CSC 106 - Fall 2018

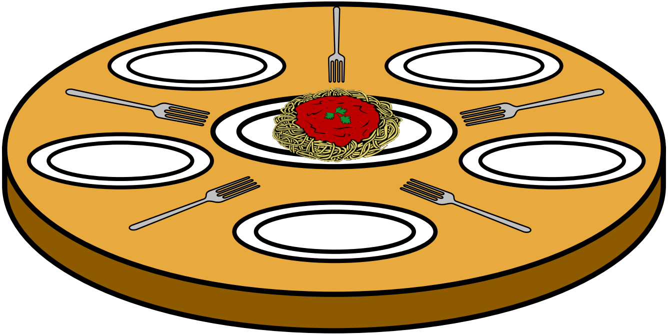
Databases

Bill Bird

Department of Computer Science
University of Victoria

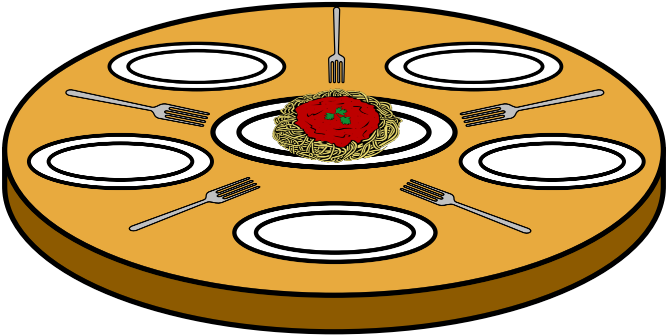
November 15, 2018

Dining Philosophers (1)



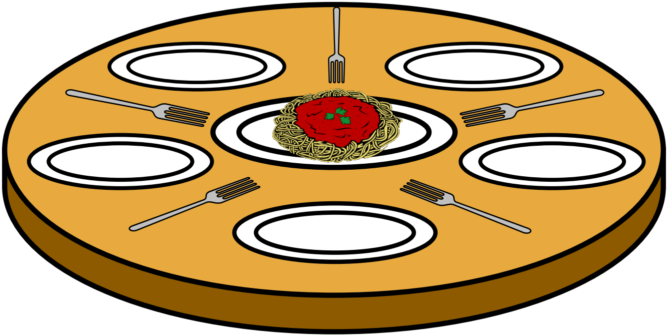
Consider a dinner party where the five guests serve themselves spaghetti from a shared plate. In order to take any spaghetti, each person must use two forks, but there are only five forks on the table.

Dining Philosophers (2)



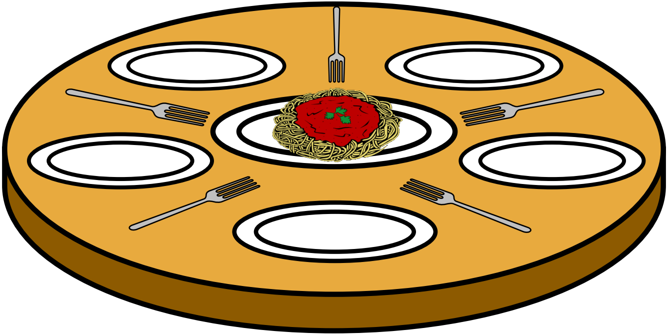
This is the *Dining Philosophers Problem*, which is an example of a **resource management** problem. Resource management is a key concern for distributed computing.

Dining Philosophers (3)



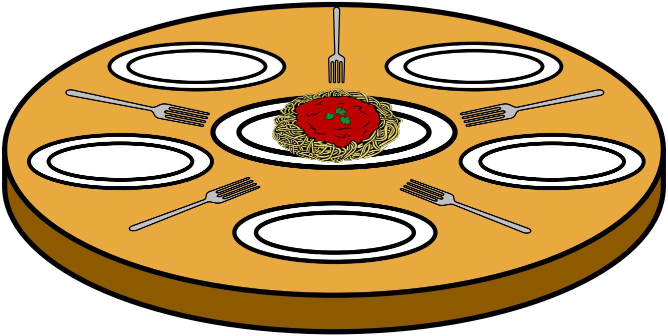
Hosting advice: When planning a dinner party, purchase enough cutlery for all guests.

Dining Philosophers (4)



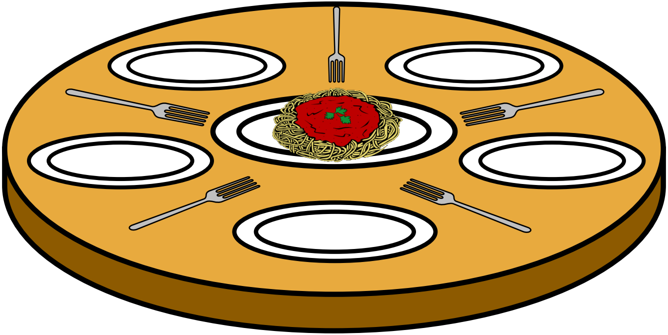
Culinary advice: When planning a dinner party, consider serving a more upscale main course than spaghetti.

Dining Philosophers (5)



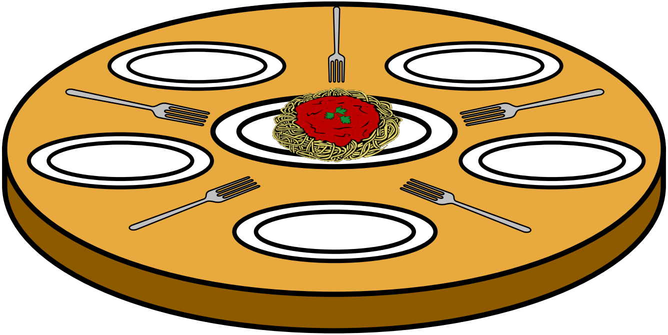
Etiquette Advice: It is generally not acceptable to use two forks at once; moreover, the use of a neighbouring diner's utensils is normally considered to be a faux pas.

Dining Philosophers (6)



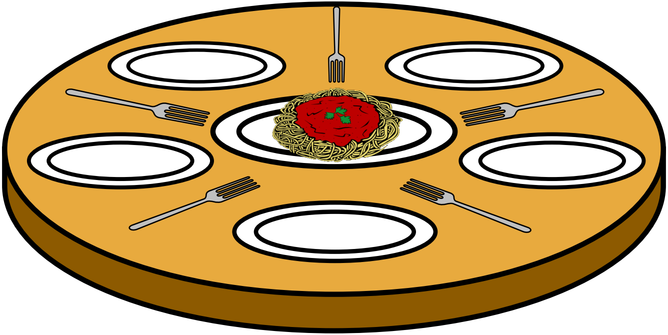
Question: How should we handle cases where two guests need the same utensil? What if every guest picks up the fork to their left and refuses to put it down?

Dining Philosophers (7)



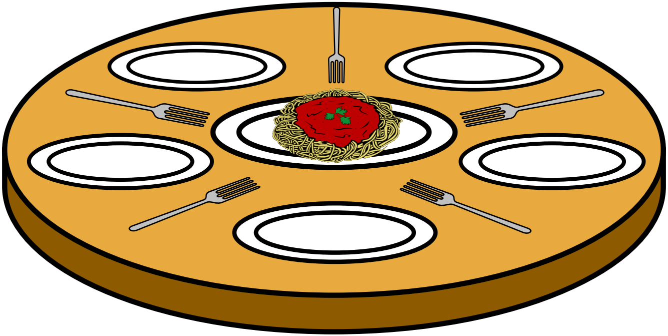
If every guest picks up the fork to their left, no guest can get both forks, and therefore no guest can serve themselves. If every guest waits for the other fork to become available, no progress will ever be made. This is known as a **deadlock**.

Dining Philosophers (8)



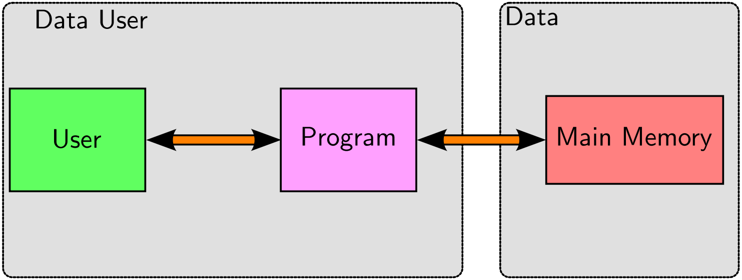
When several independent processes or services require access to the same shared computational resource, extra logic is required to prevent a deadlock from occurring. A deadlock can result in **starvation**, which occurs when a process is unable to continue due to a lack of a required resource.

Dining Philosophers (9)



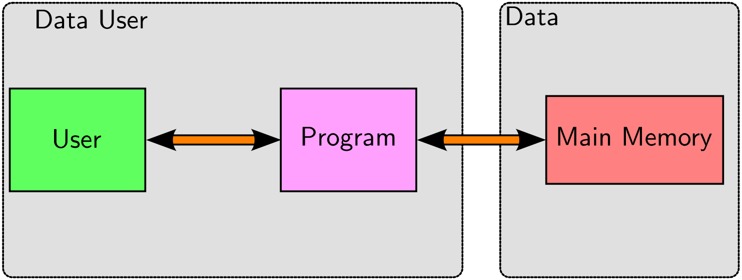
Deadlocks and resource management are core aspects of the study of **concurrency**, and the programmatic remedies for concurrency issues (or **data hazards**) tend to be complex.

Programs and Data (1)



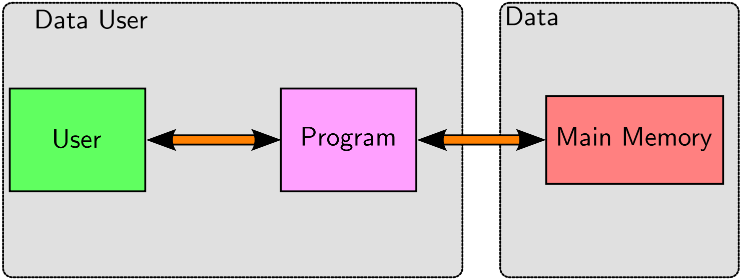
Most simple programs use a data model similar to the one above. The program works with a collection of data stored in main memory.

Programs and Data (2)



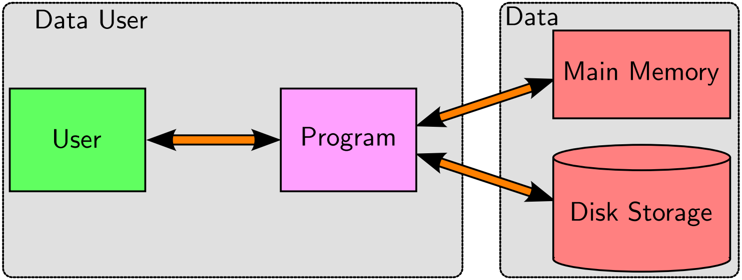
The data in memory may have originated from a secondary medium (such as a disk), and may be saved to disk occasionally, but all of the program logic works exclusively with data in memory.

Programs and Data (3)



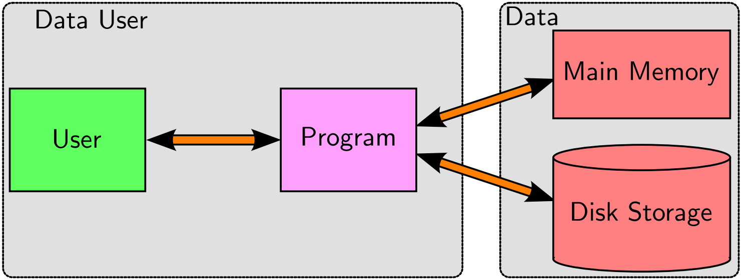
Nothing about this model is inherently wrong, but the amount of data that the program can manage is limited by the capacity of memory.

Programs and Data (4)



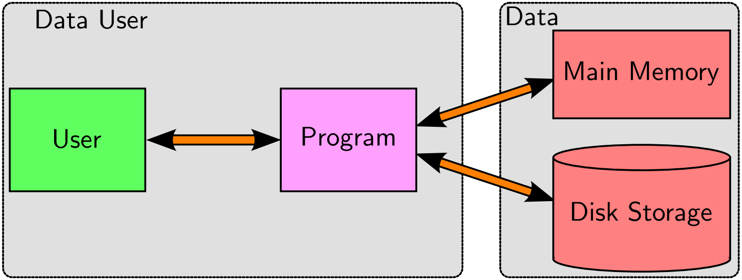
When the capacity of main memory is insufficient, data-intensive programs must rely on a combination of memory and secondary storage (traditionally hard disks).

Programs and Data (5)



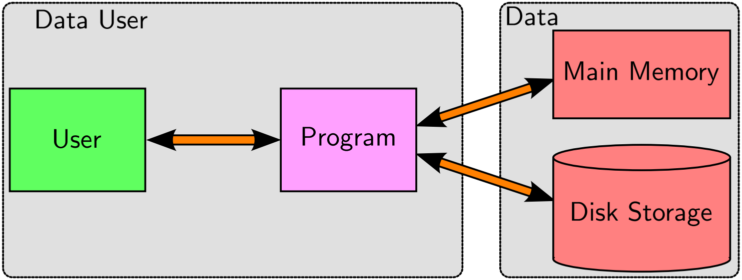
Another bottleneck arising from the use of large quantities of data is the need for advanced algorithms to organize and manage the data (for example, to allow easy retrieval of specific records).

Programs and Data (6)



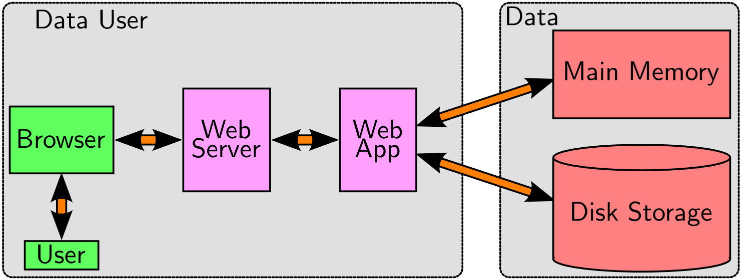
Database systems provide a way of separating the data user from the data itself, with an interface that allows data operations to be performed easily and efficiently.

Programs and Data (7)



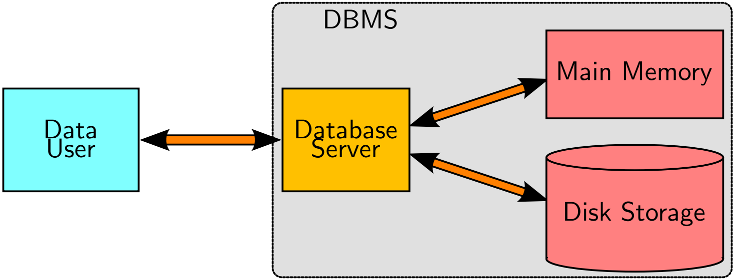
For the purposes of database systems, the exact nature of the data user is not always relevant. In the example above, the user directly interacts with a program that directly interacts with the data.

Programs and Data (8)



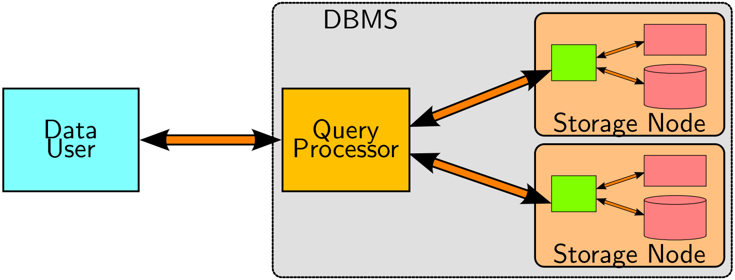
In the above example, the actual data usage occurs several steps away from the user. In the following examples, we will combine the entire ‘data user’ aspect into a single entity.

Programs and Data (9)



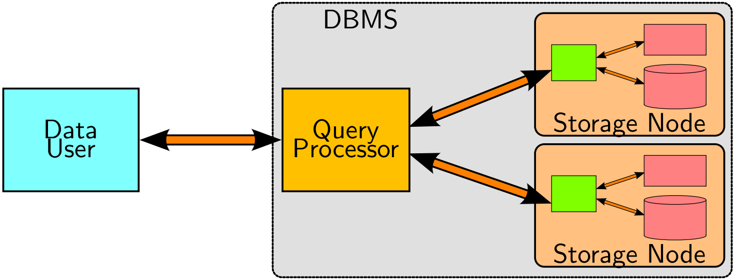
A **database management system** (DBMS) provides a mechanism for data storage, manipulation and retrieval. The example above gives a basic DBMS model, where the data user interfaces with a database server, which manages the data itself.

Programs and Data (10)



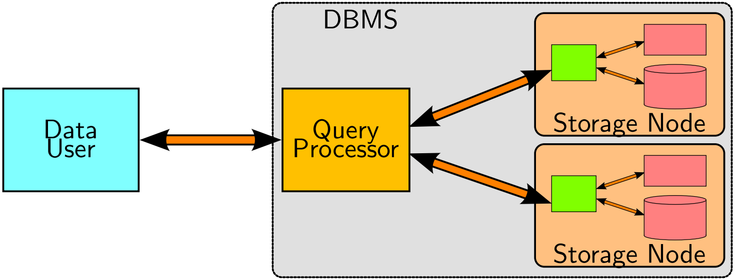
One of the advantages of this model is the ability for the structure of the data itself to be changed based on the needs of the system. In the above example, the data is stored on two independent nodes instead of at a single location.

Programs and Data (11)



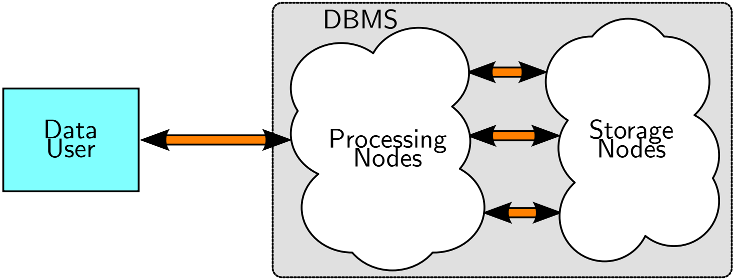
Although speed and convenience are desirable, the quality of a DBMS is often judged based on information security attributes (such as confidentiality, integrity and availability).

Programs and Data (12)



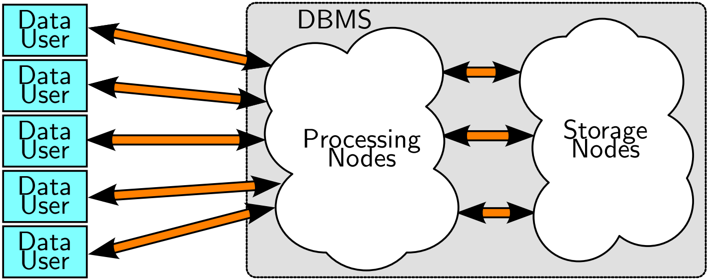
If both of the storage nodes store identical information, integrity and availability can be increased. If the nodes are used to store different information (for example, if each node stores half of the data), speed can be increased.

Programs and Data (13)



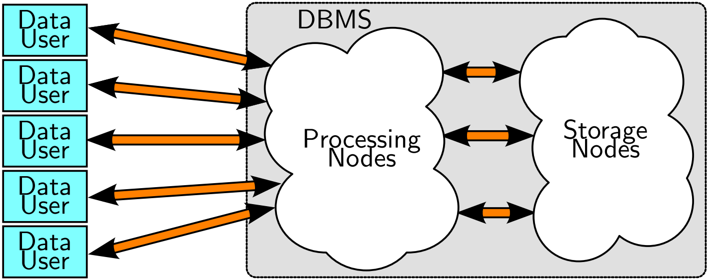
As performance needs increase, the DBMS can be distributed among an arbitrary number of computing units to share the storage and processing load. The outward interface for the data user does not change.

Programs and Data (14)



Another benefit of a DBMS is the ability to interface with multiple data users simultaneously. The DBMS is responsible for mediating resource management issues between the different users, so no extra consideration for concurrency is needed when writing a database client.

Programs and Data (15)



As a result, a DBMS model can be useful when scalability is desirable. (Often, even when scalability is not an immediate concern, it is useful to design software that can be scaled easily)

Relational Databases (1)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

A **relational database** classifies data into **relations**, which are normally called **tables** in a DBMS. The entries of a table are **rows** or **records**.

Relational Databases (2)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

Note that while relational databases are very widely used, there are other database models which are also useful. However, we will focus exclusively on relational databases and a query language for relational databases called SQL.

Relational Databases (3)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

The three tables above specify a set of products (sold by weight in kilograms) and orders (where each order is placed by a customer and may include multiple products).

Relational Databases (4)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

Notice that the data for each order is distributed among the three tables. For example, the orders table does not contain any information about which products were part of each order.

Relational Databases (5)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

This distribution is the result of **relational decomposition**. The specifics of decomposition are covered in higher level courses.

Relational Databases (6)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

However, the reasons for decomposition may be clear from the example above. One reason is to eliminate duplication: Each piece of data (such as the product name, customer name or number of units bought) is stored in only one place. Only ID numbers are duplicated between tables.

Relational Databases (7)

Table products		
`product_id`	`name`	`price_kg`
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
`order_num`	`product_id`	`kg_bought`
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
`order_num`	`order_date`	`customer_lastname`	`customer_firstname`
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

Eliminating duplication saves storage space and processing time, and also makes the data easier to update, since there are fewer redundant records to change.

Relational Databases (8)

Table products		
`product_id`	`name`	`price_kg`
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
`order_num`	`product_id`	`kg_bought`
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
`order_num`	`order_date`	`customer_lastname`	`customer_firstname`
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

The specifics of how each relation is represented in memory or on disk are up to the DBMS software. The user of the database does not have direct access to the storage. Instead, the user communicates with the DBMS using a **query language** to search and change the data.

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

For relational databases, **structured query language** (SQL) is often used as the query language. (SQL may be pronounced as an initialism or as the word ‘sequel’)

Relational Databases (10)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

There are several widely used database systems which support SQL. In this course, we will use a portable database engine called `sqlite3` (which is intended for local databases, instead of network-based databases).

Relational Databases (11)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

SQL has several parts, corresponding to various operations (searching, updating data and database maintenance). We will focus mainly on the `select` statement, which is used to search for data.

Relational Databases (12)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

Tables can be created with the `create` statement, specifying the name and type of each column. For example,

```
create table products( product_id int, name text, price_kg real);
```

Relational Databases (13)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

(A file containing all of the SQL code used in these slides has been posted to conneX. Copying and pasting the code directly from the slides may lead to errors due to character set issues.)

Relational Databases (14)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0
5	Peach	6.1

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

To add data to a table, the insert statement can be used, along with the data for the new row. For example, to add the highlighted row above,

```
insert into products values(5 ,'Peach', 6.10);
```

Relational Databases (15)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

To retrieve data from the database, the `select` statement can be used. The `select` statement is essentially its own programming language, since the set of semantics is so vast (it is not, in general, Turing complete, though).

Relational Databases (16)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

The language for `select` statements is an example of a **declarative** language, since queries only specify what is to be done, not how.

SQL Queries (1)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

All of the queries in the following slides use the database schema above, with the data shown. The output shown was produced using `sqlite3` (but the SQL commands used should work with any SQL-based DBMS).

SQL Queries (2)

Query Result			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

Query

```
select * from orders;
```

The most basic `select` statement selects every row from a table. The $*$ character is used to select all columns.

SQL Queries (3)

Query Result	
customer_firstname	customer_lastname
Leopold	Bloom
Gregor	Samsa
Bill	Bird

Query

```
select customer_firstname, customer_lastname
    from orders;
```

Instead of a $*$, a list of columns can be specified.

SQL Queries (4)

Query Result	
customer_firstname	customer_lastname
Bill	Bird
Leopold	Bloom
Gregor	Samsa

Query

```
select customer_firstname, customer_lastname
    from orders
    order by customer_lastname;
```

The `order by` directive can be used to sort the resulting data by a specific column's value.

SQL Queries (5)

Query Result	
customer_firstname	customer_lastname
Gregor	Samsa
Leopold	Bloom
Bill	Bird

Query

```
select customer_firstname, customer_lastname
    from orders
    order by customer_lastname desc;
```

The desc qualifier can be used to sort the results in descending order instead of defaulting to ascending order (which corresponds to the asc directive).

SQL Queries (6)

Query Result		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Query

```
select * from order_contents;
```

The results of a query can also be filtered by conditional expressions.

SQL Queries (7)

Query Result		
order_num	product_id	kg_bought
1001	1	2.5
1002	2	5.0
1002	4	20.0

Query

```
select * from order_contents
    where order_num >= 1001;
```

A list of conditions can be specified after the `where` directive. The query above prints all order contents for order numbers which are at least 1001.

SQL Queries (8)

Query Result		
order_num	product_id	kg_bought
1002	2	5.0
1002	4	20.0

Query

```
select * from order_contents
    where order_num >= 1001 and kg_bought > 3;
```

Conditionals can be combined with the `and` and `or` operators.

SQL Queries (9)

Query Result						
order_num	product_id	kg_bought	order_num	order_date	customer_lastname	customer_firstname
1000	3	0.6	1000	2017-06-16	Bloom	Leopold
1000	3	0.6	1001	2018-07-05	Samsa	Gregor
1000	3	0.6	1002	2018-01-06	Bird	Bill
1000	1	10.0	1000	2017-06-16	Bloom	Leopold
1000	1	10.0	1001	2018-07-05	Samsa	Gregor
1000	1	10.0	1002	2018-01-06	Bird	Bill
1001	1	2.5	1000	2017-06-16	Bloom	Leopold
1001	1	2.5	1001	2018-07-05	Samsa	Gregor
1001	1	2.5	1002	2018-01-06	Bird	Bill
1002	2	5.0	1000	2017-06-16	Bloom	Leopold
1002	2	5.0	1001	2018-07-05	Samsa	Gregor
1002	2	5.0	1002	2018-01-06	Bird	Bill
1002	4	20.0	1000	2017-06-16	Bloom	Leopold
1002	4	20.0	1001	2018-07-05	Samsa	Gregor
1002	4	20.0	1002	2018-01-06	Bird	Bill

```
select * from order_contents, orders;
```

Query

Multiple table names can be specified in the query, but without extra filtering, the results are not very useful.

SQL Queries (10)

Query Result						
order_num	product_id	kg_bought	order_num	order_date	customer_lastname	customer_firstname
1000	3	0.6	1000	2017-06-16	Bloom	Leopold
1000	3	0.6	1001	2018-07-05	Samsa	Gregor
1000	3	0.6	1002	2018-01-06	Bird	Bill
1000	1	10.0	1000	2017-06-16	Bloom	Leopold
1000	1	10.0	1001	2018-07-05	Samsa	Gregor
1000	1	10.0	1002	2018-01-06	Bird	Bill
1001	1	2.5	1000	2017-06-16	Bloom	Leopold
1001	1	2.5	1001	2018-07-05	Samsa	Gregor
1001	1	2.5	1002	2018-01-06	Bird	Bill
1002	2	5.0	1000	2017-06-16	Bloom	Leopold
1002	2	5.0	1001	2018-07-05	Samsa	Gregor
1002	2	5.0	1002	2018-01-06	Bird	Bill
1002	4	20.0	1000	2017-06-16	Bloom	Leopold
1002	4	20.0	1001	2018-07-05	Samsa	Gregor
1002	4	20.0	1002	2018-01-06	Bird	Bill

```
select * from order_contents, orders;
```

Query

Normally, selecting from multiple tables results in an output row for each possible combination of input rows (notice that there are two `order_num` columns, with all possible pairs of order numbers).

SQL Queries (11)

Query Result						
order_num	product_id	kg_bought	order_num	order_date	customer_lastname	customer_firstname
1000	3	0.6	1000	2017-06-16	Bloom	Leopold
1000	1	10.0	1000	2017-06-16	Bloom	Leopold
1001	1	2.5	1001	2018-07-05	Samsa	Gregor
1002	2	5.0	1002	2018-01-06	Bird	Bill
1002	4	20.0	1002	2018-01-06	Bird	Bill

Query

```
select * from order_contents, orders
  where order_contents.order_num = orders.order_num;
```

To synchronize order numbers between two tables, a conditional directive can be added to match the order number between each table (to refer to the order_num column of order_contents, the notation order_contents.order_num can be used).

SQL Queries (12)

Query Result			
order_date	product_id	customer_firstname	customer_lastname
2017-06-16	3	Leopold	Bloom
2017-06-16	1	Leopold	Bloom
2018-07-05	1	Gregor	Samsa
2018-01-06	2	Bill	Bird
2018-01-06	4	Bill	Bird

Query

```
select order_date, product_id,
  customer_firstname, customer_lastname
  from order_contents, orders
  where order_contents.order_num = orders.order_num;
```

The query above uses the order number to join records from the order_contents and orders tables, to print the name and date that each customer bought a given product.

SQL Queries (13)

Query Result			
order_date	product_id	customer_firstname	customer_lastname
2017-06-16	3	Leopold	Bloom
2017-06-16	1	Leopold	Bloom
2018-07-05	1	Gregor	Samsa
2018-01-06	2	Bill	Bird
2018-01-06	4	Bill	Bird

Query

```
select order_date, product_id,
  customer_firstname, customer_lastname
  from order_contents, orders
  where order_contents.order_num = orders.order_num;
```

Observation: The fact that both tables have a column called order_num implies that there is some natural correspondence between the data in each table.

SQL Queries (14)

Query Result			
order_date	product_id	customer_firstname	customer_lastname
2017-06-16	3	Leopold	Bloom
2017-06-16	1	Leopold	Bloom
2018-07-05	1	Gregor	Samsa
2018-01-06	2	Bill	Bird
2018-01-06	4	Bill	Bird

Query

```
select order_date, product_id,
    customer_firstname, customer_lastname
    from order_contents natural join orders;
```

SQL defines an operation called `natural join` which takes two tables and creates a new composite relation based on joining the tables by any common columns.

SQL Queries (15)

Query Result			
order_date	product_id	customer_firstname	customer_lastname
2017-06-16	3	Leopold	Bloom
2017-06-16	1	Leopold	Bloom
2018-07-05	1	Gregor	Samsa
2018-01-06	2	Bill	Bird
2018-01-06	4	Bill	Bird

Query

```
select order_date, product_id,
    customer_firstname, customer_lastname
    from order_contents natural join orders;
```

Natural joins are functionally equivalent to manually specifying that the column values must be equal.

SQL Queries (16)

Query Result			
order_date	product_id	customer_firstname	customer_lastname
2017-06-16	3	Leopold	Bloom
2017-06-16	1	Leopold	Bloom
2018-07-05	1	Gregor	Samsa
2018-01-06	2	Bill	Bird
2018-01-06	4	Bill	Bird

Query

```
select order_date, product_id,
    customer_firstname, customer_lastname
    from order_contents natural join orders;
```

The result of a natural join behaves exactly like a regular table for query purposes.

SQL Queries (17)

Query Result			
order_date	product_id	customer_firstname	customer_lastname
2017-06-16	3	Leopold	Bloom
2017-06-16	1	Leopold	Bloom
2018-07-05	1	Gregor	Samsa
2018-01-06	2	Bill	Bird
2018-01-06	4	Bill	Bird

Query

```
select order_date, product_id,
    customer_firstname, customer_lastname
    from order_contents natural join orders;
```

Question: Can the query above be refined to print the product name instead of numerical ID? (The name of each product is stored in the `products` table.)

SQL Queries (18)

Query Result			
order_date	name	customer_firstname	customer_lastname
2017-06-16	Lime	Leopold	Bloom
2017-06-16	Apple	Leopold	Bloom
2018-07-05	Apple	Gregor	Samsa
2018-01-06	Pear	Bill	Bird
2018-01-06	Raspberry	Bill	Bird

Query

```
select order_date, name,
  customer_firstname, customer_lastname
  from ( (order_contents natural join orders)
        natural join products);
```

Using another natural join (the parentheses around the joins are optional), the previous relation can be joined against the `products` table.

SQL Queries (19)

Query Result				
order_num	name	kg_bought	price_kg	item_price
1000	Lime	0.6	5.0	3.0
1000	Apple	10.0	3.5	35.0
1001	Apple	2.5	3.5	8.75
1002	Pear	5.0	4.0	20.0
1002	Raspberry	20.0	10.0	200.0

Query

```
select order_num, name,kg_bought,price_kg,
  kg_bought*price_kg as item_price
  from order_contents natural join products;
```

New columns can be created with the existing data using arithmetic expressions (or the results of functions).

SQL Queries (20)

Query Result				
order_num	name	kg_bought	price_kg	item_price
1000	Lime	0.6	5.0	3.0
1000	Apple	10.0	3.5	35.0
1001	Apple	2.5	3.5	8.75
1002	Pear	5.0	4.0	20.0
1002	Raspberry	20.0	10.0	200.0

Query

```
select order_num, name,kg_bought,price_kg,
  kg_bought*price_kg as item_price
  from order_contents natural join products;
```

In the example above, the total cost of each item in each order is computed as the product of the weight purchased and the price per kilogram.

Query Result				
order_num	name	kg_bought	price_kg	item_price
1000	Lime	0.6	5.0	3.0
1000	Apple	10.0	3.5	35.0
1001	Apple	2.5	3.5	8.75
1002	Pear	5.0	4.0	20.0
1002	Raspberry	20.0	10.0	200.0

Query

```
select order_num, name,kg_bought,price_kg,
  kg_bought*price_kg as item_price
  from order_contents natural join products;
```

The `select` statement is declarative because it specifies only the expected result, not the mechanism by which the data should be retrieved.

SQL Queries (22)

Query Result				
order_num	name	kg_bought	price_kg	item_price
1000	Lime	0.6	5.0	3.0
1000	Apple	10.0	3.5	35.0
1001	Apple	2.5	3.5	8.75
1002	Pear	5.0	4.0	20.0
1002	Raspberry	20.0	10.0	200.0

Query

```
select order_num, name,kg_bought,price_kg,
  kg_bought*price_kg as item_price
  from order_contents natural join products;
```

Inside the DBMS, different algorithms may be used to store and retrieve data to optimize query times.

SQL Queries (23)

Query Result				
order_num	name	kg_bought	price_kg	item_price
1000	Lime	0.6	5.0	3.0
1000	Apple	10.0	3.5	35.0
1001	Apple	2.5	3.5	8.75
1002	Pear	5.0	4.0	20.0
1002	Raspberry	20.0	10.0	200.0

Query

```
select order_num, name,kg_bought,price_kg,
  kg_bought*price_kg as item_price
  from order_contents natural join products;
```

To represent the data, data structures based on binary trees and hash tables are often used to improve retrieval times.

SQL Queries (24)

Query Result				
order_num	name	kg_bought	price_kg	item_price
1000	Lime	0.6	5.0	3.0
1000	Apple	10.0	3.5	35.0
1001	Apple	2.5	3.5	8.75
1002	Pear	5.0	4.0	20.0
1002	Raspberry	20.0	10.0	200.0

Query

```
select order_num, name,kg_bought,price_kg,
  kg_bought*price_kg as item_price
  from order_contents natural join products;
```

Before a query is evaluated, a query optimization algorithm may be used to optimize the retrieval speed for the particular data structures used.

Aggregation (1)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

Question: Is it possible to determine the total cost of a given order (such as order 1000)?

Aggregation (2)

Table products		
product_id	name	price_kg
1	Apple	3.5
2	Pear	4.0
3	Lime	5.0
4	Raspberry	10.0

Table order_contents		
order_num	product_id	kg_bought
1000	3	0.6
1000	1	10.0
1001	1	2.5
1002	2	5.0
1002	4	20.0

Table orders			
order_num	order_date	customer_lastname	customer_firstname
1000	2017-06-16	Bloom	Leopold
1001	2018-07-05	Samsa	Gregor
1002	2018-01-06	Bird	Bill

Although the database does not store the total order cost anywhere, all of the necessary data to compute the cost is present. **Aggregation** refers to the process of combining information from multiple records (such as adding up the cost of each item in the order).

Aggregation (3)

Query Result				
order_num	name	kg_bought	price_kg	item_price
1000	Apple	10.0	3.5	35.0
1000	Lime	0.6	5.0	3.0
1001	Apple	2.5	3.5	8.75
1002	Pear	5.0	4.0	20.0
1002	Raspberry	20.0	10.0	200.0

Query

```
select order_num, name,kg_bought,price_kg,
    kg_bought*price_kg as item_price
    from order_contents natural join products;
```

To compute the total cost of each order, we want to sum the item_price column within each of the highlighted groups above (corresponding to each order number).

Aggregation (4)

Query Result				
order_num	name	kg_bought	price_kg	total_price
1000	Lime	0.6	5.0	38.0
1001	Apple	2.5	3.5	8.75
1002	Raspberry	20.0	10.0	220.0

Query

```
select order_num, name,kg_bought,price_kg,
    sum(kg_bought*price_kg) as total_price
    from order_contents natural join products
    group by order_num;
```

The `group by` directive specifies which column to use for forming groups. In a query where `group by` is used, the `sum` aggregation function can be used to add up the values of a column for every row in each group.

Aggregation (5)

Query Result				
order_num	name	kg_bought	price_kg	total_price
1000	Lime	0.6	5.0	38.0
1001	Apple	2.5	3.5	8.75
1002	Raspberry	20.0	10.0	220.0

Query

```
select order_num, name,kg_bought,price_kg,
    sum(kg_bought*price_kg) as total_price
    from order_contents natural join products
    group by order_num;
```

Exercise: Write a query which prints only those orders whose total price is at least 50 dollars.

Aggregation (6)

Query Result				
order_num	name	kg_bought	price_kg	total_price
1002	Raspberry	20.0	10.0	220.0

Query

```
select order_num, name,kg_bought,price_kg,
    sum(kg_bought*price_kg) as total_price
    from order_contents natural join products
    group by order_num
    having total_price > 50;
```

The `having` directive is similar to `where`, but for aggregated data. However, there is also a method to compute the result above without the `having` directive.

Aggregation (7)

Query Result				
order_num	name	kg_bought	price_kg	total_price
1002	Raspberry	20.0	10.0	220.0

Query

```
select * from
    (select order_num, name,kg_bought,price_kg,
        sum(kg_bought*price_kg) as total_price
        from order_contents natural join products
        group by order_num)
    where total_price > 50;
```

The result of a `select` query is a relation (even though it does not directly correspond to a table stored on disk). SQL allows the result of `select` to be used as a table for another `select` statement.

Aggregation (8)

Query Result				
order_num	name	kg_bought	price_kg	total_price
1002	Raspberry	20.0	10.0	220.0

Query

```
select * from
    (select order_num, name,kg_bought,price_kg,
        sum(kg_bought*price_kg) as total_price
        from order_contents natural join products
        group by order_num)
    where total_price > 50;
```

Exercise: Write an SQL query to print the number of different products in each order.

Aggregation (9)

Query Result	
order_num	count(product_id)
1000	2
1001	1
1002	2

Query

```
select order_num, count(product_id) from
    order_contents group by order_num;
```

Grouping the table order_contents by order number and applying the count aggregation function yields the desired result.

Aggregation (10)

Query Result	
order_num	count(product_id)
1000	2
1001	1
1002	2

Query

```
select order_num, count(product_id) from
    order_contents group by order_num;
```

Exercise: Print the names of all customers who placed orders containing two or more items, sorted by the customer's last name.

Query Result				
order_num	count(product_id)	order_date	customer_lastname	customer_firstname
1000	2	2017-06-16	Bloom	Leopold
1001	1	2018-07-05	Samsa	Gregor
1002	2	2018-01-06	Bird	Bill

Query

```
select * from
    (select order_num,
        count(product_id)
        from order_contents
        group by order_num)
    natural join orders;
```

This task can be approached in several ways. One way is to use previous query as a nested `select` statement and join it with the table containing customer names.

Aggregation (12)

Query Result				
order_num	item_count	order_date	customer_lastname	customer_firstname
1000	2	2017-06-16	Bloom	Leopold
1001	1	2018-07-05	Samsa	Gregor
1002	2	2018-01-06	Bird	Bill

Query

```
select * from
    (select order_num,
        count(product_id) as item_count
        from order_contents
        group by order_num)
    natural join orders;
```

The notation as `item_count` renames the aggregated column, allowing it to be conveniently used elsewhere in the query.

Aggregation (13)

Query Result				
order_num	item_count	order_date	customer_lastname	customer_firstname
1000	2	2017-06-16	Bloom	Leopold
1002	2	2018-01-06	Bird	Bill

Query

```
select * from
    (select order_num,
        count(product_id) as item_count
        from order_contents
        group by order_num)
    natural join orders
    where item_count >= 2;
```

The query can be filtered by `item_count` to yield only orders with two or more items.

Aggregation (14)

Query Result	
customer_firstname	customer_lastname
Bill	Bird
Leopold	Bloom

```
select customer_firstname, customer_lastname from
    (select order_num,
        count(product_id) as item_count
        from order_contents
        group by order_num)
    natural join orders
    where item_count >= 2
    order by customer_lastname;
```

Query

Finally, only the columns containing the name can be selected and the `order by` directive can be used to sort the results.

Aggregation (15)

Query Result	
customer_firstname	customer_lastname
Bill	Bird
Leopold	Bloom

```
select customer_firstname, customer_lastname from
    orders natural join order_contents
    group by order_num
    having count(product_id) > 1
    order by customer_lastname;
```

Query

The same result can also be obtained without nested queries using the `having` directive.