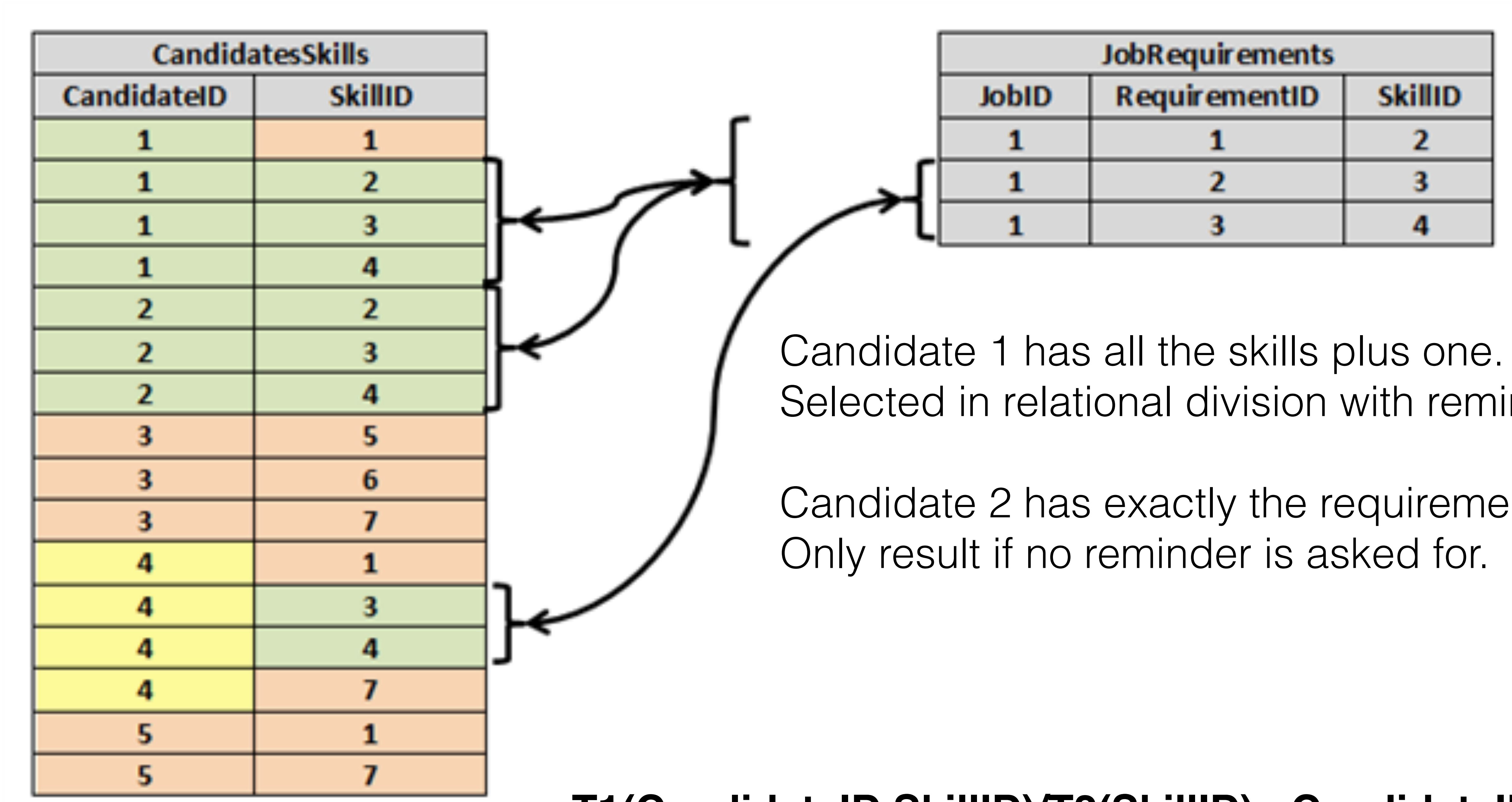
Find all the “girls” rows where is NOT true that the student 999 is NOT registered in the same course.

```
SELECT DISTINCT Name, StNumber  ←————————  Resulting fields
FROM Students AS S
WHERE Sex=GIRL AND
      NOT EXISTS  ←————————  Girls and no student not registered in the same course
         (
            SELECT Course
            FROM CourseRegistrations
            WHERE StNumber = 999 AND Course NOT IN
                 (
                    SELECT Course
                    FROM CourseRegistrations as C
                    WHERE S.StNumber = C.StNumber
                 )
         )
```

***NOTE**: There are ways for expressing divisions without double negations.*

Example

1) Table T1 with job candidates and their skills.
2) Table T2 with jobs with skills requirements.
3) Task: find candidates which have ALL the skills required for a job.
4) **Solution**: Relational division T1/T2



CandidatesSkills	
CandidateID	SkillID
1	1
1	2
1	3
1	4
2	2
2	3
2	4
3	5
3	6
3	7
4	1
4	3
4	4
4	7
5	1
5	7

JobRequirements		
JobID	RequirementID	SkillID
1	1	2
1	2	3
1	3	4

Candidate 1 has all the skills plus one.
Selected in relational division with reminder.

Candidate 2 has exactly the requirements.
Only result if no reminder is asked for.

T1(CandidateID,SkillID)/T2(SkillID)= CandidateIDs