Brief History of SQL

- Developed 1974 at IBM as the language of the relational DBMS System R
- Original name: SEQUEL = Structured English Query Language
- Versions
  - SQL-86 (1986/87): First formalization by ANSI and first ISO standard
  - SQL-89 (1989): Addition of integrity constraints
  - SQL-92 (1992): Also denoted as SQL2, major revision
  - SQL:1999 (1999): Also denoted as SQL3, addition of object-oriented features, triggers, regular expressions
  - SQL:2003: Addition of XML-related features, standardized sequences
  - SQL:2006: Addition of features to import and store XML data in an SQL database
  - SQL:2008: Minor revision
  - SQL:2011: Addition of support for temporal databases
  - SQL:2016: Addition of row pattern matching and JSON support
- Database vendors provide special extensions (packages) and SQL dialects

Components of SQL (I)

- Data definition language (DDL)
  - Commands for creating and changing the data structures for the three levels of a database (external levels, conceptual level, physical level) such as the
    - definition of relation schemas
    - deletion of relation schemas
    - creation of index structures
    - modification of relation schemas
  - Remember: *Relations* (*tables*) are based on a *set* (*list*) model
- Data manipulation language (DML)
  - Update commands for
    - inserting data objects (tuples)
    - modifying tuples
    - deleting tuples
  - Interactive formulation of queries

Components of SQL (II)

- Embedded SQL and dynamic SQL
  - Mechanisms to embed SQL statements into general-purpose programming languages
  - Examples of such host languages are Fortran, C, C++, or Java
- Integrity
  - Commands for specifying integrity constraints that the data stored in the database must satisfy
  - Attempts to violate these integrity constraints are disallowed and blocked
- View definition
  - Commands for defining views
  - Interesting aspect: Views are defined by an SQL query

Components of SQL (III)

- Transaction control
  - Explicit commands for specifying the beginning and ending of transactions
  - Transactions are usually started and ended in the background
  - Explicit aborting a transaction is also possible
- Authorization
  - Commands for specifying access rights to relations and views
  - Different user groups usually obtain different access rights

SQL Data Types (I)

- SQL data types are part of the SQL DDL, built-in types, and serve as domains, that is, attribute data types

- **char**(*n*), **character**(*n*)
  - Character strings of fixed length *n*
  - Length is user-specified
  - All character strings have the same length, strings are padded with blank characters to the right if needed

- **varchar**(*n*) , **char varying**(*n*), **character varying**(*n*)
  - Character strings of variable maximum length *n*
  - Length is user-specified
  - Less than *n* characters are stored in shorter strings

- **int**, **integer**
  - Data type for a finite subset of the integers
  - Machine-dependent (e.g., 4 bytes)

SQL Data Types (II)

- **smallint**
  - Small integers
  - Machine-dependent subset of the integer type (e.g., 2 bytes)
- **numeric**(*p*, *d*), **decimal**(*p*, *d*), **dec**(*p*, *d*)
  - Fixed-point numbers with user specified precision
  - Exact decimal numbers
  - *p* = total number of digits (plus a sign), *d* = number of the *p* digits to the right of the decimal point
  - Example: **numeric**(3,1) allows 87.3 to be stored exactly, this is not the case for 765.4 or 0.19
- **real**
  - Floating-point numbers with machine-dependent precision
- **double precision**
  - Double-precision floating-point numbers with machine-dependent precision

SQL Data Types (III)

- **boolean**
  - Has the traditional values of *true* and *false*
  - Because of the presence of *null* values, a three-valued logic is used with the additional possible Boolean value *unknown*
  - Any comparison involving the *null* value or an *unknown* truth value returns an *unknown* result
- **date**
  - Calendar date with year (4 digits), month (2 digits), and day (2 digits)
  - Format: YYYY-MM-DD
  - Example: **date** '2015-04-13'
- **time**
  - The time of day, in hours, minutes, and seconds (2 digits each)
  - Format: HH:MM:SS
  - Example: **time** '08:34:15'
  - Local time zone is assumed

SQL Data Types (IV)

- **time with time zone**
  - Time difference to Greenwich Mean Time included
  - Displacement from the standard universal time zone in the range of +13:00 to -12:59 in units of HH:MM
- **timestamp**
  - Merges date and time fields
  - Format: YYYY-MM-DD HH:MM:SS
  - Example: **timestamp** ‘2015-04-13 08:34:15’
  - Optional **with time zone** qualifier
- **interval**
  - Intervals are *relative values* that can increment or decrement an absolute value of the types **date**, **time**, or **timestamp**
  - Two classes of intervals
    - Year-month intervals containing the year and month fields
    - Day-time intervals containing a contiguous selection of day, hours, minute, second

SQL Data Types (V)

- **bit**(*n*)
  - Bit string of fixed length *n* bits
- **bit varying**(*n*)
  - Bit string of varying length with a maximum length of *n* bits
- **blob**
  - Binary large objects
  - Byte sequences of variable length up to a user-defined limit
  - Examples: image **blob**(10KB), movie **blob**(2GB)
  - Maximum length can be specified in kilobits (K), megabits (M), gigabits (G), kilobytes (KB), megabytes (MB), or gigabytes (GB)
  - Used to store complex objects from advanced applications, such as multimedia objects, video clips, spatial objects, high-resolution medical images, satellite images, proteins, enzymes, times series
  - Low-level operations for binary reads and writes only

SQL Data Types (VI)

- **clob**
  - Character large objects
  - Large character strings to store books and large documents
  - Example: textbook **clob**(1 GB)

- Declaration of a domain
  - Command: **create domain**
  - Example: **create domain** SSN_Type **as** **char(9)**
  - Advantages
    - Simple centralized change of a data type for a domain which is used by several attributes in a database schema
    - Domains improve schema readability
  - Disadvantage: No complex declarations possible

Integrity Constraints and Default Values (I)

- In the following: The most important commands of the SQL Data Definition Language (DDL)

- Integrity constraints
  - are conditions that restrict the possible database states
  - ensure the consistency of a database
  - will be discussed in much more detail in a later lecture

- Since SQL allows null values (**null**), an integrity constraint **not null** can be defined, if for a specific attribute a null value is *not* allowed
  - Example 1: To store a student's UFID without storing the student's name does not make much sense
  - Example 2: A lecture code (e.g., CIS 4301) should always be accompanied by the lecture title ("Information and Database Management Systems I")

- It is recommended to specify the **not null** condition explicitly for each primary key, although this holds implicitly already for each primary key

Integrity Constraints and Default Values (II)

- Default values
  - Definition of a default value for an attribute possible by attaching the clause **default** <value> to the attribute definition
  - A default value is inserted into each new tuple if an explicit value for this attribute is not specified
  - If a default clause is not defined, the default value is *null*

- Primary keys
  - The clause **primary key** $(A_1, \ldots, A_n)$ specifies that the attributes $A_1, \ldots, A_n \in \mathscr{R}$ $(n \geq 1)$ form the primary key of the relation R with respect to $\mathscr{R}$
  - The set of selected attributes must be unique and minimal
  - Let $\mathscr{R}(A_1, \ldots, A_n, B_1, \ldots, B_m)$ be a relation schema, $A_1, \ldots, A_n$ be the primary key, $B_1, \ldots, B_m$ be nonprime attributes, and R be a relation with schema $\mathscr{R}$. Then the following holds for all tuples of R:

    $\forall\ s, t \in R, s \neq t : \neg\ ((s.A_1, \ldots, s.A_n) = (t.A_1, \ldots, t.A_n))$

Integrity Constraints and Default Values (III)

- Foreign keys
  - A foreign key value references ("points to") a primary key value in another relation to represent an *m*:1-relationship of an E-R diagram
  - Definition of a foreign key by the **foreign key clause**
  - Foreign keys ensure referential integrity, that is, there are no inconsistent or dangling references
  - A foreign key can or cannot become (part of) the primary key of the relation schema into which it is imported

assistants			
pers-id	name	room	boss
3002	Platon	156	2125
3003	Aristoteles	199	2125
3004	Wittgenstein	101	2126
3005	Rhetikus	130	2127
3006	Newton	120	2127
3007	Spinoza	155	2134

professors			
pers-id	name	rank	room
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	052
2134	Augustinus	C3	309
2136	Curie	C4	036
2137	Kant	C4	007

Integrity Constraints and Default Values (IV)

- Candidate keys
  - Any unique and minimal set of attributes that could serve but has not been selected as a primary key
  - The fact that the attributes $A_1, \ldots, A_n$ form a candidate key is specified by the integrity constraint **unique** ($A_1, \ldots, A_n$)
  - Later it will be an important task to find *all* candidate keys of a relation schema

Creation of a Relation Schema / Table Schema (I)

- Precisely speaking, there are no relations in SQL but tables (set model versus list model)
- Creation of a schema with the aid of the DDL clause

  create table R (A_1 $\boldsymbol{D_1}$, …, A_n $\boldsymbol{D_n}$,

  [< integrity constraint$_1$ >, …, < integrity constraint$_k$ >])

  R relation name, A_i name of an attribute in the schema of relation R, D_i domain of A_i

- In BNF notation
  - **create table** <relation name>(<relation comp> [, <relation comp>]*)
  - <relation comp> ::= <column definition> | <integrity constraint>
  - <column definition> ::= <attribute name> <type>
    [<default value> | **not null** | **unique**]
  - <default value> ::= [**default** <literal> | **null**]
  - The exact treatment of integrity constraints is discussed later.

Creation of a Relation Schema / Table Schema (II)

- Example: University schema (with the main integrity constraints)

```
create table students
    (reg-id   int not null,
     name     varchar(30) not null,
     sem      int,
     primary key (reg-id));
```

```
create table professors
    (pers-id  int not null,
     name     varchar(30) not null,
     room     int unique,
     rank     char(2),
     primary key (pers-id));
```

```
create table assistants
    (pers-id  int not null,
     name     varchar(30) not null,
     room     int unique,
     boss     int,
     primary key (pers-id),
     foreign key (boss) references
              professors(pers-id));
```

```
create table lectures
    (id         int not null,
     title      varchar(30),
     credits    int,
     held_by    int,
     primary key (id),
     foreign key (held_by) references
              professors(pers-id));
```

Creation of a Relation Schema / Table Schema (III)

- Example: University schema (with the main integrity constraints, continued)

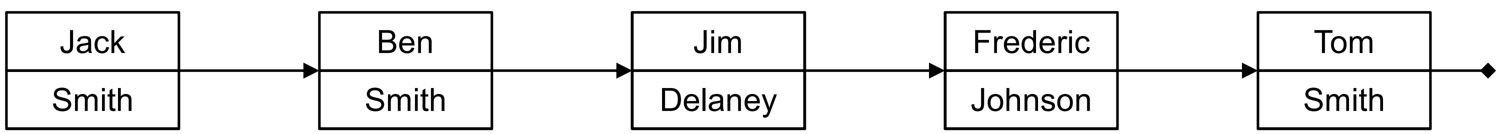
```
create table attends
    (reg-id    int not null,
     id        int not null,
     primary key (reg-id, id),
     foreign key (reg-id) references
                 students(reg-id),
     foreign key (id) references
                 lectures(id));

create table test
    (reg-id    int not null,
     id        int not null,
     pers-id   int not null,
     grade     numeric(2,1),
     primary key (reg-id, id, pers-id),
     foreign key (reg-id) references students(reg-id),
     foreign key (id) references lectures(id),
     foreign key (pers-id) references professors(pers-id));

create table is_precondition_of
    (predecessor   int not null,
     successor     int not null,
     primary key (predecessor,
                  successor),
     foreign key (predecessor)
             references lectures(id),
     foreign key (successor)
             references lectures(id));
```

Modification of a Relation Schema / Table Schema

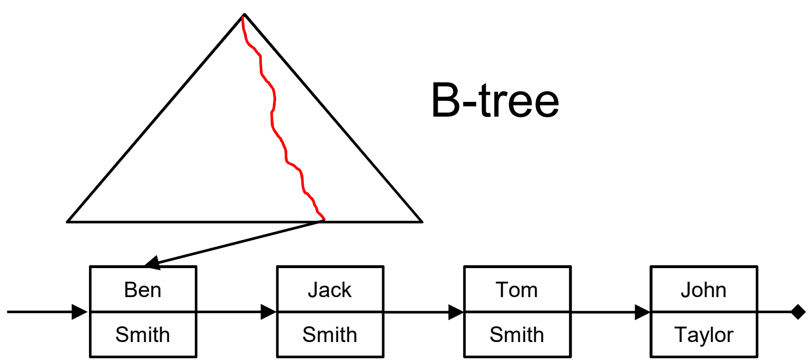
- Adding an attribute (a new column) to an *existing* relation
  - **alter table** <relation name> **add** <column definition>
  - All tuples in the relation are assigned *null* as the value for the new attribute
  - Constraint **not null** is only allowed if a default value is specified
- Deleting an attribute (a column) from an *existing* relation
  - **alter table** <relation name> **drop** <column definition>
  - Command will be declined if integrity constraints would be violated
  - Example 1: Delete a column that has values which are referenced by tuples from other relations
  - Example 2: Delete a column that forms the primary key of a relation
- Deletion of the schema and instance (that is, data) of a relation
  - **drop table** <relation name>
- Deletion of the instance (that is, data) of a relation but not its schema
  - **delete from** <relation name>

Creation of an Index (I)

- An index is a *persistent* data structure that provides accelerated access to the rows of a table based on the values of one or more attributes
- Goal: Significant improvement of query response time
- Example: List of tuples that store first and last names of persons



- A search for Tom Smith above takes $O(n)$ time (n number of tuples)



- A search for Tom Smith takes $O(\log n + k)$ time (k number of "Smith" tuples)
- A measure for the efficiency is, in general, the number of page accesses to the hard disk

Creation of an Index (II)

- Disadvantage: Indexes have to be updated by the system every time the underlying tables are updated

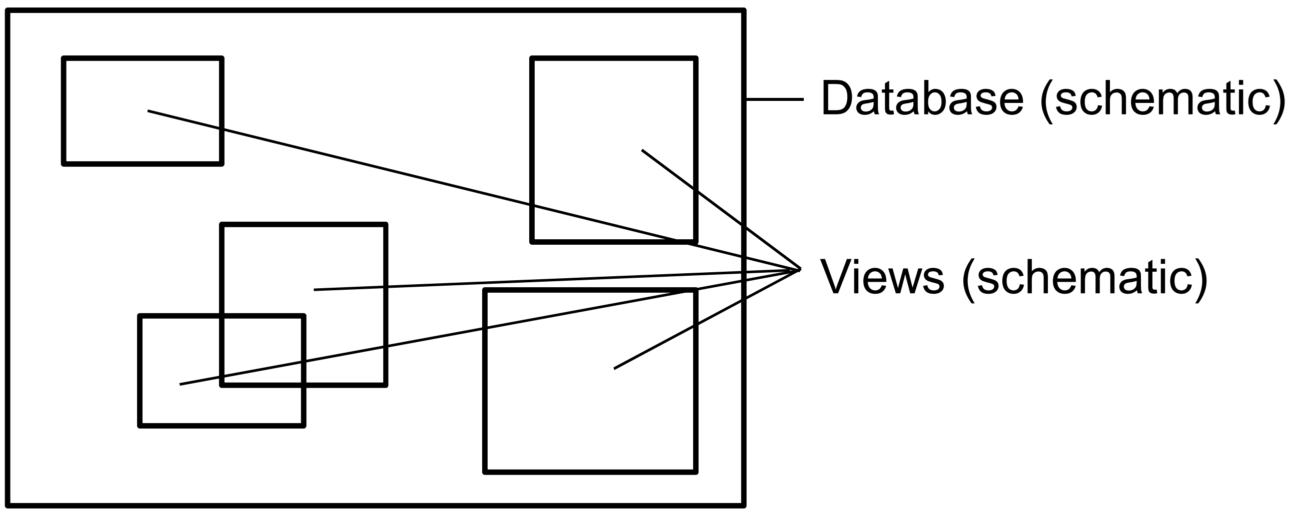
  → Too many indexes lead to a large overhead and slow down the performance of the DBMS, right balance needed

- SQL command for creating an index
  - **create** [**unique**] **index** <index name> **on** <relation name>

    (<attribute name> [<order>] [, <attribute name> [<order>]]*) [**cluster**]
  - <order> ::= **Asc** | **Desc**
  - **unique** means that two tuples with the same values for all indexed attribute names are forbidden (indexed attributes fulfil key condition)
  - **cluster** means that the tuples of the relation are actually inserted into the index structure by physical proximity (only one cluster index per relation)
- Example: **create unique index** room_index **on** professors (room);
- SQL command for deleting an index
  - **drop index** <index name>

Creation of Views (I)

- A view is a *virtual relation* that does not exist persistently in the database but can be produced upon request by a particular user, at the time of request



- Views are regarded as derived relations which are defined by queries
- **create view** <view name> [(<attribute name> [, <attribute name>]*)]
  as <subquery>

Creation of Views (II)

- Example

  create view major_students **as**

          select * **from** students **where** sem > 4;

- The keyword “*” is a shortcut for the complete attribute list of those relations placed after **from**

- Deletion of views

  drop view <view name>