# The Relational Model

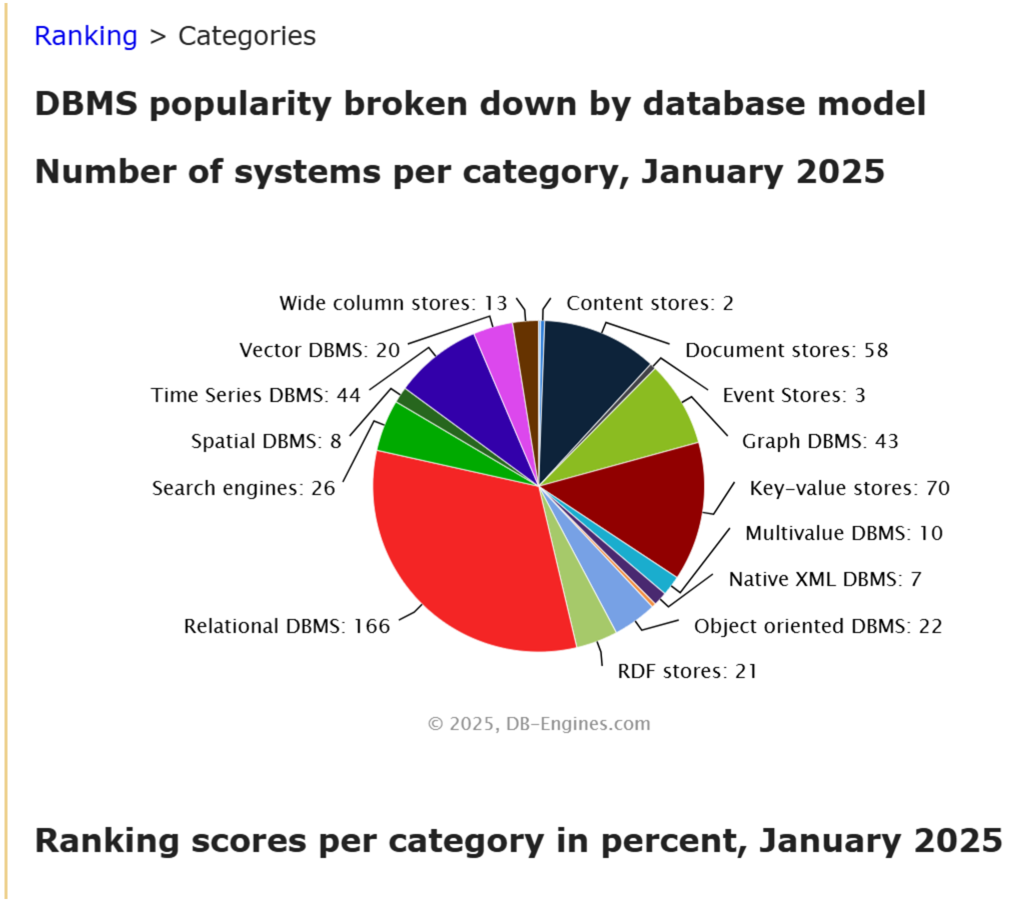
# Database Models

- The entity-relationship is a semantic model (quite rich in its expressiveness) but it is not used as a model in any DBS.
- Relational model
  - Most common
  - Vendors: IBM DB2, Microsoft SQL Server, Oracle, SAP Hana, SAP Sybase, etc…
  - Open-source: PostgreSQL, MySQL, SQLite, Derby, MonetDB,
- "Legacy systems" have older models
  - E.g., IBM's IMS (hierarchical), CODASYL network model

# Database Models

- Recent and future competitors:
  - object-oriented model: ObjectStore, Versant
  - Semi-structured (document-based): XML, JSON
    - *MongoDB*, CouchDB
  - Key-value
    - *Riak*, Voldemort, *Amazon's Dynamoth*
  - Column-store (key – basic relational)
    - *Google's BigTable*, Hadoop's Hbase, Cassandra
  - Integrated: object-relational, XML+relational …

# Database Engines in Use



Ranking > Categories

**DBMS popularity broken down by database model**

**Number of systems per category, January 2025**

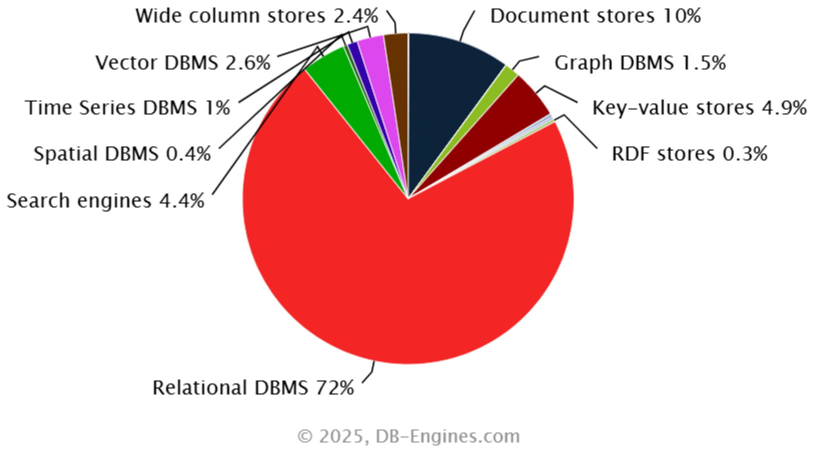
**Ranking scores per category in percent, January 2025**

DB-Engines lists 423 different database management systems, which are classified according to their database model (e.g. relational DBMS, key-value stores etc.).
This pie-chart shows the number of systems in each category. Some of the systems belong to more than one category.

https://db-engines.com/en/ranking_categories

# Database Engines in Use

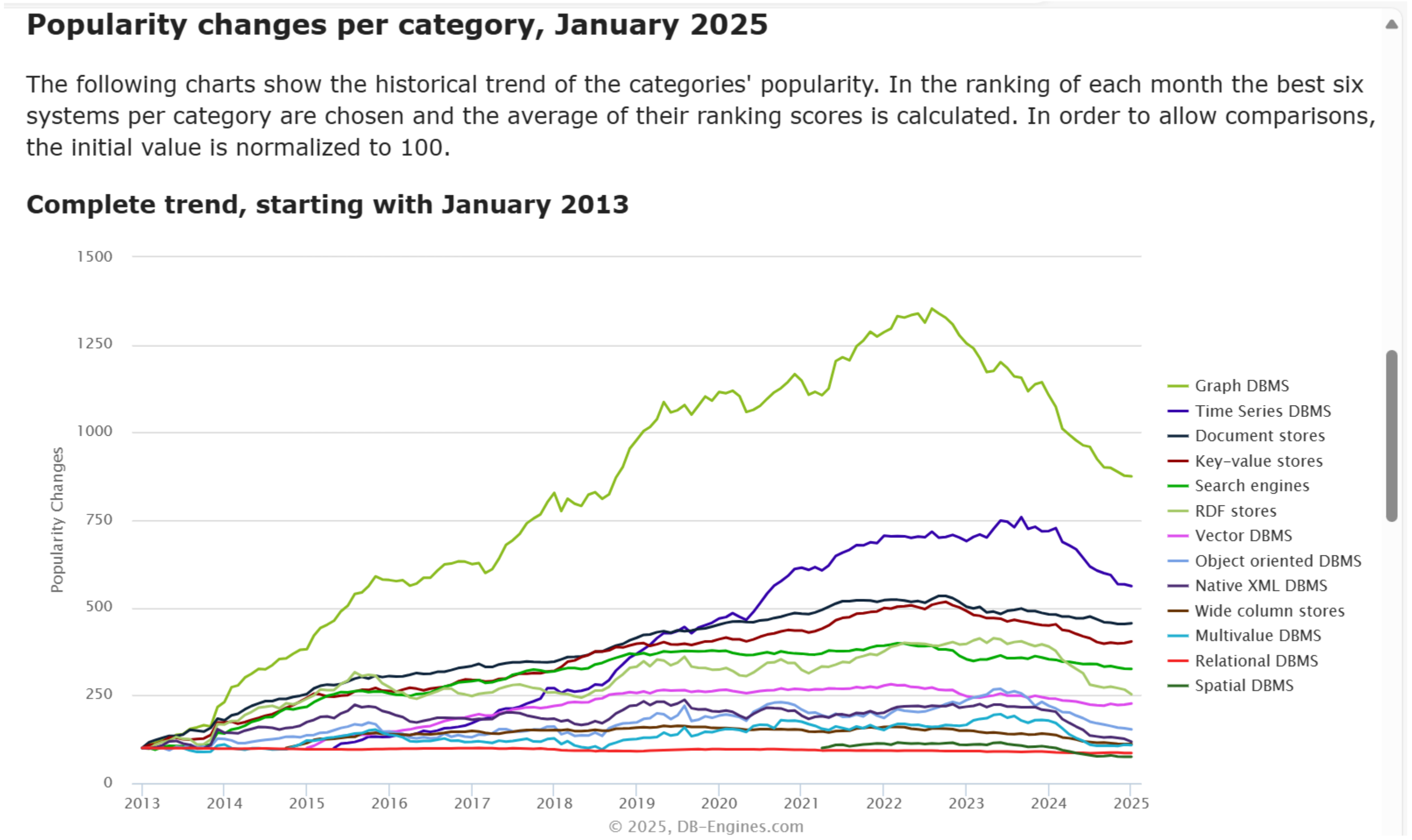
**Ranking scores per category in percent, January 2025**



This chart shows the popularity of each category. It is calculated with the popularity (i.e. the ranking scores) of all individual systems per category. The sum of all ranking scores is 100%.

https://db-engines.com/en/ranking_categories

# Database Engines in Use



**Popularity changes per category, January 2025**

The following charts show the historical trend of the categories' popularity. In the ranking of each month the best six systems per category are chosen and the average of their ranking scores is calculated. In order to allow comparisons, the initial value is normalized to 100.

**Complete trend, starting with January 2013**

https://db-engines.com/en/ranking_categories

# Definitions

- **Relational Database:** a set of relations
- **Relation:** Consists of two parts:
  - **Schema:** specifies name of relation, plus a set of attributes, plus the domain/type of each attribute
    - E.g., `Students(sid:`*`int`*`, name:`*`string`*`, login:`*`string`*`, faculty:string, major:`*`string`*`)`
  - **Instance:** set of tuples (all tuples are distinct).
  - Compare with entity set and entity; or with object class and object instance
- A relation can be seen as a table:
  - Column headers = attribute names, rows = tuples/records, columns/fields = attribute values
- If clear from context, we say instead of "instance of a relation" simply "relation"
- Database Schema: collection of relation schemas

# Example of *Students* Relation

Column headers = attributes

| sid | name | login | faculty | major |
|---|---|---|---|---|
| 53666 | Bartoli | bartoli@cs | Science | Software Engineering |
| 53688 | Chang | chang@eecs | Eng | Software Engineering |
| 53650 | Chang | chang@math | Science | Computer Science |
| … | | | | |

- All rows are distinct (set-oriented)
- Rows are not ordered (a permutation of rows represents still the same table)
- Columns are per definition not ordered but in practice we often assume a fixed order
  - with this, a single tuple can be represented as
    - `(53666, Bartoli, bartoli@cs, Science, Software Engineering)`

# Relational DDL and DML

- **Data Definition Language (DDL):** defines the schema of a database
- **Data Manipulation Language (DML):** "manipulates" the data, i.e., the instances of the relations
  - Insert, update, delete tuples
  - "Query" the relations: retrieve tuples that fulfill certain criteria (hence, often called "query language")
  - The Relational Model offers simple and powerful querying of data with precise semantics
  - Data-centric
- *Physical data independence*
  - User only sees relations
  - User does not need to know how data is stored
    - Mapping to files
    - Rows vs. columns
  - User does not need to know how queries are executed

# The SQL Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - …
  - SQL-99 / SQL3 (adds object-relational features)
  - SQL:2003 (adds XML features)
  - …
  - SQL:2011
- Used to define relations (Data definition)
- Used to write and query data (Data manipulation)
- Standard is very slow to develop
- Many dialects in various DBS

# Concepts vs. Details

- We will discuss the main concepts of SQL (data definition and data manipulation) in class
- To figure out the details, students have to use the handbook
  - In particular: DBS specific dialects, variations…

# SQL Data Types

- All attributes must have a data type.
- SQL supports several basic data types
  - Just as any programming language does
  - Notation somewhat "old-fashioned" as language very old
  - Character and string types
    - CHAR(n) denotes a character string of fixed length (containing trailing blanks for padding if necessary).
    - VARCHAR(n) denotes a string of up to n characters (between 0 and n characters).
- Integer Types
  - INT or INTEGER (names are synonyms)
  - SHORTINT

# Data Types (contd.)

- Floating point numbers
  - FLOAT or REAL (names are synonyms)
  - DOUBLE PRECISION
  - DECIMAL(n,d): real number with fixed decimal point. Value consists of n digits, with the decimal point d positions from the right.
- Dates and time:
  - DATE: has the form 'YYYY-MM-DD'
  - TIME: has the form '15:00:02' or '15:00:02.5'
  - May be compared and converted to string types
- Bit strings
- User defined domains
  - New name for a data type
  - Possibility to define restrictions on values of domain (< 10)

# Data Definition: Table Creation

- Defines all attributes of the relation
- The type/domain of each attribute is specified
- DBMS enforce correct type whenever a tuple is added or modified
- SQL is case insensitive
- It is possible to define <u>default</u> values

```
CREATE TABLE Students
  (sid INTEGER,
   name VARCHAR(30),
   login VARCHAR(30),
   faculty VARCHAR(20),
   major VARCHAR(20) DEFAULT‘undefined’)
```

## Data Manipulation: Insert

## Data Manipulation: Insert

- Insert a single tuple

```
INSERT INTO Students
     VALUES (53666, 'Bartoli', 'bartoli@cs',
             'Science', 'Software Engineering')
```

  – can contain all or only a subset of attributes:
    - Not indicated attributes have special NULL value

```
INSERT INTO Students (sid,name,faculty)
     VALUES(53688, 'Chang', 'Eng')

INSERT INTO Students (sid,name,major)
     VALUES (53650, 'Chang', 'Computer Science')
```

| sid | name | login | faculty | major |
|---|---|---|---|---|
| 53666 | Bartoli | bartoli@cs | Science | Software Engineering |
| 53688 | Chang | NULL | Eng | undefined |
| 53650 | Chang | NULL | NULL | Computer Science |

# Data Manipulation: Delete

- Can delete all tuples satisfying some condition

```
DELETE
FROM Students
WHERE name = 'Chang'
```

| sid | name | login | faculty | major |
|---|---|---|---|---|
| 53666 | Bartoli | bartoli@cs | Science | Software Engineering |
| 53688 | Chang | NULL | Eng | NULL |
| 53650 | Chang | NULL | NULL | Computer Science |

## Data Manipulation: Update

- Can update all tuples satisfying some condition

```
UPDATE Students
SET major = 'Software Engineering'
WHERE sid = 53688
```

| sid | name | login | faculty | major |
|---|---|---|---|---|
| 53666 | Bartoli | bartoli@cs | Science | Software Engineering |
| 53688 | Chang | NULL | Eng | NULL Software Engineering |
| 53650 | Chang | NULL | NULL | Computer Science |

# Querying the Data

- Find the names and major of all students in the Faculty of Science

```
SELECT name, major
FROM Students
WHERE faculty = 'Science'
```

| sid | name | login | faculty | major |
|---|---|---|---|---|
| 53666 | Bartoli | bartoli@cs | Science | Software Engineering |
| 53688 | Chang | NULL | Eng | NULL |
| 53650 | Chang | NULL | Science | Computer Science |

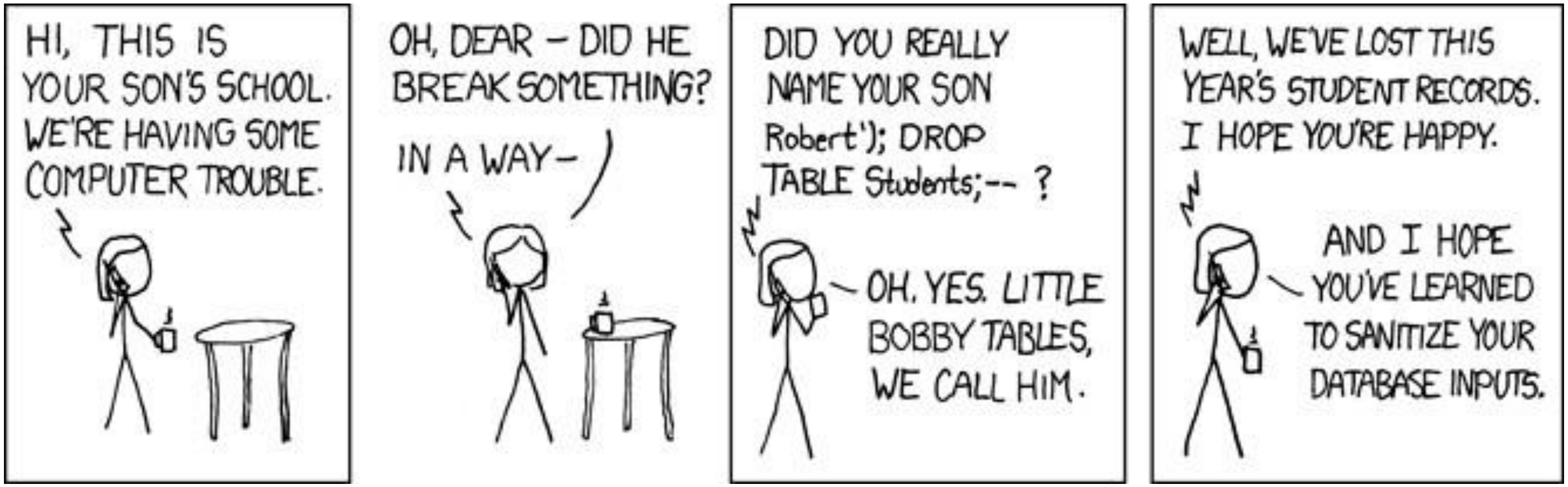
| | |
|---|---|
| Bartoli | Software Engineering |
| Chang | Computer Science |

Much more of that in a few weeks…

# Data Definition: Destroying Relations

```
DROP TABLE Students
```

- Destroys the relation Students. The schema information *and* the tuples are deleted.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

# Data Definition: Altering Tables

```
DROP TABLE Students
```

- Destroys the relation Students. The schema information *and* the tuples are deleted.

```
ALTER TABLE Students
     ADD COLUMN firstYear:integer
```

- The schema of students is altered by adding a new field;
- every existing tuple in the current instance is extended with a **null** value in the new field.

## Integrity Constraints (ICs)

- Integrity Constraints must be true for any instance of the database;
  - Domain definitions (INT or FLOAT) are already a form of constraint
  - Database designer specifies ICs when schema is defined
  - DBMS checks whether ICs remain true whenever relations are modified.
    - If modification violates an IC, the DBMS disallows the modification (throws an error message)
  - Of course, DBMS can only check what is specified in the schema

# Not Null

```
CREATE TABLE Students
   (sid CHAR(9),
    name VARCHAR(30) NOT NULL,
    login VARCHAR(30),
     faculty VARCHAR(20),
    major VARCHAR(20)
        DEFAULT ‘undefined’)
```

requires an attribute to always have a proper value

# Primary Key Constraints

- A set of fields is a **key candidate** for a relation if
  - No two distinct tuples can have same values in all key fields, and
  - This is not true for any subset of the key.
    - Minimum subset of attributes that fulfill uniqueness property
- If there are two or more keys, one of the candidates is chosen to be the **primary key.**
- The primary key attributes of a tuple may not be NULL.
- E.g. **`sid`** is a key for **`Students`**. (What about **`name`**?).
- Example of combined keys:
  - `Location(building, roomNo, capacity)`
  - Attributes `building` and `roomNo` together build the primary key

# Primary and Candidate Keys in SQL

- Possibly many candidate keys exist, one of which is chosen as the primary key
  - Each student has a unique id.
  - Each student has a unique login

```
CREATE TABLE Students
    (sid CHAR(9) PRIMARY KEY,
     login VARCHAR(30) NOT NULL UNIQUE,
     name VARCHAR(20),
     …)
```

What if UNIQUE without NOT NULL? **--->** Implies ONLY one NULL for that Attribute across the relation tuples. Most DB engines requires NOT NULL whenever UNIQUE is used

- Defining primary keys that have more than one attribute

```
CREATE TABLE Location
    (building VARCHAR(20),
     roomNo    INT,
    capacity INT,
    PRIMARY KEY(building, roomNo)
)
```

I ONLY MADE ONE CHANGE TO THE DATABASE
I JUST CHANGED THE PRIMARY KEY ON EVERY TABLE

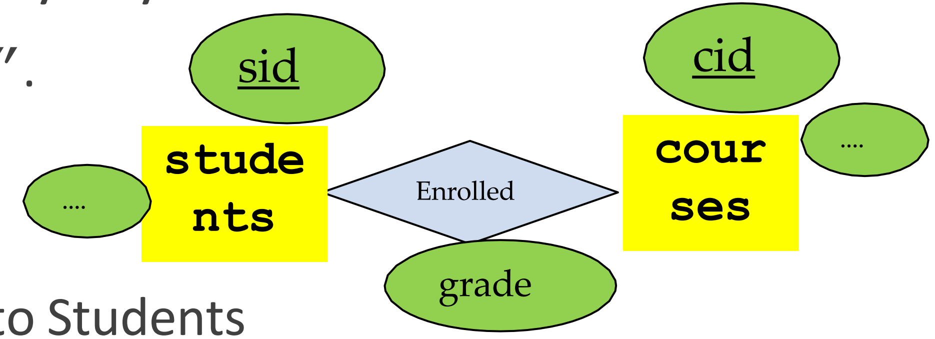
# Note

- Specifics of individual database systems
  - E.g., DB2 required the following
    - If primary key, then also NOT NULL must be defined
  - If there are error messages
    - Consult the handbooks of the resp. database system

- Multi-Set
  - Tables do not require to have a primary key
  - If there is no primary key
    - Two records/tuples can have the same value in all attributes
    - That is, duplicates are allowed
  - Different to Relational Algebra Assumptions
    - Relations are set ➡ no identical tuples in a relation

# Foreign Key

- **Foreign Key**: Set of attributes in one relation R that is used to "refer" to a tuple in another relation Q.
  - Must correspond to the primary key of the second relation Q.
  - Represents a "logical pointer".
- Examples
  - in relation `Enrolled`,
    - `sid` is a foreign key referring to Students



**Students Relation (Referenced)**

| sid | name | login | faculty | … |
|---|---|---|---|---|
| 53666 | Bartoli | bartoli@cs | Science | … |
| 53688 | Chang | chang@eecs | Eng | … |
| 53650 | Chang | chang@math | Science | … |
| … | | | | |

**Enrolled Relation (Referencing)**

| sid | cid | grade |
|---|---|---|
| 53666 | Topology112 | C |
| 53666 | Reggae203 | B |
| 53650 | Topology112 | A |
| 53668 | History105 | B |

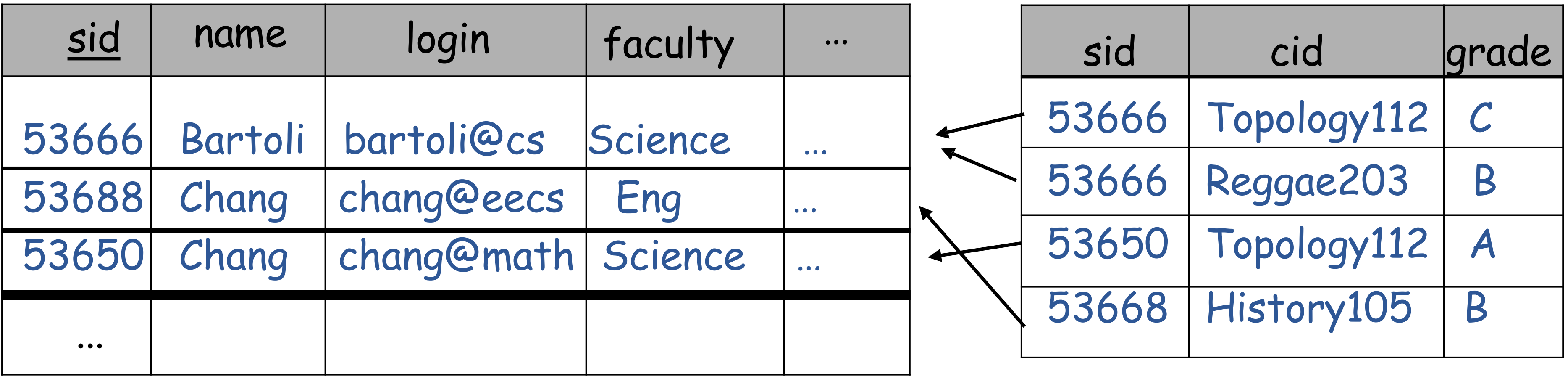
# Referential Integrity

- **Foreign Key Constraint:** the foreign key value of a tuple must represent an existing tuple in the referred relation
  - A tuple with this value in the primary key must exist in the referred relation
  - Enrollment may only contain a tuple referring to a student who exists in the Students relation
- If all foreign key constraints are enforced, **referential integrity** is achieved, i.e., no dangling references

# Foreign Keys in SQL

Only students listed in the Students relation should be allowed to enroll for courses

```
CREATE TABLE Enrolled
(
    sid CHAR(9),
    cid VARCHAR(20),
    grade CHAR(2),
    PRIMARY KEY (sid,cid),
    FOREIGN KEY (sid) REFERENCES Students,
    FOREIGN KEY (cid) REFERENCES Courses
)
```



| <u>sid</u> | name | login | faculty | … |
|---|---|---|---|---|
| 53666 | Bartoli | bartoli@cs | Science | … |
| 53688 | Chang | chang@eecs | Eng | … |
| 53650 | Chang | chang@math | Science | … |
| … | | | | |

| sid | cid | grade |
|---|---|---|
| 53666 | Topology112 | C |
| 53666 | Reggae203 | B |
| 53650 | Topology112 | A |
| 53668 | History105 | B |

# Enforcing Referential Integrity: Default

- An `Enrolled` tuple with a sid is inserted but no tuple with this sid exists in `Students`
  - Disallow insertion
- A `Students` tuple is deleted
  - Disallow the deletion of a `Students` tuple to which `Enrolled` tuples point
- Other options:
  - cascade