

# Databases

## Part 1/2

# The Relational Data Model

**Kai Brännler**  
**CAS Applied Data Science**  
**University of Bern**

Material adapted from: Silberschatz, Korth, Sudarshan: Database System Concepts. 6th Edition.

# Database Management System

## ■ A DBMS consists of

- A collection of interrelated data
- A set of programs to access the data
- An environment that is both *convenient* and *efficient* to use

python: high-level language (convenient but not efficient)

C: low-level language (not convenient but efficient)

--> Database should combine those

## ■ Classic Database Applications:

- Banks, Airlines, Universities, Shops, Manufacturers, Human resources

## ■ Our running Example: A University Database.

- Some routine tasks:
  - ▶ add new student, register student for course, list course participants, assign grades, compute grade point averages, generate transcripts
- Can you think of a data science task?

# Drawbacks of File Systems

In the early days, database applications were built directly on top of file systems. But that leads to many problems.

- Data redundancy and inconsistency
  - Duplication of information in different files
  - Example: data of a student with double major is stored by both departments
- Difficulty in accessing data
  - Need to write a new program for each unexpected new task
  - Example: There is an application program to generate a list of all students. Suddenly, there is a new requirement to generate a list of all students with certain grade.

# Drawbacks of File Systems

## ■ Data isolation

- Multiple files can have multiple file formats
- It becomes harder to write new applications to access the data
- Example: First, data was stored in CSV for interoperability, but later XML became popular, so new data was stored in XML.

## ■ Integrity problems

- Integrity constraints become “buried” in program code: they become hard to understand and hard to change
- Example: There are only two semesters in a year: “Fall” and “Spring”. It is hard to identify all the places in the application code that ensure that.

# Drawbacks of File Systems

## ■ Updates are not atomic

- System failures may leave data in an inconsistent state
- Example: A transfer of funds consists of two operations: subtract from one account, add to the other. The system might fail after the first operation.

we want that the whole transfer is atomic in itself (so that it either works fully or not at all, and not partly)

## ■ Problems with concurrent access

- Concurrent access can lead to inconsistencies
- Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time

at the same time --> both machines give back that now there should be 50 left --> 50 are left

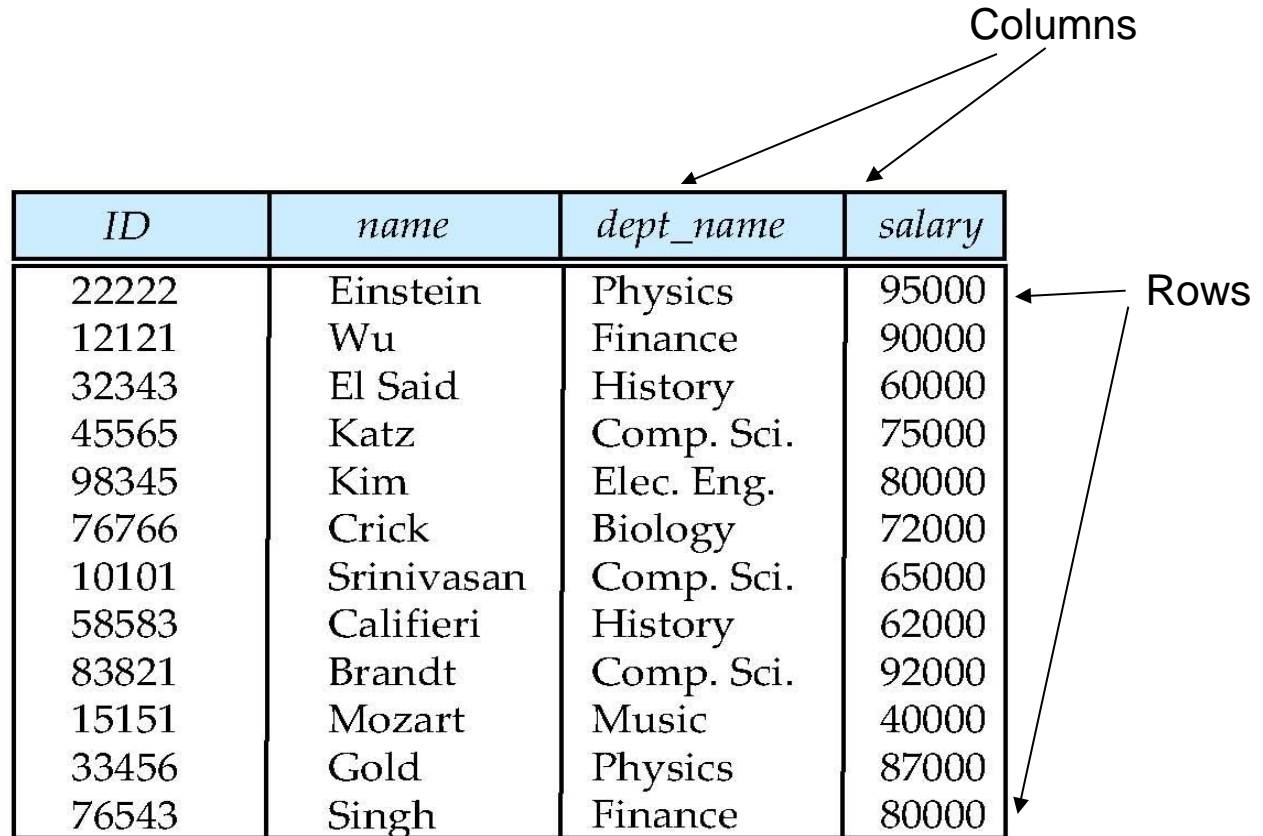
## ■ Security problems

- It is hard to provide a user with access to some, but not all, data
- Example: A schedule planner needs to know the department of an instructor but is not allowed to see salary

**Database systems were developed to solve these problems.**

# Relational Model

- Example of tabular data in the relational model



The diagram illustrates a table with four columns and twelve rows. Two arrows labeled 'Columns' point to the top row, specifically to the 'dept\_name' and 'salary' columns. Two arrows labeled 'Rows' point to the right side of the table, specifically to the first and last rows.

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

# A Sample Relational Database

multiple files that relate to each other

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

# Schema and Instance

- They are similar to type and value of a variable in programming languages
- **Schema** – the logical structure of the database
  - Example:
    - ▶ instructor(ID, name, dept\_name, salary)
    - ▶ department(dept\_name, building, budget)
    - ▶ ... and the meaning of all those terms
- **Instance** – the actual content of the database at a particular point in time
  - Example: the tables we have seen before the actual table at one timepoint



# SQL

- We usually use **SQL** (Structured Query Language) to access a database
- SQL has two distinct parts:
  - The schema is modified by the **Data Definition Language** (DDL)
  - The instance is modified by the **Data Manipulation Language** (DML)

# SQL: Data Definition Language

- Defining the database **schema** and implementation details

Example:        **create table** *instructor* (  
                              *ID*              **char**(5),  
                              *name*          **varchar**(20),  
                              *dept\_name* **varchar**(20),  
                              *salary*      **numeric**(8,2))

- The DDL can also specify **integrity constraints**:
  - Primary key constraint
    - ▶ Example: An ID uniquely identifies an instructor
  - Foreign key constraint    constraints in relation to another file
    - ▶ Example: The *dept\_name* of an *instructor* must exist in the *department* relation

# SQL: Data Manipulation Language

- Example: Find the name of the instructor with ID 22222

```
select   name  
from     instructor  
where    ID = '22222'           -> "Einstein"
```

- Example: Find the building of the instructor “Einstein”

```
select building  
from   instructor, department  
where instructor.dept_name = department.dept_name and  
       name = 'Einstein'
```

Result will be a (small) table (e.g. a table with 1 column and 1 row [Building = Watson])

# Database Schema Design

- Is there any problem with this schema?

Redundancy: relations between department and building, and department and budget are redundant

<i>ID</i>	<i>name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

with redundancy we would need to update relations individually for each row

# Database Schema Design

- The previous database schema is bad:
  - Repetition of information:
    - ▶ the budget of a certain department is stored redundantly for each instructors in the department
  - Difficulty of storing information:
    - ▶ how to store the budget of a department without instructors?
- How would you improve the schema?

-> split the tables

# Database Management Software

- Relational Database Software
  - Commercial
    - ▶ Oracle, IBM DB2, MS SQL Server
  - Open Source
    - ▶ Postgres, MySQL
  
- What's the most widely deployed database software?

SQLite

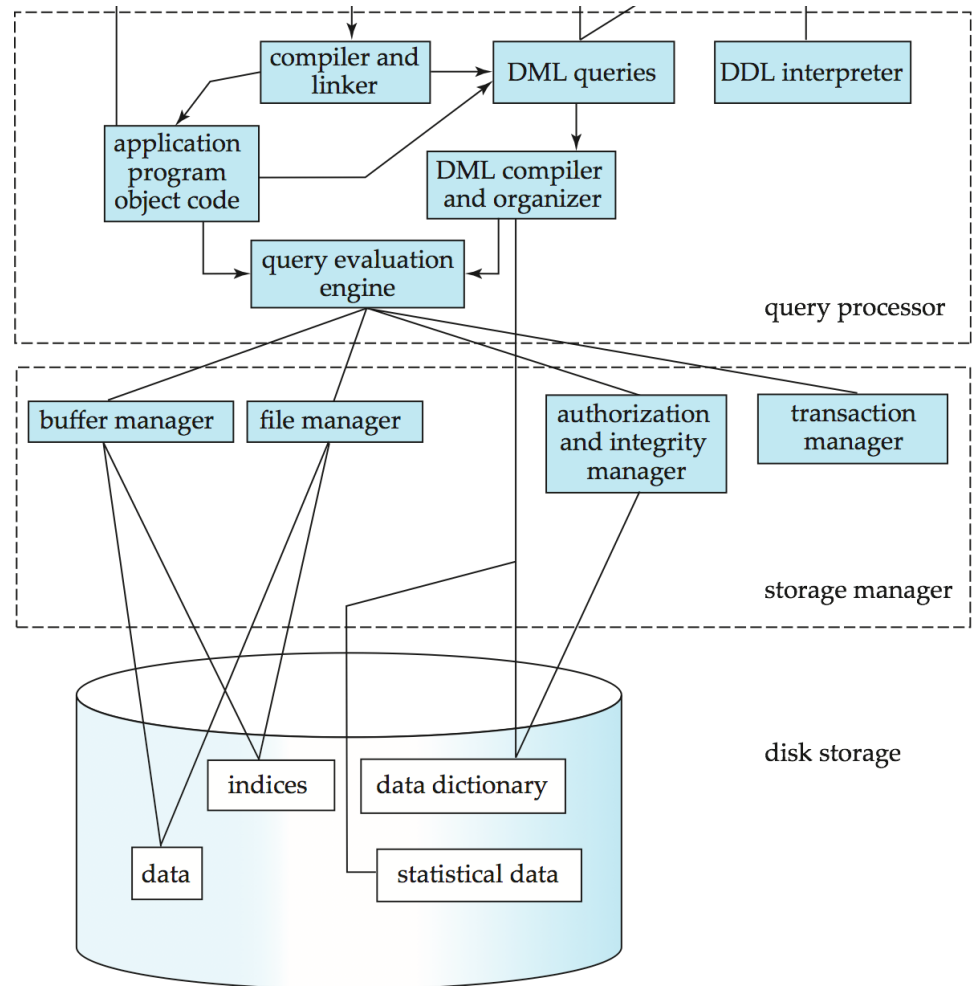
# SQLite

- SQLite is an open source SQL database library
- Less than 500KB
- "a replacement for fopen()"
- The most widely deployed database, used in:
  - Android, iPhone
  - Windows 10, MacOS
  - Firefox, Chrome, Safari
  - Skype, Dropbox, Adobe Reader
  - etc.

# Structure of a DBMS

A database system consists of two main components:

- The **Query Processor** translates queries into an efficient sequence of operations at the physical level.
- The **Storage manager** stores data and executes operations at the physical level.





# The *course* Relation

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

base info about the course (than can happen more than once / at different locations)

# The *section* Relation

<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>building</i>	<i>room_number</i>	<i>time_slot_id</i>
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

*sec\_id*: necessary when same course twice in same semester and year

the actual runs of courses

# The *teaches* Relation

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

# Primary Keys

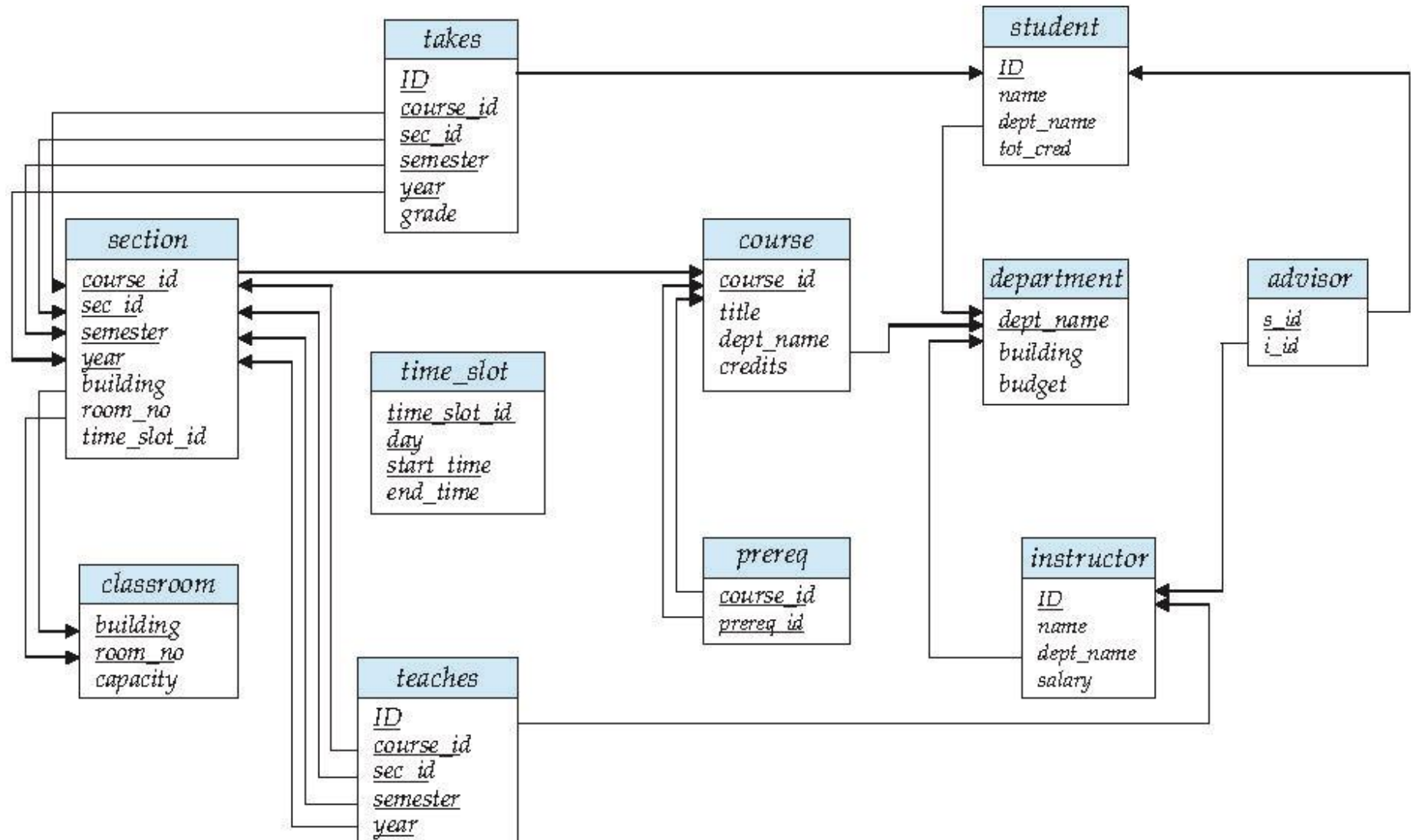
- For each table, the database designer chooses a primary means of identifying a tuple: the **primary key**.
- Notation: underline the attributes which form the primary key.
  - instructor (ID, name, dept\_name, salary)
  - section (course\_id, sec\_id, semester, year, building, room\_number, time\_slot\_id)
  - teaches (ID, course\_id, sec\_id, semester, year)
- The **primary key constraint** ensures that two <sup>rows</sup> tuples can not have the same values for the primary key attributes
  - For example, the database will reject adding a new instructor with an existing ID

# Foreign Keys

- A **foreign key** is a set of attributes in one relation which is the primary key of another relation
  - Example: the dept\_name attribute in the instructor relation is the primary key of the department relation
- Notation:
  - instructor (ID, name, dept\_name, salary)
  - dept\_name → department<sub>table</sub>
  - instructor is called the **referencing** relation
  - department is called the **referenced** relation
- The **foreign key constraint** says that attribute values of the foreign key in the referencing relation have to occur in the referenced relation
  - The database will reject adding an instructor with a department which is not in the department relation

# Schema Diagram for University Database

relations and references between tables



# Other Data Models

- We only talk about the **relational data model**.
  - It essentially says “data is a set of tables”.
  - Most data fits well into tables – how about your data?
  
- There are also other, non-relational, data models data that doesn't naturally fit into tables
  - Semi-structured data e.g. a bachelor thesis  
"data is a set of tree-like documents" (like JSON or XML).  
Example: MongoDB
  - Graph data, or object-oriented data e.g. SBB connections network  
"data is pieces of data which include pointers to other pieces of data"  
Example: Neo4J
  - Key-value stores, "data is tables with two columns"  
Example: Redis when performance optimization is important