As we’ll see, we have to specify types when we create a table. You can see types for columns in a table when you do `\d tablename` in psql:

```sql
\d mines_courses
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>crn</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>course_id</td>
<td>text</td>
<td>not null</td>
</tr>
<tr>
<td>section</td>
<td>text</td>
<td>not null</td>
</tr>
<tr>
<td>title</td>
<td>text</td>
<td>not null</td>
</tr>
<tr>
<td>instructor</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>enrollment</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

SQL provides lots of types, and most DBMSes add custom types. Types of types:

- Numbers
  - Integers
  - Fixed-precision
  - Floating point
- Strings
- Date/time

https://www.postgresql.org/docs/9.5/static/datatype.html

### Integers

- INTEGER – 32-bit integers
- SMALLINT – 16-bit integers
- BIGINT – 64-bit integers

### Fixed-Precision Numeric

These are exact decimal numbers (cf. FP):

- `NUMERIC(w,p)` - Max w digits, precision of p
  - e.g. `NUMERIC(5,2)` can hold values between -999.99 and 999.99.
  - If you insert a number larger in magnitude than 999.99, you get an overflow error.
  - If you insert a number with more than 2 places past the decimal point, your value will be rounded.
- `DECIMAL(w,p)` – Same as `NUMERIC(w,p)`
Floating Point

Inexact real numbers:
- REAL – 32-bit floating point
- DOUBLE PRECISION – 64-bit floating point

Strings

- CHAR(n) – strings of length exactly n, padded with spaces if necessary
- VARCHAR(n) – strings of length at most n
- TEXT – arbitrary, variables length strings
  (PostgreSQL and many others, not standard)

Note: in PostgreSQL, CHAR(n) acts mostly like VARCHAR(n), and the type VARCHAR – no "n" parameter – acts like TEXT. Not standard!

Dates

DATE – holds dates.
SQL will automatically convert from various text formats into dates for you (DBMS-dependent). Be careful that you get what you expect, though!
Here are some examples:
- "2005-06-11" = June 11, 2005
- "03-AUG-35" = August 3, 2035
- "03-AUG-1935" = August 3, 1935
- "4/30/78" = April 30, 1978 – at least, in the U.S.
- "September 1, 2018" = well, you can guess

DATE values are very powerful – you can do cool things like add an integer to a date, and it will compute the correct date that many days later.

Times & Timestamps

- Lots of flavors:
  - With/without time zone
  - Various precisions (sub-second)
  - TIME just records times
  - TIMESTAMP records date & time
- Lots of input/output formats – see docs
- Also, check out the INTERVAL type

Miscellaneous

- BOOLEAN
- SERIAL – auto-incrementing integer pseudo-type
- MONEY
- UUID

…and many more

Type Conversion

Two syntaxes, by example:

SELECT CAST('23-AUG-04' AS DATE);

SELECT '23-AUG-04'::DATE;
**TABLE CREATION**

---

Simple Table Creation

CREATE TABLE name (column1 type1, column2 type2, ...);

e.g.,
CREATE TABLE junk (x INTEGER, y DATE);

---

More General

CREATE TABLE [schemaname.]tablename ( 
  columnname datatype [NOT NULL] [UNIQUE] [PRIMARY KEY] | table constraint } [,...])

This form includes things like constraints, which we haven’t talked about yet. Stay tuned.

Actually, the above is a simplification... see https://www.postgresql.org/docs/9.5/static/sql-createtable.html (Be afraid.)

---

CREATE...AS

Easy way to create a table from a SELECT query:

CREATE TABLE name AS SELECT ...

e.g.,
CREATE TABLE cs_courses AS SELECT * FROM mines_courses WHERE course_id LIKE 'CSCI%';

---

Simple INSERT

There are more details to this, but just so you can get started:

INSERT INTO name VALUES (val1, val2, ...);

e.g.
CREATE TABLE junk (x INTEGER, y DATE);
INSERT INTO junk VALUES (4, '2010-07-06');
INSERT INTO junk VALUES (33, '2015-11-29');
SELECT * FROM junk;

---

Up Next

- Next lecture:
  Data modification queries: INSERT, UPDATE, DELETE