# CM0133 Internet Computing

Databases & SQL



## Objectives

- What is a database?
- Definitions
- Data Modelling
- The Relational Model
- SQL

**Note:** Databases, Data Modelling Information Systems are covered in other lectures during your studies. This lecture only covers so much to enable you to use databases for the implementation with web sites. Some of what I introduce is specific to MySQL.



## Database

- A database is a collection of interrelated data of different data types.
- Data that answers a user's query is information.
- Databases are the core of information systems. A system that provides answers to users' questions and needs based on data is an information system.
- A website powered by a database can be the interface of an information system:
  - A web site that is capturing registered users
  - A client tracking application for social service organisations
  - A medical record system for a health care facility
  - Your personal address book in your web-mail client
  - A collection of word processed documents (e.g. google docs)
  - A system that issues airline reservations
  - etc.

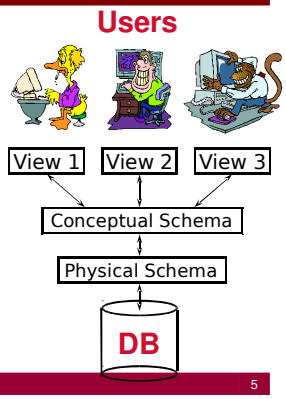
# Data Models

- Data models are notations for describing data.
- They are meta concepts defining the contents, structure, and meaning of data. This meta data which describe data populating a database must be distinguished from the data itself.
- Data models can be classified into:
  - Conceptual data models
  - Logical data models
  - Physical data models



# Levels of Abstraction

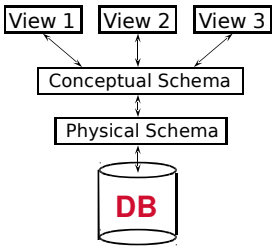
- Views describe how users see the data.
- Conceptual schema defines logical structure
- Physical schema describes the files and indexes used.
- (sometimes called the ANSI/SPARC model)





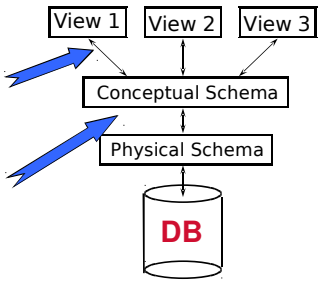

# Example: University Database

- Conceptual schema:
  - *Students*(sid: *string*, name: *string*, login: *string*, age: *integer*, gpa:*real*)
  - *Courses*(cid: *string*, cname:*string*, credits:*integer*)
  - *Enrolled*(sid:*string*, cid:*string*, grade:*string*)
- External Schema (View):
  - *Course_info*(cid:*string*,enrollment:*integer*)
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.



# Data Independence

- Applications insulated from how data is structured and stored.
- Logical data independence: Protection from changes in *logical* structure of data.
- Physical data independence: Protection from changes in *physical* structure of data.
- Q: Why are these particularly important for DBMS?






# Data Models

- **Conceptual Models** provide easy to perceive high level concepts. They are used in early design stages of your information system, e.g. the Entity Relationship Model, independently from any existing database management system.
- **Physical data models** provide low-level concepts to describe how data is stored and accessed in the computer.
- **Logical data models** bridge the gap between conceptual and physical models and are often referred to as implementation data models:
  - Hierarchical data model
  - Network data model
  - **Relational data model**
  - Object / Relational model
  - Object oriented model
  - and others



# Relational Model

- The Relational Model was introduced by E. Codd in 1970 and has been implemented in many commercial data base management systems.
- Represents data in a database as a collection of relations. A relation can be thought of as a table of values representing a set of similar real world objects and their relationships.
- The rows of a table, called tuples, define real world objects or relationships between real world objects
- The columns of a table represent attributes and attribute values, respectively.

## Example & Key

- Table city1

| ZIP | Name |
|---|---|
| 5020 | Vienna |
| 3040 | Paris |
| 6060 | Rome |

- The **key** of a relation schema is a minimal set of attributes such that no two different tuples in a relation agree on them.
- Both attributes are key candidates in the table city1.
- In this example one of the two can be selected to be the **primary key**.
  - Which properties should the primary key have ?



## Foreign Key

- A **foreign key** of a relation schema is a set of attributes that is a primary key in another relation schema of the same database.

| ZIP | Name |
|---|---|
| 5020 | Vienna |
| 3040 | Paris |
| 6060 | Rome |

Primary Key : ZIP
Foreign Key : Name

| Name | Population |
|---|---|
| Vienna | 2,2 |
| Paris | 11,2 |
| Rome | 3,7 |

Primary Key: Name

Question: Are these good keys?



## Normalization

- **Normalization** is the process applied during database design to test and improve the quality of database schemes.
- The relation schemes are tested whether or not they belong to certain **normal forms**.
- Relation schemes which do not satisfy some normal form are decomposed into smaller relations to avoid redundancies and anomalies caused by redundancies.
- **Decomposition** helps to get data chunks that represent one real world object only and thus support easy maintenance and extension of databases.

# First Normal Form (1. NF)

- A relational schema is in first normal form (1. NF) if the domains of its attributes consist of atomic values.
- Example: Stock **is not** in 1. NF because the attribute warehouse is set valued.

**stock**

| part | warehouse |
|---|---|
| 101 | {1,3} |
| 102 | {1,2,4} |
| 103 | {4} |



# Redundancy

- Now the table is in 1.NF. But first normal form does not help to avoid anomalies due to redundancy in relational schemes as the following example shows:

Keys are underlined in this examples.

**stock**

| part | warehouse | quantity | address |
|---|---|---|---|
| 101 | 1 | 25 | St. Mary Street |
| 102 | 3 | 410 | Welfield Rd. |
| 102 | 1 | 300 | St. Mary Street |
| 112 | 4 | 10 | City Rd. |

redundant address



# Update Anomaly

- Redundancy may cause an anomaly on address modification (update anomaly). If warehouse 1 moves to a new address the address must be changed for all parts located at warehouse 1. :

**stock**

| part | warehouse | quantity | address |
|---|---|---|---|
| 101 | 1 | 25 | Newport Rd. |
| 102 | 3 | 410 | Welfield Rd. |
| 102 | 1 | 300 | St. Mary Street |
| 112 | 4 | 10 | City Rd. |

Update anomaly

## Insertion Anomaly

- Similarily it is not possible to store the address of a new warehouse which has no parts in it (insertion anomaly), unless using null values.

| part | warehouse | quantity | address |
|---|---|---|---|
| 101 | 1 | 25 | Newport Rd. |
| ~~102~~ | ~~3~~ | ~~410~~ | ~~Welfield Rd.~~ |
| 102 | 1 | 300 | Newport Rd. |
| 112 | 4 | 10 | City Rd. |



## Second Normal Form (2. NF)

- A relation schema is in second normal form (2. NF) if no non-key attribute is dependent on part of the primary key.
- Given the previous example we can see that non-key attribute address is dependent on attribute warehouse.
- To obey 2.NF we have to split the relation stock into two relations.

**stock**

| part | warehouse | qty |
|---|---|---|
| 101 | 1 | 25 |
| 102 | 3 | 410 |
| 102 | 1 | 300 |
| 112 | 4 | 10 |

**warehouse**

| warehouse | address |
|---|---|
| 1 | Newport Rd. |
| 3 | Welfield Rd. |
| 4 | City Rd. |



## Example

- If warehouse 1 moves now to a new address (High Street) it's address entry remains consistent, as it is not stored redundant.
- If we remove part 112 the address of warehouse 4 is still available. We have overcome the previously mentioned anomalies (update & insertion).

**stock**

| part | warehouse | qty |
|---|---|---|
| 101 | 1 | 25 |
| 102 | 3 | 410 |
| 102 | 1 | 300 |
| ~~112~~ | ~~4~~ | ~~10~~ |

**warehouse**

| warehouse | address |
|---|---|
| 1 | High Street |
| 3 | Welfield Rd. |
| 4 | City Rd. |

## Further Anomalies

- Although in 2.NF (no non-key attribute is dependent on part of the primary key) the following example does not avoid all anomalies:

**employee**

| no. | dept. | building |
|---|---|---|
| 1215 | Accounting | A12 |
| 2410 | Sales | A14 |
| 2412 | PR | B8 |
| 809 | Accounting | A12 |

There is a functional dependency in this relation:

$$FD = \{ \text{no.} \rightarrow \text{dept}, \text{dept} \rightarrow \text{building} \}$$



## Update & Deletion Anomaly

- Modifying an address, e.g. moving department Accounting from building A12 to building A7, may cause an update anomaly.
- If a department has no employees for a short time, e.g. employee no. 2412 is fired, the information which building locates the department is lost (deletion anomaly), unless using null values .

**employee**

| no. | dept. | building |
|---|---|---|
| 1215 | Accounting | A7 |
| 2410 | Sales | A14 |
| ~~2412~~ | ~~PR~~ | ~~B8~~ |
| 809 | Accounting | A12 |

Deletion anomaly →

Update anomaly



## Third Normal Form (3. NF)

- A relation scheme is in third normal form (3. NF) if no non-key attribute is transitively dependent on the primary key.
- Given the set of functional dependencies over relation employee (see before) we see that non-key attribute building is transitively dependent on the primary key (attribute no.). To obey 3.NF relation employee is split into two relations. Then each real world object is represented by exactly one tuple of one relation schema.

**employee**

| no. | dept |
|---|---|
| 1215 | Accounting |
| 2410 | Sales |
| 2412 | PR |
| 809 | Accounting |

**dept**

| dept | building |
|---|---|
| Accounting | A12 |
| PR | B8 |
| Sales | A14 |

# Summary Normalization

- Normalization helps to avoid anomalies. However do not split relations arbitrarily. Using a database with a website is not difficult, but requires some careful design considerations if you want to avoid consistency problems and also performance bottlenecks.
- This table is in 3.NF and the previous anomalies are removed.

**employee**

| no. | dept |
|---|---|
| 1215 | Accounting |
| 2410 | Sales |
| ~~2412~~ | ~~PR~~ |
| 809 | Accounting |

Although deleted PR has still an address.

**dept**

| dept | building |
|---|---|
| Accounting | A7 |
| PR | B8 |
| Sales | A14 |

All accounting employees are now at the modified address



# Database Management Systems (DBMS)

- A database management system (DBMS) supports reliable and efficient sharing of large sets of data among several users. In particular, a DBMS provides the following features:
  - persistency
  - efficient storage management
  - recovery
  - concurrency control
  - ad-hoc queries (e.g. SQL)
  - data security
- A DBMS allows to insert, retrieve and maintain data.

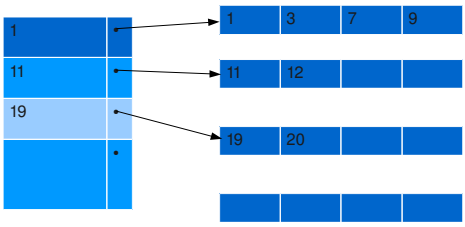


# Features of DBMS

- **Persistent storage** of data means that that data survive the execution of programs.
- **Efficient Storage Management:** Databases support efficient storage of large sets of data that do not fit entirely into main memory. Data is moved from a secondary storage e.g. disk to main memory using pages and buffers. There is a variety of buffering techniques that can not be covered in this course.

  Indexing techniqes are used to retrieve data from the disk. An index I associated to a data file D is an ordered file (a sequence of records) with entries $(k_i,p_i)$ where $k_i$ is the value of the indexing field of a record in D and $p_i$ is the address of the block containing that record.

# Primary Index




A **primary index** is specified on the ordering key field of a data file. The index file contains one entry for each block in the data file (e.g. value of the index field and address of the first record in each block).

Consider a dictionary. It contains a primary index with one index entry for every page ( = block). The index entry is printed on the header of the page and contains the first and last entry given at this page.



# Indices

- Indices provide fast access to our records (e.g. binary search).
- A rule for your web databases: If you do a lot of search on an attribute (column) then use an index! No matter which it will improve your access.
- There are many ways to index data and they will be covered in other lectures. You will come across clustering index, hashing, B* Trees (hierarchical multilevel index).
- MySQL mostly implements a B-Tree index, if you work with memory tables than hashing is used and if you work with spatial data MySQL uses R-Trees.

http://dev.mysql.com/doc/refman/5.0/en/mysql-indexes.html



# Transaction: An Execution of a DB Program

- Key concept is *transaction*, which is an *atomic* sequence of database actions (reads/writes).
- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.
  - Users can specify some simple *integrity constraints* on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data.
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's/developer's responsibility!

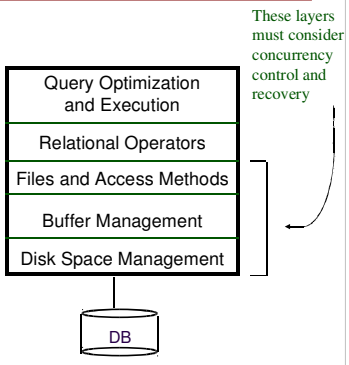
## Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a transaction.
- Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of transactions:
  - Before a change is made to the database, the corresponding log entry is forced to a safe location.
  - After a crash, the effects of partially executed transactions are *undone* using the log. (the change was not applied to database but to the log itself!)



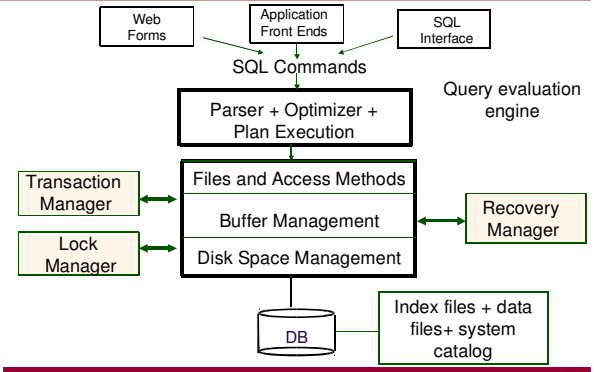
## Structure of a DBMS

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.






## Structure of a DBMS (cont.)



## Database Transactions - Atomicity

- **Atomicity**: Transactions are executed atomically. This means that either none of the actions of a transaction is carried out or all of them are carried out. Special commands are carried out to indicate the start of a transaction (begin transaction), the successful completion of a transaction (commit transaction), and the abort of a transaction (abort transaction).



## Consistency

- Transactions move the database from one **consistent state** to another even if the database is accessed by several users simultaneously, executing several transactions interleaved (or in parallel). The traditional correctness criterion for executing several transactions interleaved is serializability.
- **Serializability** means that the overall effect of several transactions executed interleaved is the same as if these transaction had been executed in some serial order.



## Isolation & Durability

- Transactions are executed in **isolation**. Interim results of a transaction are not visible to other transactions. This means that effects of a transaction are visible to other transactions only after it has been completed successfully.
- **Durability** guarantees that once a transaction has been completed successfully, its effects remain persistent despite possible subsequent failures.

## Question ?

How can you test if a database or information system supports data integrity and consistency?



## Reasons for a DBMS

- Changes to the type and format of data may occur frequently. Data independence is important
- Large amounts of data must be stored and be retrieved efficiently
- Data must be updated reliably. Inconsistent database states due to hardware and software failure are not tolerable.
- Date are accessed by several users simultaneously.
- Unexpected queries should be handled fast.
- Data are very sensitive. Data security is very important.



## Reasons against DBMS

- The amount of data is small.
- The application is very simple, no future changes to data types and data formats are expected.
- Concurrent access to the database is not required.
- The high costs of a data base management system (DBMS) are unjustified (although nowadays there are low cost solutions)
- The application has strict real time requirements and DBMS would be to slow.
- The application is very special and cannot be supported by a standard DBMS efficiently.

## Querying a DBMS

- A DBMS provides a Query Language.
- Query languages allow querying and updating a DMBS in a simple way.
- Most popular DML (Data Manipulation Language) : SQL(Structured Query Language).
- Queries:
  - List the name of student with sid=27373
  - Name and age of students enrolled in CM0133

The following examples are SQL queries for MySQL. There might be a difference with another DBMS. MySQL often conforms with ANSI SQL standard.



## SQL – CREATE TABLE

```
CREATE TABLE 'CM0133'.'students' (
    'uid' BIGINT NOT NULL AUTO_INCREMENT ,
    'firstName' VARCHAR( 100 ) NOT NULL ,
    'surname' VARCHAR( 100 ) NOT NULL ,
    'address' TEXT NULL ,
   PRIMARY KEY ( 'uid' )
   );
```

**MySQL Data Types**



## CREATE TABLE

- Different database implementations support different data types. For our examples we can use integer (BIGINT), characters (VARCHAR (length)), Text, Date and Timestamp.
- NOT NULL indicates a constraint. Data has to be entered for this attribute. In our example key and full name has to be provided but not the address (NULL).
- AUTO_INCREMENT is a non standard SQL convenience function by MySQL that creates unique integers for you by incrementing.

# INSERT

```
INSERT INTO CM0133.students (uid , firstName ,surname , address)
VALUES (
NULL , 'Florian', 'Twaroch', 'Cardiff'
);
```

- With INSERT we populate the created table.
- Note that NULL is entered for the uid, AUTOINCREMENT creates the value for us.

| uid | firstName | surname | address |
|---|---|---|---|
| 1 | Florian | Twaroch | Cardiff |



# UPDATE

```
UPDATE CM0133.students
SET address = 'Zurich'
WHERE students.uid =1;
```

- To change an entry we use the UPDATE command together with a condition.
- We have a number of operators at hand to support that
  - Logical operators: AND, OR, NOT
  - Equivalence op: ==, !=
  - Comparision op: >,<, etc.



# DELETE

```
DELETE * FROM CM0133.students
WHERE students.uid =1;
```

- The DELETE command will delete entries. Again we can use conditions on which tuples we would like to delete.
- Here the user with the unique id 1 is deleted.

## SELECT

The SELECT command allows you to extract tuples from your database, e.g.:

```
SELECT firstName, surname from CM0133.students
WHERE uid > 0 AND uid < 10
ORDER BY surname  DESC;
```

We will look closer at database queries in a tutorial and also how SQL queries interact with PHP in the next lectures.



## Literature

- http://dev.mysql.com/doc/ - MySQL documentation for download, lot of MySQL  Tutorials on the Web.
- Lot of database books in our library http://library.cf.ac.uk
- http://www.unixspace.com/context/databases.html
- Google, Yahoo for Database Tutorials on the Web !