



Sample Database

Install Sample Database

- Go to
 - <https://www.mysqltutorial.org/mysql-sample-database.aspx>
- Should download an sql file

Install Sample Databases

- Method 1:
 - Open MySQL Workbench
 - Connect to MySQL server
 - File —> Run SQL script
 - Choose the downloaded file

Install Sample Databases

- Method 2
 - Connect to the MySQL server with a terminal

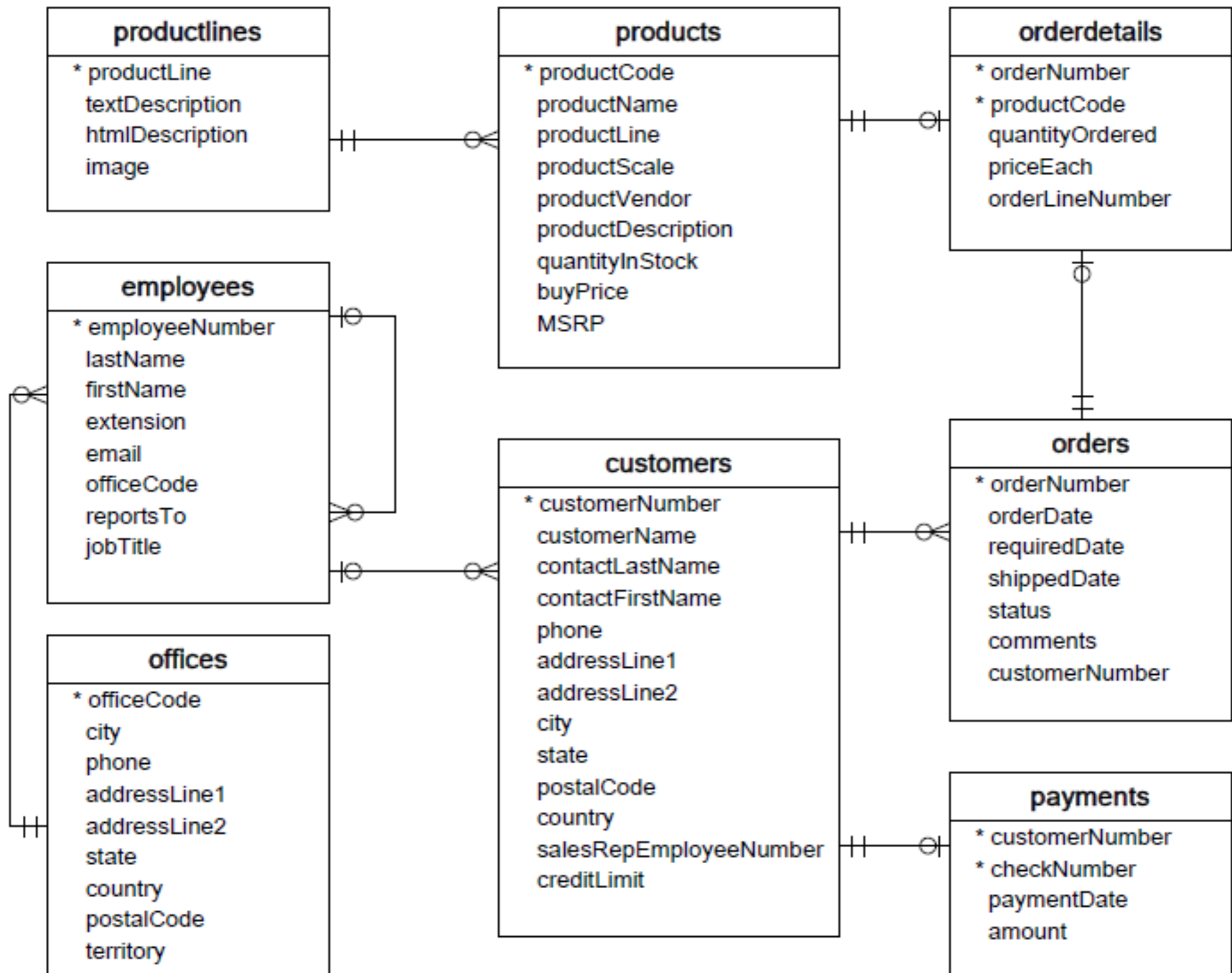
```
mysql -u root -p
```

- Should prompt for your password
- Use the source program

```
mysql> source c:\myPath\to\myfile
```

- Check with

```
mysql> show databases;
```



Install Sample Databases

- You can download the diagram and bring a printed copy to the next class

SQL

Repetition

- Creating Schemas
- Inserting
- Selection
- Constraints

Data Definition Language

SQL DDL

- Create a database with CREATE DATABASE

```
CREATE DATABASE IF NOT EXISTS USNavy;
```

SQL DDL

- Three type of tables in SQL
 - Stored Relations, called tables
 - Views: relations calculated by computation
 - Temporary tables: created during query execution

SQL DDL

- Data Types
 - Character strings of fixed or varying length
 - CHAR(n) - fixed length string of up to n characters
 - VARCHAR(n) - fixed length string of up to n characters
 - Uses and endmarker or string-length for storage efficiency
 - Bit strings
 - BIT(n) strings of length exactly n
 - BIT VARYING(n) - strings of length up to n

SQL DDL

- Data Types:
 - Boolean: BOOLEAN: TRUE, FALSE, UNKNOWN
 - Integers: INT = INTEGER, SHORTINT
 - Floats: FLOAT = REAL, DOUBLE, DECIMAL(n,m)
 - Dates: DATE
 - SQL Standard: '1948-05-14')
 - Times: TIME
 - SQL Standard: 19:20:02.4

SQL DDL

- Data Types:
 - MySQL: ENUM('M', 'F')

SQL DDL

- CREATE TABLE creates a table

```
CREATE TABLE Movies (  
    title          CHAR(100),  
    year          INT,  
    length        INT,  
    genre         CHAR(10),  
    studioName    CHAR(30),  
    producerC#   INT  
);
```

SQL DDL

```
CREATE TABLE MovieStar (  
    name          CHAR(30) ,  
    address       VARCHAR(255) ,  
    gender        CHAR(1) ,  
    birthday      DATE  
);
```


SQL DDL

- Drop Table drops a table

```
DROP TABLE Movies;
```

SQL DDL

- Altering a table with ALTER TABLE
 - with ADD followed by attribute name and data type
 - with DROP followed by attribute name

```
ALTER TABLE MovieStar ADD phone CHAR(16);
```

```
ALTER TABLE MovieStar DROP Birthday;
```

SQL DDL

- Default Values
 - Conventions for unknown data
 - Usually, NULL
 - Can use other values for unknown data

```
CREATE TABLE MovieStar(  
    name            CHAR(30),  
    address         VARCHAR(255),  
    gender          CHAR(1) DEFAULT '?',  
    birthday        DATE DEFAULT '0000-00-00'  
);
```

SQL DDL

- Declaring Keys
 1. Declare one attribute to be a key
 2. Add one additional declaration:
 - Particular set of attributes is a key
- Can use
 1. PRIMARY KEY
 2. UNIQUE

SQL DDL

- UNIQUE for a set S:
 - Two tuples cannot agree on all attributes of S unless one of them is NULL
 - Any attempted update that violates this will be rejected
- PRIMARY KEY for a set S:
 - Attributes in S cannot be NULL

SQL DDL

```
CREATE TABLE MovieStar (  
    name          CHAR(30) PRIMARY KEY,  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthday      DATE  
);
```

SQL DDL

```
CREATE TABLE MovieStar(  
    name          CHAR(30),  
    address       VARCHAR(255),  
    gender        CHAR(1) DEFAULT '?',  
    birthday      DATE DEFAULT '0000-00-00',  
    PRIMARY KEY (name)  
);
```

SQL DDL

```
CREATE TABLE Movies (  
    title            CHAR(100),  
    year            INT,  
    length          INT,  
    genre           CHAR(10),  
    studioName      CHAR(30),  
    producerC#     INT,  
    PRIMARY KEY (title, year)  
);
```


Simple Diagrams

- A schema is represented by a networked diagram
 - Nodes represent tables
 - Name of the table labels the node
 - Interior of the node are the name of the attributes
 - Underline the primary key
 - Optionally, add domain to each attribute

Simple Diagrams

Sales

<u>purchase_number</u> :	int
date_of_purchase :	date
customer_id:	int
item_code:	varchar(10)

Customers

<u>customer_id</u> :	int
first_name :	varchar(255)
last_name :	varchar(255)
email_address :	varchar(10)
number of complaints :	int

Items

<u>item_code</u> :	int
item :	varchar(255)
unit_price:	decimal(10,2)
company_id:	int

Companies

<u>company_id</u> :	int
company_name :	varchar(63)
headquarters_ph_nr:	char(25)

Constraints in MySQL

- Constraints in MySQL have names
 - Often automatically generated
 - Use the SHOW CREATE TABLE query

```
Table, "Create Table"  
customers, "CREATE TABLE `customers` (  
  `customer_id` int NOT NULL AUTO_INCREMENT,  
  `first_name` varchar(255) DEFAULT NULL,  
  `last_name` varchar(255) DEFAULT NULL,  
  `email_address` varchar(255) DEFAULT NULL,  
  `number_of_complaints` int DEFAULT (0),  
  PRIMARY KEY (`customer_id`),  
  UNIQUE KEY `email_address` (`email_address`)  
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_0900_ai_ci"
```

Constraints in MySQL

- Missing values are usually a NULL
 - Can automatically assign INT with AUTO_INCREMENT
 - Used widely to assign artificial primary keys

Constraints in MySQL

- NOT NULL constraint
 - When inserting a tuple with NULL value in the constrained column, error will be thrown

```
CREATE TABLE tasks (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    title VARCHAR(255) NOT NULL,  
    start_date DATE NOT NULL,  
    end_date DATE  
);
```

- Considered good practice to include in all columns where a NULL value is not expected

Constraints in MySQL

- ALTER TABLE allows to introduce new / remove old constraint
- Need to check that the inserted values comply

```
ALTER TABLE tasks  
CHANGE  
    end_date  
    end_date DATE NOT NULL;
```

```
ALTER TABLE tasks  
MODIFY  
    end_date  
    end_date DATE NOT NULL;
```

Constraints in MySQL

- UNIQUE
 - Values in a single attribute are different
 - Value groups in a group of attributes are different
- Creating a constraint:
 - Specify in CREATE TABLE for a single attribute
 - Add a CONSTRAINT cstr_name UNIQUE(attr1, attr2, ...)
 - Can leave out constraint name, will be replaced by an automatically created name
 - Use ALTER TABLE ADD CONSTRAINT

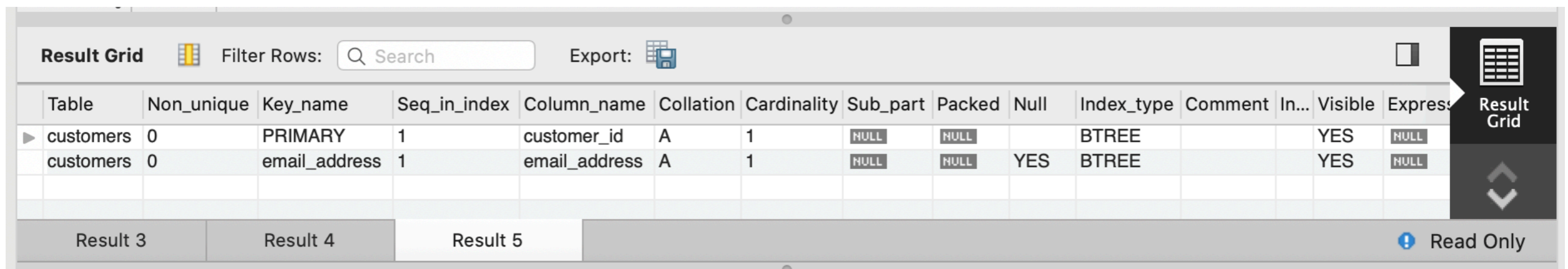
Constraints in MySQL

- UNIQUE

```
CREATE TABLE suppliers (  
    supplier_id INT AUTO_INCREMENT,  
    name VARCHAR(255) NOT NULL,  
    phone VARCHAR(15) NOT NULL UNIQUE,  
    address VARCHAR(255) NOT NULL,  
    PRIMARY KEY (supplier_id),  
    CONSTRAINT uc_name_address UNIQUE (name , address)  
);
```


Constraints in MySQL

- UNIQUE constraint creates an *index*
 - Index is a data structure with quick look-up
- Access indices through the SHOW INDEX FROM table command



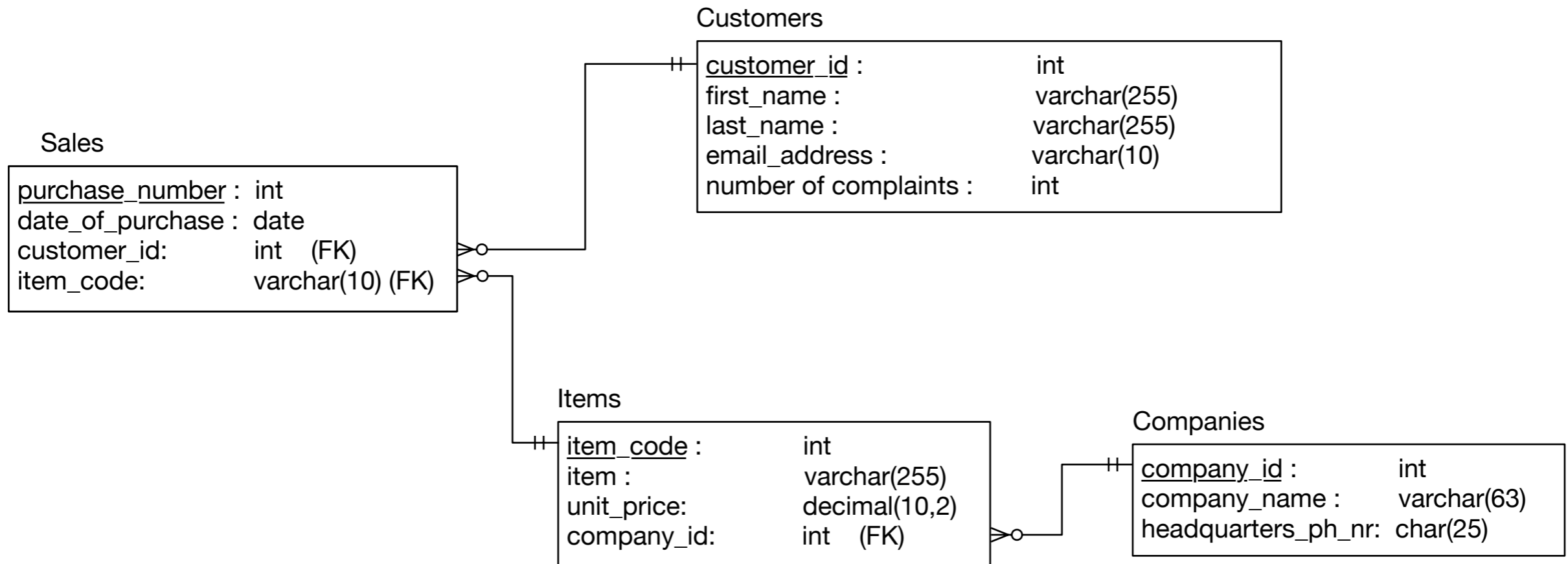
The screenshot shows a 'Result Grid' window with a search bar and an 'Export' button. The main area displays a table of index information for the 'customers' table. The table has 15 columns: Table, Non_unique, Key_name, Seq_in_index, Column_name, Collation, Cardinality, Sub_part, Packed, Null, Index_type, Comment, In..., Visible, and Express. Two rows are visible, both for the 'customers' table. The first row shows a PRIMARY index on 'customer_id' with a cardinality of 1. The second row shows a BTREE index on 'email_address' with a cardinality of 1 and is marked as 'VISIBLE'. The interface also shows 'Result 3', 'Result 4', and 'Result 5' tabs at the bottom, and a 'Read Only' status indicator.

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	In...	Visible	Express
customers	0	PRIMARY	1	customer_id	A	1	NULL	NULL		BTREE			YES	NULL
customers	0	email_address	1	email_address	A	1	NULL	NULL	YES	BTREE			YES	NULL

Foreign Keys

- Relationships between tables are sometimes constructed with shared values
 - Sales has an attribute `client_id`
 - Customers has a primary key `client_id`
 - Need not be named the same
 - But it is usually convenient to do so

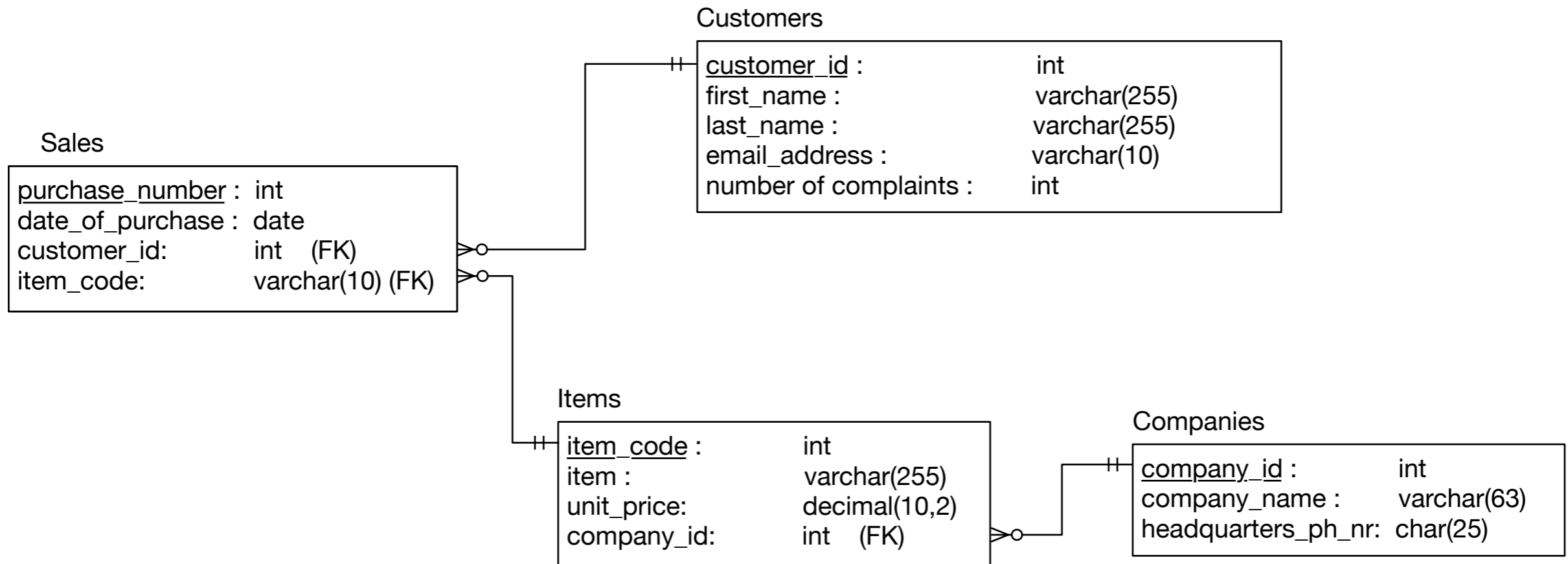
Constraints in MySQL



Constraints in MySQL

- Example:
 - A customer can have many sales
 - But each sale has only one customer
 - Relationship customers sales is a **one-to-many** relationship
 - customers is the referenced (or parent) table
 - sales is the referencing (or child) table
 - As is typical, the referenced attribute is a primary key in the referenced table

Constraints in MySQL



Constraints in MySQL

- In a diagram:
 - crow-feet with ball indicate many
 - double bar indicates one

Constraints in MySQL

- Foreign key constraint
 - Once established, insures that action is taken upon insertion or deletion of a record affecting the other table

Constraints in MySQL

- Possible Actions:
 - CASCADE: if a tuple from the referenced table is deleted or updated, the corresponding tuple in the referencing table is also deleted / updated
 - SET NULL: If a row from the referenced table is deleted or updated, the values of the foreign key in the referencing table are set to NULL

Constraints in MySQL

- Possible Actions:
 - RESTRICT: if a row from the referenced table has a matching row in the referencing table, then deletion and updates are rejected
 - SET DEFAULT: Accepted by MySQL parser but action not performed

Constraints in MySQL

- Foreign keys constraint actions
 - Are for
 - ON UPDATE
 - ON DELETE

Constraints in MySQL

- Creating foreign key constraints:

```
CREATE TABLE categories (  
    categoryId INT AUTO_INCREMENT PRIMARY KEY,  
    categoryName VARCHAR(100) NOT NULL  
);
```

```
CREATE TABLE products (  
    productId INT AUTO_INCREMENT PRIMARY KEY,  
    productName varchar(100) not null,  
    categoryId INT,  
    CONSTRAINT fk_category  
    FOREIGN KEY (categoryId)  
        REFERENCES categories(categoryId)  
        ON UPDATE CASCADE  
        ON DELETE CASCADE  
);
```

Constraints in MySQL

- You can drop a foreign key restraint using the ALTER TABLE statement

```
ALTER TABLE table_name  
DROP FOREIGN KEY constraint_name;
```

Constraints in MySQL

- When loading a database from (e.g.) .csv files
 - Can carefully create referenced tables before referencing tables
 - Temporarily disable foreign key checks

```
SET foreign_key_checks = 0;
```

```
SET foreign_key_checks = 1;
```

Insert Operations

- Insert Syntax

- No need to insert into automatic values

- If only a few attributes are set,

```
INSERT INTO  
table(attr1, attr2, ...)  
Values(v1, v2, ...)
```

- If all attributes are set, just list the values

- Can set many tuples at once

```
INSERT INTO served
```

```
VALUES
```

```
('William Howe', 'Great Britain', '1746-1-1', '1778-4-1'),  
( 'Benedict Arnold', 'Great Britain', '1757-1-1', '1775-1-1'),  
( 'Benedict Arnold', 'United States', '1775-1-1', '1780-9-1'),  
( 'Benedict Arnold', 'Great Britain', '1780-9-1', '1787-1-1')
```

Select

Select

- In order to avoid having to prefix the database name to tables, use the Use command:
 - `USE classicmodels;`

Select

- `SELECT * FROM table`
- `SELECT col1, col2 FROM table`
- `SELECT * FROM table WHERE conditions`

Select

```
thomasschwarz — mysq
mysql> SELECT lastName FROM employees;
+-----+
| lastName |
+-----+
| Murphy   |
| Patterson |
| Firrelli |
| Patterson |
| Bondur   |
| Bow      |
| Jennings |
| Thompson |
| Firrelli |
| Patterson |
| Tseng    |
| Vanauf   |
| Bondur   |
| Hernandez |
| Castillo |
| Bott     |
| Jones    |
| Fixter   |
| Marsh    |
| King     |
```

Select

- You do not need to specify a table to obtain values

```
[mysql> SELECT 2*3+1;
+-----+
| 2*3+1 |
+-----+
|      7 |
+-----+
1 row in set (0.01 sec)
```

```
[mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2023-02-22 21:01:03 |
+-----+
1 row in set (0.00 sec)
```

Select

- To make SELECT list work you can use the dummy table name dual
- To rename expressions, use AS

```
[mysql> SELECT firstName AS baptismal_name  
[    -> FROM  
[    -> employees;  
+-----+  
| baptismal_name |  
+-----+  
| Diane          |  
| Mary           |  
| Jeff           |  
| William        |  
| Gerard         |  
| Anthony        |  
| Leslie         |  
| Leslie         |  
| Julie          |  
| Steve          |  
| Frank         |
```

Select

```
thomasschwarz — mysql -u root -p — 93x33
mysql> SELECT
  ->     CONCAT_WS(' ', firstname, lastName) AS 'Full Name'
  -> FROM
  ->     employees;
+-----+
| Full Name |
+-----+
| Diane Murphy |
| Mary Patterson |
| Jeff Firrelli |
| William Patterson |
| Gerard Bondur |
| Anthony Bow |
| Leslie Jennings |
| Leslie Thompson |
| Julie Firrelli |
| Steve Patterson |
| Foon Yue Tseng |
| George Vanauf |
| Loui Bondur |
| Gerard Hernandez |
| Pamela Castillo |
| Larry Bott |
| Barry Jones |
| Andy Fixter |
| Peter Marsh |
| Tom King |
| Mami Nishi |
| Yoshimi Kato |
| Martin Gerard |
+-----+
23 rows in set (0.00 sec)
```

Select

- Use ordering with ORDER BY and ASC / DESC

```
thomasschwarz — mysql -u root -p — 80x24
mysql> SELECT firstName, lastName, email
  -> FROM employees
  -> ORDER BY
  -> lastName DESC,
  -> firstName ASC;
```

firstName	lastName	email
George	Vanauf	gvanauf@classicmodelcars.com
Foon Yue	Tseng	ftseng@classicmodelcars.com
Leslie	Thompson	lthompson@classicmodelcars.com
Mary	Patterson	mpatterso@classicmodelcars.com
Steve	Patterson	spatterson@classicmodelcars.com
William	Patterson	wpatterson@classicmodelcars.com
Mami	Nishi	mnishi@classicmodelcars.com
Diane	Murphy	dmurphy@classicmodelcars.com
Peter	Marsh	pmarsh@classicmodelcars.com
Tom	King	tking@classicmodelcars.com
Yoshimi	Kato	ykato@classicmodelcars.com
Barry	Jones	bjones@classicmodelcars.com
Leslie	Jennings	ljennings@classicmodelcars.com
Gerard	Hernandez	ghernande@classicmodelcars.com
Martin	Gerard	mgerard@classicmodelcars.com
Andy	Fixter	afixter@classicmodelcars.com

Select

```
[mysql> SELECT  
[   -> firstName, lastName, reportsTo  
[   -> FROM  
[   -> employees  
[   -> ORDER BY reportsTo DESC;
```

firstName	lastName	reportsTo
Yoshimi	Kato	1621
Leslie	Jennings	1143
Leslie	Thompson	1143
Julie	Firrelli	1143
Steve	Patterson	1143
Foon Yue	Tseng	1143
George	Vanauf	1143
Loui	Bondur	1102
Gerard	Hernandez	1102
Pamela	Castillo	1102
Larry	Bott	1102
Barry	Jones	1102
Martin	Gerard	1102
Andy	Fixter	1088
Peter	Marsh	1088
Tom	King	1088
William	Patterson	1056
Gerard	Bondur	1056
Anthony	Bow	1056
Mami	Nishi	1056
Mary	Patterson	1002
Jeff	Firrelli	1002
Diane	Murphy	NULL

NULL is always smallest



```
23 rows in set (0.00 sec)
```

Select

- We use a WHERE clause in order to specify search conditions
 - Employees whose job title is 'Sales Rep'

```
[mysql> SELECT  
[   -> firstName, lastName  
[   -> FROM  
[   -> employees  
[   -> WHERE  
[   -> jobtitle = 'Sales Rep';
```

firstName	lastName
Leslie	Jennings
Leslie	Thompson
Julie	Firrelli
Steve	Patterson
Foon Yue	Tseng
George	Vanauf
Loui	Bondur
Gerard	Hernandez
Pamela	Castillo
Larry	Bott
Barry	Jones

SELECT

- There are a number of comparison operators:
 - = equals (comparison operator)
 - AND, OR
 - IN, NOT IN
 - LIKE, NOT LIKE
 - BETWEEN ... AND
 - EXISTS, NOT EXISTS
 - IS NULL, IS NOT NULL

Select

- Examples:



```
[mysql> SELECT firstName, lastName
[      -> FROM employees
[      -> WHERE reportsTo IS NULL;
+-----+-----+
| firstName | lastName |
+-----+-----+
| Diane     | Murphy   |
+-----+-----+
1 row in set (0.00 sec)
```

Select

```
[mysql> SELECT contactLastName AS 'Last Name', contactFirstName AS 'First Name', phone  
[   -> FROM customers  
[   -> WHERE country = 'Germany';
```

Last Name	First Name	phone
Keitel	Roland	+49 69 66 90 2555
Kloss	Horst	0372-555188
Messner	Renate	069-0555984
Pfalzheim	Henriette	0221-5554327
Franken	Peter	089-0877555
Andersen	Mel	030-0074555
Cramer	Philip	0555-09555
Josephs	Karin	0251-555259
Müller	Rita	0711-555361
Donnermeyer	Michael	+49 89 61 08 9555
Feuer	Alexander	0342-555176
Ottlieb	Sven	0241-039123
Moos	Hanna	0621-08555

```
13 rows in set (0.00 sec)
```

Comparisons with NULL

- NULL in any expression gives NULL
 - If you compare anything with NULL in MySQL, you get NULL
- In other SQL dialects: depends

SELECT

- LIKE
 - Pattern matching
 - Wild cards
 - % means zero or more characters
 - _ means a single letter
 - [] means any single character within the bracket
 - ^ means any character not in the bracket
 - - means a range of characters

Like Examples

- WHERE name LIKE 't%'
 - any values that start with 't'
- WHERE name LIKE '%t'
 - any values that end with 't'
- WHERE name LIKE '%t%'
 - any value with a 't' in it
- WHERE name LIKE '_t%'
 - any value with a 't' in second position

Select

- Beware of bad data when you make searches

```
[mysql> SELECT contactLastName AS 'Last Name', contactFirstName AS 'First Name', phone  
[   -> FROM customers  
[   -> WHERE phone LIKE '+49 %';
```

Last Name	First Name	phone
Keitel	Roland	+49 69 66 90 2555

1 row in set (0.00 sec)

```
[mysql> SELECT contactLastName AS 'Last Name', contactFirstName AS 'First Name', phone  
[   -> FROM customers  
[   -> WHERE phone LIKE '%+49 %';
```

Last Name	First Name	phone
Keitel	Roland	+49 69 66 90 2555
Donnermeyer	Michael	+49 89 61 08 9555

2 rows in set (0.00 sec)

SELECT

- BETWEEN ... AND ...
 - Selects records with a value in the range
 - endpoints included

```
mysql> SELECT orderNumber, orderDate, requiredDate, shippedDate  
-> FROM orders  
[ -> WHERE requiredDate between '2003-1-1' AND '2003-2-1';
```

orderNumber	orderDate	requiredDate	shippedDate
10100	2003-01-06	2003-01-13	2003-01-10
10101	2003-01-09	2003-01-18	2003-01-11
10102	2003-01-10	2003-01-18	2003-01-14

```
3 rows in set (0.01 sec)
```


SELECT

- SELECT DISTINCT

```
[mysql> SELECT DISTINCT country FROM customers;
```

```
+-----+  
| country |  
+-----+  
| France |  
| USA    |  
| Australia |  
| Norway |  
| Poland |  
| Germany |  
| Spain  |  
| Sweden |  
| Denmark |  
| Singapore |  
| Portugal |  
| Japan   |  
| Finland |  
| UK      |  
| Ireland |  
| Canada  |  
| Hong Kong |  
| Italy    |  
| Switzerland |  
| Netherlands |  
| Belgium  |  
| New Zealand |  
| South Africa |  
| Austria  |  
| Philippines |  
| Russia   |  
| Israel   |  
+-----+
```

```
27 rows in set (0.00 sec)
```

SELECT

- LIMIT gives the maximum number of rows returned
 - Can be used for a sample
 - Can be used with ORDER BY ASC



**Queries with more
than one Table**

Naming Tables

- We can name tables in the WHERE clause

```
SELECT
    e.firstName,
    e.lastName
FROM
    employees e
ORDER BY e.firstName;
```

Simple Joins

- Cartesian product of two tables is called CROSS JOIN:

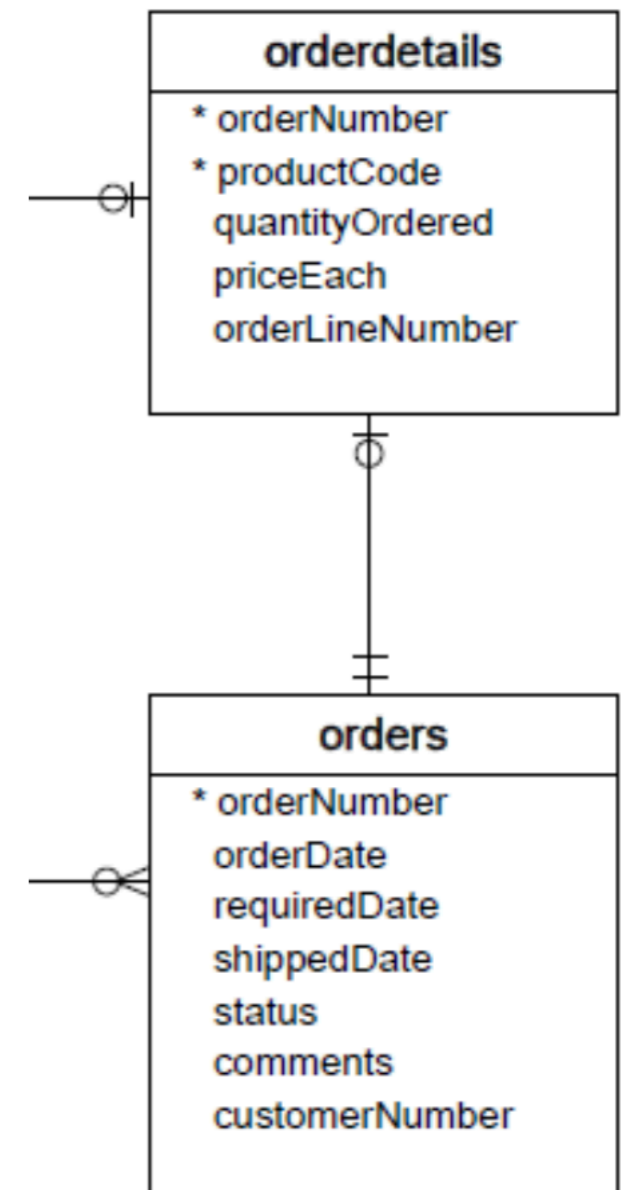
```
SELECT
    *
FROM
    offices
    CROSS JOIN
    products;
```

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory	productCode	productName	productLine	productScale	productVendor
▶	7	London	+44 20 7877 2041	25 Old Broad Street	Level 7	NULL	UK	EC2N 1HN	EMEA	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2	NULL	Australia	NSW 2010	APAC	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	5	Tokyo	+81 33 224 5000	4-1 Kioicho	NULL	Chiyoda-Ku	Japan	102-8578	Japan	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans	NULL	NULL	France	75017	EMEA	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA	S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast
	7	London	+44 20 7877 2041	25 Old Broad Street	Level 7	NULL	UK	EC2N 1HN	EMEA	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Crea
	6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2	NULL	Australia	NSW 2010	APAC	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Crea
	5	Tokyo	+81 33 224 5000	4-1 Kioicho	NULL	Chiyoda-Ku	Japan	102-8578	Japan	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Crea
	4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans	NULL	NULL	France	75017	EMEA	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Crea
	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Crea
	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA	S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Crea

Simple Joins

- You can convert a cross join to an inner join with a where clause

```
SELECT
    productcode, comments
FROM
    orderdetails orde
    CROSS JOIN
    orders ord
WHERE
    orde.ordernumber = ord.ordernumber
AND
    ord.comments IS NOT NULL;
```



Simple Joins

- But that just gives code harder to read

```
SELECT
    productcode, comments
FROM
    orderdetails orde
    INNER JOIN
    orders ord
ON
    orde.ordernumber = ord.ordernumber
WHERE
    ord.comments IS NOT NULL;
```

Simple Joins

- When the column names are the same, we can use the USING clause
 - Notice the parentheses

```
SELECT
    productcode, comments
FROM
    orderdetails orde
    INNER JOIN
    orders ord
USING
    (ordernumber)
WHERE
    ord.comments IS NOT NULL;
```


Simple Joins

- You can also use the pre-1992 SQL92 notation

```
SELECT
    productcode, comments
FROM
    orderdetails orde, orders ord
WHERE
    orde.orderNumber = ord.orderNumber
AND
    ord.comments IS NOT NULL;
```

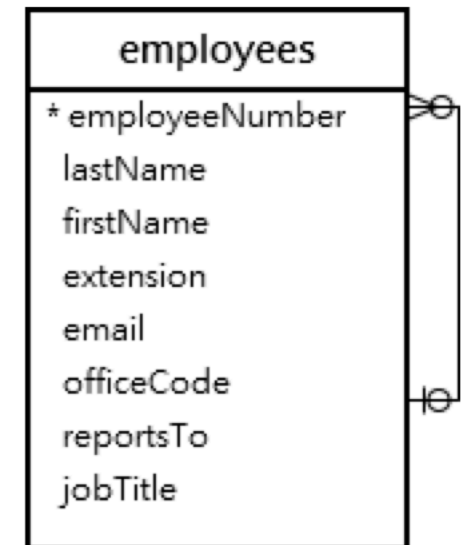
Simple Joins

- The SQL-92 is clearer whenever the joins are complex

```
SELECT
    customerName, city, cus.country,
    quantityOrdered*priceEach AS 'volume'
FROM
    customers cus
    INNER JOIN orders ord ON cus.customerNumber =
ord.customerNumber
    INNER JOIN orderdetails orddet ON orddet.orderNumber =
ord.orderNumber
WHERE
    ord.comments IS NOT NULL AND orddet.productCode =
'S18_2325'
;
```

Simple Joins

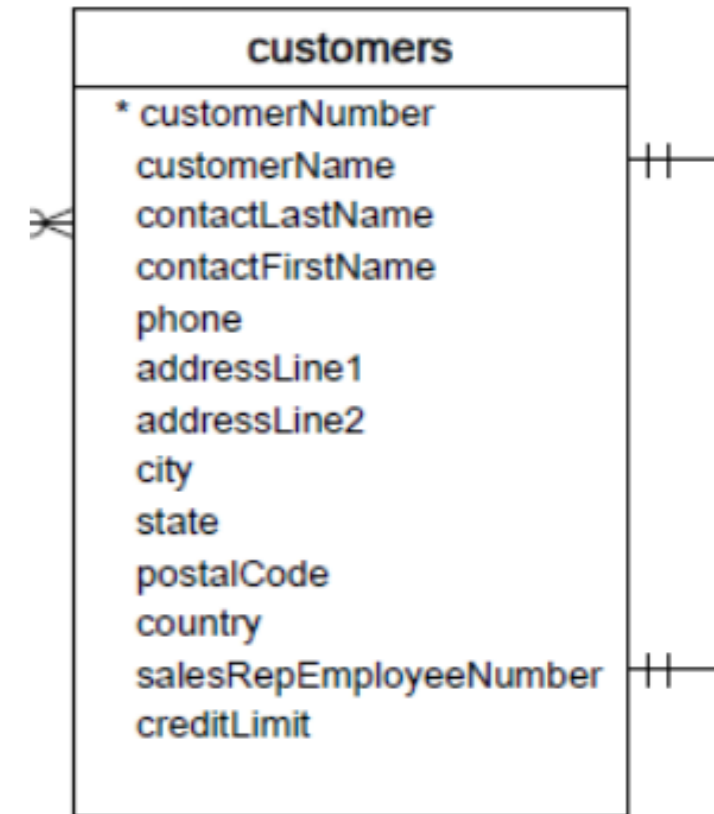
- Self-joins: Use different table aliases



```
SELECT
    CONCAT(m.firstName, ' ', m.lastName) AS manager,
    CONCAT(e.firstName, ' ', e.lastName) AS managee
FROM
    employees e
    INNER JOIN employees m
    ON
        m.employeeNumber = e.reportsTo
ORDER BY manager;
```

Simple Joins

- Find pairs of clients that are in the same city



```
SELECT
    c1.city, c1.customerName, c2.customerName
FROM
    customers c1 INNER JOIN customers c2 ON
        c1.city = c2.city
        AND c1.customerName > c2.customerName
ORDER BY
    c1.city
```

Examples

- SQL has explicit commands for the various joins and products
- Normally, combine tables by listing them in the FROM clause

```
SELECT name
FROM movies, moviesExec
WHERE title = 'Star Wars'
      AND movies.producerC# = moviesExec.cert#
```

Examples

- Find all movie execs that live with a star
- `MovieStar(name, address, gender, birthdate)`
`MovieExec(name, address, cert#, netWorth)`

```
SELECT MovieStar.name, MovieExec.name)
FROM MovieStar, MovieExec
WHERE
    MovieStar.address = MovieExec.address
```

Examples

- Tuple Variables
 - Sometimes need to combine two tuples in the same table
 - Can extend the FROM clause

```
SELECT Star1.name, Star2.name
FROM MovieStars Star1, MovieStars Star2
WHERE
    Star1.address = Star2.address
    AND Star1.name < Star2.name
```

Updates

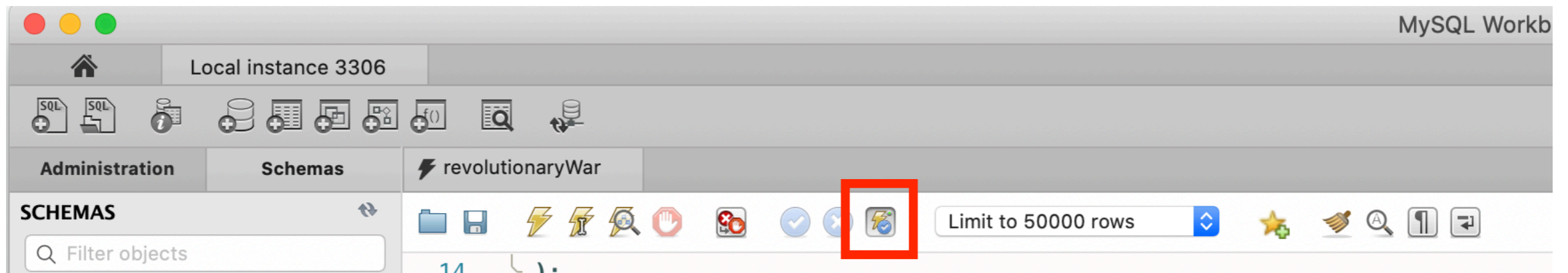
- Changes existing records
- Syntax:

```
UPDATE tablename  
SET attr1=val1, attr2=val2, ...  
WHERE conditions;
```

- Does not need to change all attributes
- If there is no WHERE condition, all records are updated

Commit and Rollback

- A database allows us to rollback to a previous state unless we have committed
- MySQLWorkbench has an auto-commit button



- Rollback puts database into the state of the last commit

Delete

- Just like an update

```
DELETE FROM tablename  
WHERE condition
```

- The Where clause is not necessary

Delete, Drop, Truncate

- Drop Table:
 - Definite action: cannot recover with rollback
- Truncate:
 - All records removed
 - Auto-increment values reset
 - Table description stays
- Delete:
 - Delete removes records row by row
 - Auto-increment values remain
 - Slower than truncate



Sub-Queries

Subqueries

- Subqueries are helper queries

Subqueries

- Subqueries producing a scalar value
 - Example: Producer of Star Wars

```
SELECT name
From movies, movieExec
WHERE title = 'Star Wars'
      AND
      producerC# = cert#;
```

- Can achieve the same effect by first looking for the producerC#

Subqueries

- Example: Producer of Star Wars

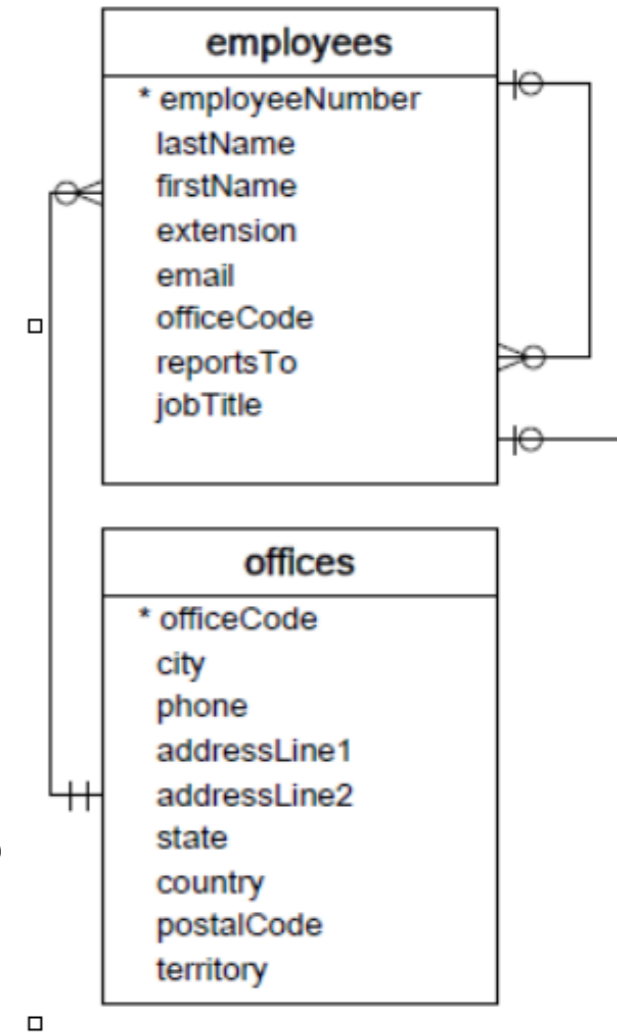
```
SELECT name
FROM movieExec
WHERE cert# =
    (SELECT producerC#
     FROM movies
     WHERE title = 'star wars'
    )
```

- While the queries are different, their execution can be the same

Subqueries

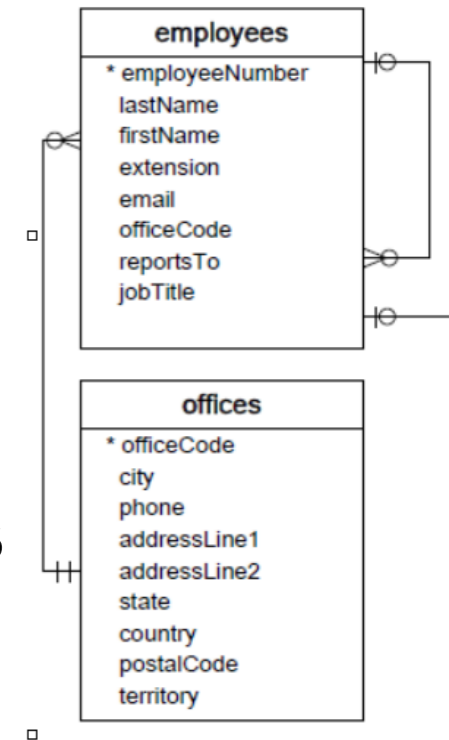
- You can create sub-tables
- Find employees working in the US
 - First: Find officeCodes with country = US

```
SELECT
    officeCode
FROM
    offices
WHERE
    country = 'USA';
```



Subqueries

- Second: Connect employees to these office codes

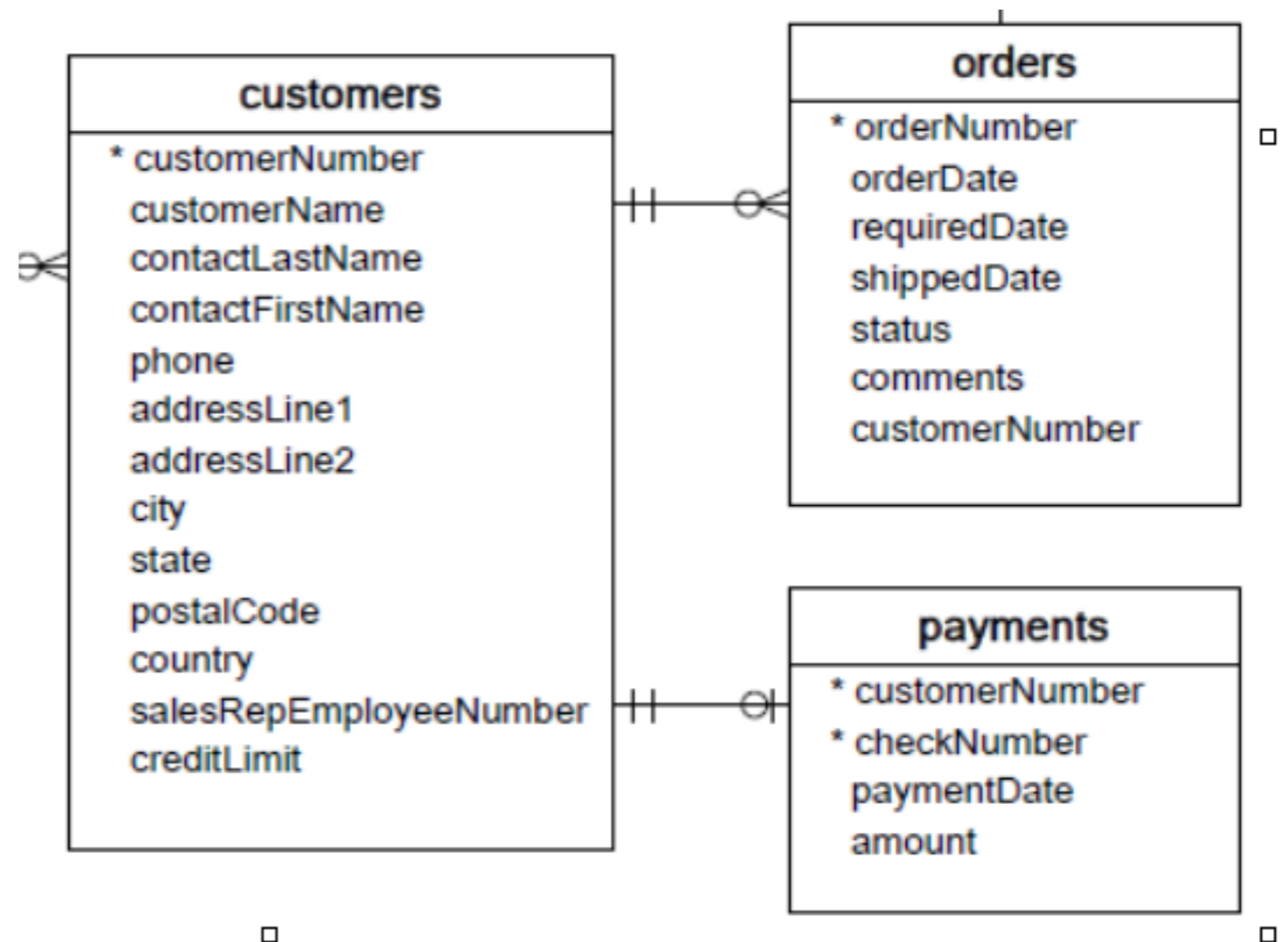


```
SELECT
    CONCAT(firstName, ' ', lastName) AS 'employee'
FROM
    employees
WHERE
    officeCode IN (
        SELECT
            officeCode
        FROM
            offices
        WHERE
            country = 'USA');
```

employee
▶ Diane Murphy
Mary Patterson
Jeff Firrelli
Anthony Bow
Leslie Jennings
Leslie Thompson
Julie Firrelli
Steve Patterson
Foon Yue Tseng
George Vanauf

Subqueries

- Find the contact that made the largest payment



Subqueries

- Step 1:
 - Need to find maximum payment

Subqueries

- Step 1:
 - Need to find maximum payment

```
SELECT MAX(amount) FROM payments
```

Subqueries

- Step 2:
 - Display the details

```
SELECT
    CONCAT(c.contactFirstName, ' ', c.contactLastName) AS
    'client contact', checkNumber, amount
FROM
    customers c, payments p
WHERE
    amount = (SELECT MAX(amount) FROM payments)
    AND c.customerNumber = p.customerNumber;
```

Subqueries

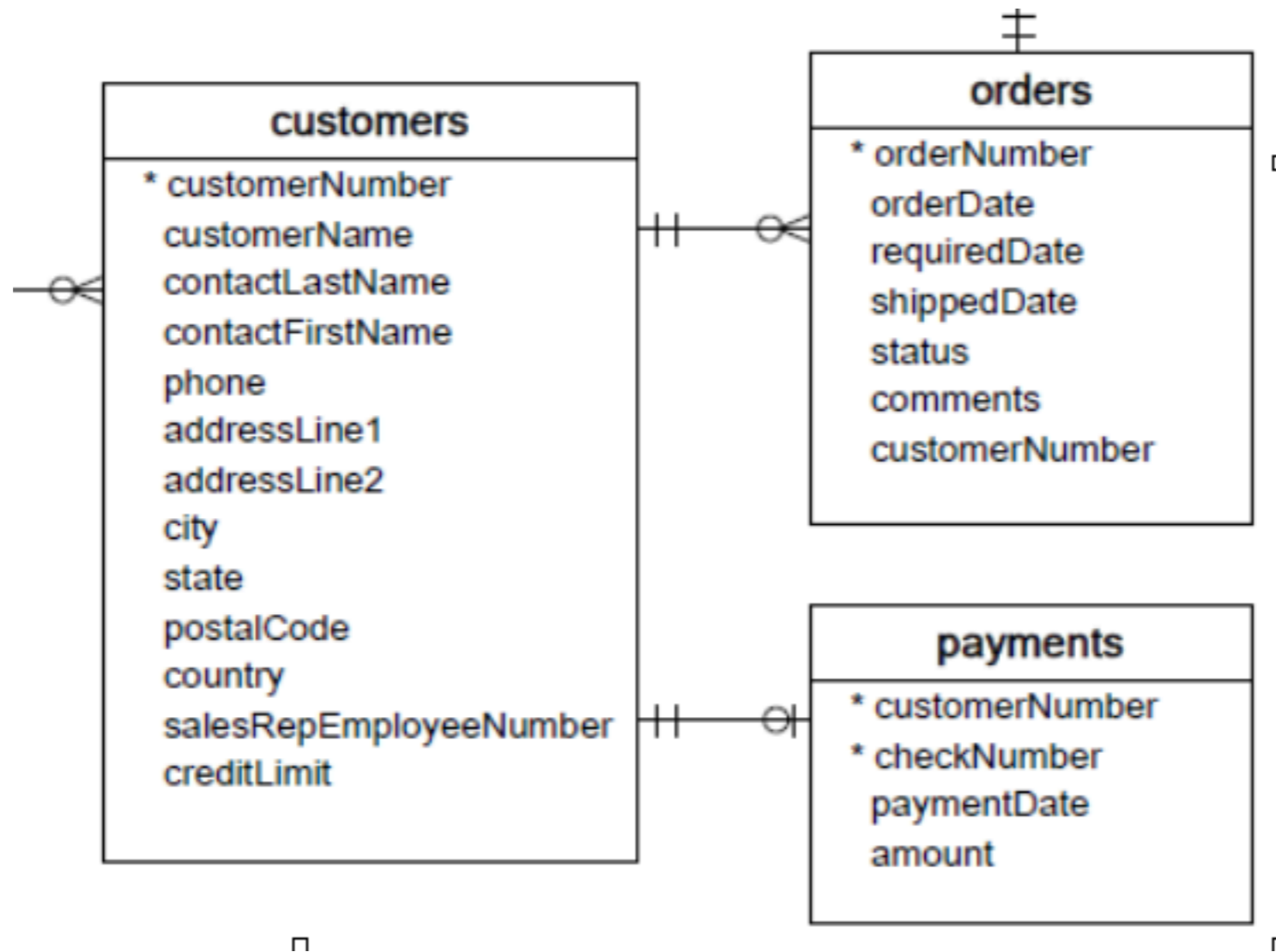
- Same, but payments larger than the average amount

```
SELECT
  CONCAT(c.contactFirstName, ' ', c.contactLastName) AS
  'client contact', checkNumber, amount
FROM
  customers c, payments p
WHERE
  amount > (SELECT AVG(amount) FROM payments) AND
  c.customerNumber = p.customerNumber;
```

	client contact	checkNumber	amount
▶	Jean King	HQ55022	32641.98
	Jean King	ND748579	33347.88
	Peter Ferguson	GG31455	45864.03
	Peter Ferguson	MA765515	82261.22
	Peter Ferguson	NR27552	44894.74
	Janine Labrune	LN373447	47924.19
	Janine Labrune	NG94694	49523.67
	Jonas Bergulfsen	DB889831	50218.95
	Jonas Bergulfsen	MA302151	34638.14
	Susan Nelson	AE215433	101244.59
	Susan Nelson	BG255406	85410.87
	Susan Nelson	ET64396	83598.04
	Susan Nelson	HI366474	47142.70
	Susan Nelson	HR86578	55639.66
	Susan Nelson	KI131716	111654.40
	Susan Nelson	LF217299	43369.30
	Susan Nelson	NT141748	45084.38
	Roland Keitel	FH668230	33820.62
	Kwai Lee	MA724562	50025.35
	Kwai Lee	NB445135	35321.97
	Diego Freyre	AU364101	36251.03
	Diego Freyre	DB583216	36140.38

Subqueries

- Find customers that did not order anything:
 - Find the connection!



Subqueries

- The set of customers with orders is given by
customerNumber

```
SELECT  
    customerNumber  
FROM  
    orders;
```


Subqueries

- We want customer information where the customer number is *not* in this set

```
SELECT *  
FROM customerS  
WHERE customerNumber NOT IN (SELECT  
    customerNumber  
FROM  
    orders) ;
```

Subqueries

- And then project

```
SELECT customerName,  
       concat(contactFirstName, ' ', contactLastName) AS contact,  
       city,  
       country  
FROM customers  
WHERE customerNumber NOT IN (SELECT  
       customerNumber  
FROM  
       orders);
```

Subqueries

- How big are orders?
 - ```
SELECT orderNumber, COUNT (orderNumber) AS items
FROM orderdetails
GROUP BY orderNumber;
```

| orderNumber | items |
|-------------|-------|
| 10100       | 4     |
| 10101       | 4     |
| 10102       | 2     |
| 10103       | 16    |
| ▶ 10104     | 13    |
| 10105       | 15    |
| 10106       | 18    |
| 10107       | 8     |
| 10108       | 16    |
| 10109       | 6     |
| 10110       | 16    |
| 10111       | 6     |
| 10112       | 2     |
| 10113       | 4     |
| 10114       | 10    |
| 10115       | 5     |
| 10116       | 1     |

# Subqueries

- Now find maximum, minimum, and average

```
SELECT
 MAX(items),
 MIN(items),
 AVG(items)
FROM
 (SELECT orderNumber, COUNT(orderNumber) AS items
 FROM orderdetails
 GROUP BY orderNumber) AS tempTable;
```

# Subqueries

- Notice that we need to give a name to the subtable

|   | MAX(items) | MIN(items) | AVG(items) |
|---|------------|------------|------------|
| ▶ | 18         | 1          | 9.1902     |

# Examples

- Unions, intersections, excepts
- To execute the corresponding set operations

- 

```
(SELECT name, address
FROM movieStars
WHERE gender = 'F'
)
```

**INTERSECT**

```
(SELECT name, address
FROM movieExecs
WHERE netWorth > 1000000
)
```

# Subqueries

- Subqueries with conditions involving relations
  - We obtain a relation  $R$  as a subquery
  - E.g. with subquery (SELECT \* FROM foobar)
  - Queries are:
    - EXISTS  $R$
    - $s \text{ IN } R$      $s \text{ NOT IN } R$
    - $s > \text{ALL } R$     NOT  $s > \text{ALL } R$
    - $s > \text{ANY } R$     NOT  $s > \text{ANY } R$

# Subqueries

- Subqueries involving tuples
  - Tuple is a list of scalar values
  - Can compare tuples with the same number of components
  - Example:
    - Finding the producers of 'Harrison Ford' movies



# Subqueries

```
SELECT name
FROM movieExec
WHERE cert# IN
 (SELECT producerC#
 FROM movies
 WHERE (title, year) IN
 (SELECT movieTitle, movieYear
 FROM StarsIn
 WHERE starName = 'Harrison Ford'
)
)
);
```

# Subqueries

- To analyze a query, start with the inmost query

```
SELECT name
FROM movieExec
WHERE cert# IN
 (SELECT producerC#
 FROM movies
 WHERE (title, year) IN
 (SELECT movieTitle, movieYear
 FROM StarsIn
 WHERE starName = 'Harrison Ford'
)
)
);
```

# Subqueries

- This query can also be written without nested subqueries

```
SELECT name
FROM movieExec, movies, starsIn
WHERE cert# = producerC#
 AND starsIn.title = movies.title
 AND starsIn.year = movie.year
 AND starName = 'Harrison Ford'
```

# Subqueries

- Correlated subqueries
  - Subquery is evaluated many times
    - Once for each value given
- Example

```
SELECT title
FROM movies Old
WHERE year < ANY (
 SELECT year
 FROM movies
 WHERE title = Old.title
);
```

# Subqueries

- Scoping rules
  - First look for the subquery and tables in that subquery
  - Then go to the nesting subquery
  - etc.



# Subqueries

- SQL JOIN expression
  - Explicit construction of various joins
    - CROSS JOIN (product)
    - NATURAL JOIN
    - FULL OUTER JOIN
    - NATURAL FULL OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN

# Subqueries

- Examples

```
movies FULL OUTER JOIN starsIn ON
movies.title = stars.
```



# Subqueries

- Examples

```
movieStar(name, address, gender, birthday)
```

```
movieExec(name, address, cert#, netWorth)
```

```
movieStar NATURAL FULL OUTER JOIN movieExec(
 name, address, gender, birthday, cert#, netWorth)
```

# Eliminating Duplicates

- Use Distinct

```
SELECT DISTINCT name
FROM movies
```

- Warning: Invoking distinct is costly

# Eliminating Duplicates

- Union, intersection, difference usually remove duplicates automatically
- If we do not want this, but bag semantics:
  - Use the keyword all

```
(SELECT title, year
FROM movies)
UNION ALL
(SELECT movieTitle AS title,
 movieYear AS year
FROM
starsIn);
```

# Aggregate Functions

- COUNT
  - numeric and non-numeric data
  - null values excepted
- SUM, MIN, MAX, AVG - only numeric data
- Exercise: Find the number of different stars in the starsIn table

```
SELECT COUNT(DISTINCT name)
FROM starsIn
```

# Aggregate Functions

- Find the combined net-worth of movieExecs

```
SELECT SUM(networth)
FROM movieExecs
```

- Find the average net-worth of movieExecs

```
SELECT ROUND(AVG(networth), 2)
FROM movieExecs
```

# Aggregate Functions

- Dealing if NULL values
  - IFNULL(EXPR1, EXPR2):
    - Gives EXPR1 if it is not NULL and EXPR2 if not
- ```
SELECT
    name,
    IFNULL(studio, 'not president') AS studio
FROM movieExecs;
```

Aggregate Functions

- COALESCE(EXPR1, EXPR2, EXPR3, ... EXPRn)
 - Gives first nonNULL expression

Grouping

- Aggregation happens usually with grouping
 - To group, use GROUP BY followed by a WHERE clause

```
SELECT studioName, SUM(length) AS totalRunTime  
FROM movies  
GROUP BY studioName;
```


Grouping

- Example
 - Computing the total run time of movies produced by a producer

```
SELECT name, SUM(length) AS totalRunTime
FROM MovieExec, Movies
WHERE producerC# = cert#
GROUP BY name;
```

Grouping

- Aggregation and Nulls
 - NULL does not contribute to a sum, average, or count
- Grouping and Nulls
 - NULL is an ordinary value for grouping purposes
- Aggregation except COUNT over an empty bag gives result NULL

Transactions

Transactions

- Databases have to process many operations in parallel
- This means some support for inter-process communication
 - Usually provided by logging
- DBMS differ in what they provide
 - Serializability:
 - All transactions appear to have been executed one after the other

Transactions

- Atomicity
 - A single query is never interrupted:
 - Example:
 - A transfer of money from one account to another is executed completely or not at all
 - Both accounts have changed or none

Transactions

- Transaction
 - A group of SQL statements that are all processed in the order given or not at all
- SQL:
 - START TRANSACTION
 - either
 - COMMIT
 - ROLLBACK

Transactions

- Read only transactions
 - By declaring a transaction as read-only, SQL can usually perform it quicker
 - SET TRANSACTION READ ONLY;
 - SET TRANSACTION READ WRITE;

Transactions

- Dirty Reads:
 - Reading a record from an update that will be rolled-back
- Are dirty reads bad?
 - Depends
 - Sometimes, it does not matter, and we do not want the DBMS spend time on making sure that there are no dirty reads
 - Sometimes, a dirty read can absolutely mess up things
 - Selling the same commodity to two customers, ...

Transactions

- SQL Isolation Levels:
 - Allow dirty reads:
 - SET TRANSACTION READ WRITE
 - SET ISOLATION LEVEL READ UNCOMMITTED

Transactions

- SQL Isolation Levels:
 - Allow reads only of committed data:
 - SET TRANSACTION READ WRITE
 - SET ISOLATION LEVEL READ COMMITTED

Transactions

- SQL Isolation Levels:
 - Disallow dirty reads, but insure that the reads are consistent:
 - SET TRANSACTION READ WRITE
 - SET ISOLATION LEVEL READ REPEATABLE READ

Transactions

- SQL Isolation Levels:
 - Serializability (default):
 - SET TRANSACTION READ WRITE
 - SET TRANSACTION ISOLATION LEVEL SERIALIZABLE