CS 1555
www.cs.pitt.edu/~nlf4/cs1555/

Using SQL as a DDL
SQL

- Declarative query language
- Originally developed for System R
  - First presented in 1974
- Current incarnation is the result of a very successful standardization effort by both ANSI and ISO
  - SQL-86 (SQL)
  - SQL-89 (SQL1)
  - SQL-92 (SQL2)
- De-facto language for RDBMS
  - DDL, DML and VDL
The database schema

- Describes the data stored in the database

Specifically:
  - Base relations
    - (tbl_name, creator, #tuples, tuple_length, #attributes...)
  - Attributes of relations (columns)
    - (tbl_name, atrb_name, type, format, order, key_no, ...)
  - Indexes
    - (tbl_name, index_name, key_attribute,...)
  - Authorizations
  - Integrity constraints

All of this information is stored in the *catalog*

SQL-92 and later allow multiple database schemas
  - Tables are named as SchemaName.TableName
Creating and deleting schemas

- CREATE SCHEMA name AUTHORIZATION user;
- DROP SCHEMA name [RESTRICT | CASCADE];
  - RESTRICT: removes schema if it doesn't contain any elements
  - CASCADE: remove schema and everything it contains
Creating database tables

- CREATE `TABLE` `tname` (`attribute list` ...);
  - Need to specify a name and a data type for each attribute
Numerical data types

● **C-style integers**
  ○ SMALLINT
  ○ INT

● **Floating point types**
  ○ FLOAT[:precision:]
    ■ Allows custom precision floating point types
  ○ REAL and DOUBLE
    ■ Set precision floating point types

● **DECIMAL**(i, j) or **NUMERIC**(i, j)
  ○ i: total number of digits (precision)
  ○ j: digits after a decimal point (scale)
Very few DBMSs "speak" standard SQL

- In reality, most will speak its own SQL dialect
  - Integer data type example:
    - **PostgreSQL:**
      - SMALLINT (16 bit)
      - INTEGER (32 bit)
      - BIGINT (64 bit)
    - **MySQL:**
      - SMALLINT (16 bit)
      - INT (32 bit)
      - BIGINT (64 bit)
    - **Oracle:**
      - SHORTINTEGER (16 bit)
      - INTEGER (32 bit)
      - LONGINTEGER (64 bit)
- Always consult the documentation for your DBMS!
Character string data types

- Strings of *printable* characters
- Enclosed in 'single quotes'
- CHAR(n) or CHARACTER(n)
  - Fixed length n strings
- VARCHAR(n) or VAR(n) or CHAR VARYING(n)
  - Variable length string (max of n)
- Concatenation operator:  ||
  - 'abc' || 'XYZ' results in 'abcXYZ'
- CLOB(size)
  - Character Large OBject
  - size specified in kilobytes (K), megabytes (M), or gigabytes (G)
Bit string data types

- Sequences of bits
- Enclosed in single quotes with a leading B
  - B'001100101'
- BIT(n)
  - Fixed length of n bit
- VARBIT(n) or BIT VARYING(n)
  - Variable length (max of n)
- BLOB(size)
  - Binary Large OBject
  - size specified in kilobytes (K), megabytes (M), or gigabytes (G)
Boolean values

- Valued TRUE or FALSE
NULL values

Several reasons to store a NULL value in a tuple:

- Unknown value
  - A person’s date of birth is not known, so it is represented by NULL in the database.

- Unavailable or withheld value
  - A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.

- Not applicable attribute
  - An attribute LastCollegeDegree would be NULL for a person who has no college degrees because it does not apply to that person.
Three-valued logic

- TRUE, FALSE, or UNKNOWN
- Consider:
  - Storing a NULL value for a BOOLEAN attribute
    - How should this be treated in a logical expression?
  - Evaluating a condition on a NULL value
    - Consider the partial condition:
      - … Students.Name = 'SUSAN' AND Students.GPA > 2.0 …
        - If a row in the students table has a value of 3.0 for the GPA attribute and a NULL value for the Name attribute, should this condition be TRUE or FALSE?
        - What about NULL name, but a 1.0 for GPA?
Date and time data types

- **DATE**
  - YYYY-MM-DD

- **TIME**
  - HH:MM:SS

- **TIME(i)**
  - HH:MM:SS.dddd .. d

- **TIME WITH TIME ZONE**
  - Adds displacement of -13:00 to +12:59
  - HH:MM:SS{+-}hh:mm
  - If not specified, default is local timezone

- **TIMESTAMP**
  - Complete date and time with 6 fractional seconds and optional time zone

- **Dates and times must be valid!**
Date and time implementations

- PostgreSQL sticks pretty close to SQL standard
- MySQL implements both TIMESTAMP and DATETIME
  - DATETIME is not a valid ANSI type
  - DATETIME range:
    - '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
  - TIMESTAMP range in MySQL:
    - '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC
- Oracle DATE is not equivalent to ANSI DATE, it instead functions like ANSI TIMESTAMP
Intervals

- Represent periods of time and are used in operations on date and time data types
- DATE {+,−} INTERVAL results in DATE
- DATE - DATE results in INTERVAL
- INTERVAL {*/*} number results in INTERVAL
- INTERVAL {+,−} INTERVAL results in INTERVAL

Examples
- (CURRENT_DATE + INTERVAL '1' MONTH)
- (CURRENT_DATE + INTERVAL '18' DAY)
- (CURRENT_DATE - BirthDate)
- Different DBMSs tend to use specialized DATE and TIME functions to manipulate DATE, TIME, and INTERVAL data
- Be sure to check the documentation of your DBMS
Interval data types

- INTERVAL start-field(p) [TO end-field(fs)]
  - p specifies the precision of the start field, defaults to 2
  - fs specifies the fractional seconds for ending times

- Year-Month
  - INTERVAL YEAR, INTERVAL YEAR(p), INTERVAL MONTH, INTERVAL MONTH(p), INTERVAL YEAR TO MONTH, INTERVAL YEAR(p) TO MONTH

- Day-Time
  - E.g., INTERVAL DAY(10) TO HOUR, INTERVAL DAY TO MINUTE, INTERVAL SECOND(5), INTERVAL DAY(5) TO SECOND, INTERVAL MINUTE TO SECOND
All the data types you could want!

- **DOMAIN** defines datatype macros in a schema
  - Basic datatype
  - DEFAULT value
  - CHECK (validity conditions)

- **DROP DOMAIN** `dname` `[RESTRICT|CASCADE];`
  - RESTRICT drops domain if it is unused
  - CASCADE drops domain and replaces it with underlying type

- **Examples:**
  - `CREATE DOMAIN sectno_dom AS SMALLINT;`
  - `CREATE DOMAIN section_dom VARCHAR(20) DEFAULT 'none';`
  - `CREATE DOMAIN gpa_dom DECIMAL(3,2) DEFAULT 0.00;`
  - `CREATE DOMAIN ssn_dom CHAR(9)`
    - CONSTRAINT `ssn_dom_value` CHECK (VALUE BETWEEN '000000000' AND '999999999');
Constraints on table attributes

- Constraints:
  - NOT NULL
  - DEFAULT value
    - without the DEFAULT clause, the default value is NULL
  - PRIMARY KEY (attribute_list)
  - UNIQUE (attribute list)
    - allows the specification of alternative key
  - FOREIGN KEY (key) REFERENCES table (key)
CREATE TABLE Students

(  
  ID INTEGER,
  Name VARCHAR(20),
  Major VARCHAR(10),
  GPA DECIMAL(3,2),
  CONSTRAINT Students_PK PRIMARY KEY (ID)
);

CREATE TABLE Enrollment
(
  Stud_ID INTEGER,
  Course VARCHAR(15),
  CONSTRAINT Enrollment_FK
    FOREIGN KEY (Stud_ID) REFERENCES Students(ID)
);

CREATE TABLE Students
(
    ID INTEGER,
    Name VARCHAR(20),
    Ssn CHAR(9) NOT NULL,
    Major VARCHAR(10),
    GPA DECIMAL(3,2),
    CONSTRAINT Students_PK PRIMARY KEY (ID),
    CONSTRAINT Students_AK UNIQUE (Ssn)
);
Modifying tables after creation

- `ALTER TABLE tname ALTER COLUMN cname options;`
  - `ALTER TABLE Students ALTER COLUMN GPA DECIMAL(4,2);`
  - `ALTER TABLE Students ALTER COLUMN GPA SET DEFAULT NULL;`
  - `ALTER TABLE Students ALTER COLUMN GPA DROP DEFAULT;`
- `ALTER TABLE tname ADD COLUMN cname type;`
- `ALTER TABLE tname DROP COLUMN cname [RESTRICT | CASCADE];`
- `ALTER TABLE tname ADD CONSTRAINT con_name description;`
- `ALTER TABLE tname DROP CONSTRAINT con_name;`
Dropping tables

- **DROP TABLE** `tname` [**RESTRICT** | **CASCADE**];
  - **RESTRICT** drops the table only if it is not referenced
    - E.g., by constraints or views
  - **CASCADE** drops the table and items that reference it