

# SQL Introduction

---

- SQL vs Relational Model
- SQL History
- SQL Intro
- SQL Syntax in a Nutshell
- Names in SQL
- Types/Constants in SQL
- Examples of Defining Domains/Types
- Tuple and Set Literals

## ❖ SQL vs Relational Model

---

The **relational model** is a formal system for

- describing data (relations, tuples, attributes, domains, constraints)
- manipulating data (relational algebra ... covered elsewhere)

**SQL** is a "programming" language for

- describing data (tables, rows, fields, types, constraints)
- manipulating data (query language)

SQL extends the relational model in some ways (e.g bags vs sets of tuples)

SQL omits some aspects of the relational model (e.g. general constraints)

## ❖ SQL History

---

Developed at IBM in the mid-1970's (System-R)

Standardised in 1986, and then in 1989, 1992, 1999, 2003, ... 2019

Many database management systems (DBMSs) have been built around SQL

- System-R, Oracle, Ingres, DB2, PostgreSQL, MySQL, SQL-server, SQLite, ...

DBMSs vs the standard

- all DBMSs implement a subset of the 1999 standard (aka SQL3)
- all DBMSs implement proprietary extensions to the standard

Conforming to standard should ensure portability of database applications

## ❖ SQL Intro

---

SQL has several sub-languages ...

- meta-data definition language (e.g. `create table`, etc.)
- meta-data update language (e.g. `alter table`, `drop table`)
- data update language (e.g. `insert`, `update`, `delete`)
- query language (e.g. `select ... from ... where`, etc.)

Meta-data languages manage the database **schema**

Data update language manages sets of **tuples**

## ❖ SQL Intro (cont)

Syntax-wise, SQL is similar to other programming languages

- has keywords, identifiers, constants, operators
- but strings are different to most PLs
  - '...' are constant strings, e.g. 'a', 'abc123', 'John' 's bag'
  - "... " allow non-alpha chars in identifiers and make id's case-sensitive

In the standard, all non-quoted identifiers map to all upper-case

- e.g. BankBranches = bankbranches are treated as BANKBRANCHES

In PostgreSQL, all non-quoted identifiers map to all lower-case

- e.g. BankBranches = BANKBRANCHES are treated as bankbranches

In all standards-adhering DBMSs, different quoted identifiers are different

- "BankBranches" ≠ "bankbranches" ≠ "BANKBRANCHES"

## ❖ SQL Syntax in a Nutshell

---

SQL definitions, queries and statements are composed of:

- **comments** ... `--` comments to end of line
- **identifiers** ... similar to regular programming languages
- **keywords** ... a large set (e.g. CREATE, DROP, TABLE)
- **data types** ... small set of basic types (e.g. integer, date)
- **operators** ... similar to regular programming languages
- **constants** ... similar to regular programming languages

*Similar* means "often the same, but not always" ...

## ❖ SQL Syntax in a Nutshell (cont)

Comments: everything after `--` *is a comment*

Identifiers: alphanumeric (a la C), but also "An Identifier"

Reserved words: many e.g. CREATE, SELECT, TABLE, ...

Reserved words cannot be used identifiers unless quoted e.g. "table"

Strings: e.g. 'a string', 'don't ask', but no '\n' (use e' \n')

Numbers: like C, e.g. 1, -5, 3.14159, ...

Types: integer, float, char(*n*), varchar(*n*), date, ...

Operators: =, <>, <, <=, >, >=, AND, OR, NOT, ...

## ❖ Names in SQL

---

Identifiers denote:

- database objects such as tables, attributes, views, ...
- meta-objects such as types, functions, constraints, ...

Naming conventions that I (try to) use in this course:

- relation names: e.g. `Branches`, `Students`, ... (use plurals)
- attribute names: e.g. `name`, `code`, `firstName`, ...
- foreign keys: named after either or both of
  - table being referenced e.g. `staff` or `staff_id`, ...
  - relationship being modelled e.g. `teaches`, ...

We initially write SQL keywords in all upper-case in slides.



## ❖ Types/Constants in SQL

Numeric types: INTEGER, REAL, NUMERIC (*w*, *d*)

10      -1      3.14159      2e-5      6.022e23

String types: CHAR (*n*), VARCHAR (*n*), TEXT

'John'      'some text'      '!%#%!\$'      'O''Brien'  
'"'      '[A-Z]{4}\d{4}'      'a VeRy! LoNg String'

PostgreSQL provides extended strings containing \ escapes, e.g.

E' \n'      E' O\ 'Brien'      E' [A-Z]{4} \d{4}'      E' John'

Type-casting via *Expr* :: *Type* (e.g. '10' :: integer)

## ❖ Types/Constants in SQL (cont)

**Logical type:** BOOLEAN, TRUE and FALSE (or true and false)

**PostgreSQL** also allows 't', 'true', 'yes', 'f', 'false', 'no'

**Time-related types:** DATE, TIME, TIMESTAMP, INTERVAL

'2008-04-13'   '13:30:15'   '2004-10-19 10:23:54'  
'Wed Dec 17 07:37:16 1997 PST'  
'10 minutes'   '5 days, 6 hours, 15 seconds'

**Subtraction of timestamps yields an interval, e.g.**

`now()::TIMESTAMP - birthdate::TIMESTAMP`

**PostgreSQL** also has a range of non-standard types, e.g.

- geometric (point/line/...), currency, IP addresses, JSON, XML, objectIDs, ...
- non-standard types typically use string literals ('...') which need to be interpreted

## ❖ Types/Constants in SQL (cont)

Users can define their own types in several ways:

-- domains: constrained version of existing type

```
CREATE DOMAIN Name AS Type CHECK ( Constraint )
```

-- tuple types: defined for each table

```
CREATE TYPE Name AS ( AttrName AttrType, ... )
```

-- enumerated type: specify elements and ordering

```
CREATE TYPE Name AS ENUM ( 'Label', ... )
```

## ❖ Examples of Defining Domains/Types

```
-- positive integers
CREATE DOMAIN PosInt AS integer CHECK (value > 0);

-- a UNSW course code
CREATE DOMAIN CourseCode AS char(8)
    CHECK (value ~ '[A-Z]{4}[0-9]{4}');

-- a UNSW student/staff ID
CREATE DOMAIN ZID AS integer
    CHECK (value between 1000000 and 9999999);

-- standard UNSW grades (FL, PS, CR, DN, HD)
CREATE DOMAIN Grade AS char(2)
    CHECK (value in ('FL', 'PS', 'CR', 'DN', 'HD'));

-- or
CREATE TYPE Grade AS ENUM ('FL', 'PS', 'CR', 'DN', 'HD');
```

## ❖ Tuple and Set Literals

---

Tuple and set constants are both written as:

$(\text{val}_1, \text{val}_2, \text{val}_3, \dots)$

The correct interpretation is worked out from the context.

Examples:

```
INSERT INTO Student(studeID, name, degree)
VALUES (2177364, 'Jack Smith', 'BSc')
      -- tuple literal
```

```
CONSTRAINT CHECK gender IN ('male', 'female', 'unspecified')
      -- set literal
```

Produced: 10 Feb 2021