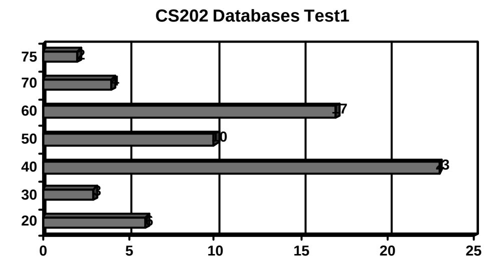
# TEST Review

# Test Results




N=65 DNW=5 Class average 52% 51% Pass rate

# Test Feedback

- General Weak Areas
  - Following instructions
  - Reading the questions
- DB Weaknesses
  - Rules for normalisation
  - Relational Algebra
  - Datatypes
- DB Strengths
  - SQL
  - Keys

# SQL Data Types

| Type Name | Aliases | Description |
|---|---|---|
| boolean | Bool, logical | Boolean (T/F) |
| bytea | | binary data |
| character | varying(n) varchar(n) | variable-length character string |
| character(n) | char(n) | fixed-length char string |
| date | | calendar date (year, month, day) |
| integer | int, int4 | signed four-byte integer |
| numeric [ (p, s) ] | decimal [(p,s)] | exact numeric with selectable precision |
| smallint | int2 | signed two-byte integer |
| text | | variable-length character string |
| serial | serial4 | autoincrementing four-byte integer |

See PostgreSQL User Guide Ch 5

# Esoteric Data Types

- box — rectangular box in 2D plane
- cidr — IP network address
- circle — circle in 2D plane
- inet — IP host address
- macaddr — MAC address
- polygon — closed geometric path in 2D plane

See PostgreSQL User Guide Ch 5

# SQL Compatibility

- The following generic types are generally supported across all SQL implementations
- *bit, bit varying, boolean, char, character varying, character, varchar, date, double precision, integer, interval, numeric, decimal, real, smallint, time, timestamp*

## Numeric Data Types

- Integer
  - store **whole** numbers, without fractional components
  - The type ***integer*** is the usual choice
  - best balance between range, storage size, and performance.
  - The ***smallint*** type is generally only used if disk space is at a premium.
  - The ***bigint*** type should only be used if the integer range is not sufficient

## Numeric Data Types

- Arbitrary Precision Numbers
  - The scale of a numeric is the count of decimal digits in the fractional part, to the right of the decimal point.
  - The precision of a numeric is the total count of significant digits in the whole number, that is, the number of digits to both sides of the decimal point.
  - So the number 23.5141 has a precision of 6 and a scale of 4. Integers can be considered to have a scale of zero.

## Numerical Data Types

- Numeric
  - The type ***numeric*** can store numbers with up to 1,000 digits
  - It is especially recommended for storing monetary amounts and other quantities where exactness is required.
  - ***numeric*** type is very slow compared to the floating-point types
  - Both the precision and the scale of the ***numeric*** type can be configured.

  `NUMERIC(precision, scale)`
  `NUMERIC(precision)` -- selects a scale of 0.

## Numerical Data Types

- Floating Point
  - Data types ***real*** and ***double*** precision are inexact, variable-precision numeric types.
  - Inexact means that some values cannot be converted exactly to the internal format and are stored as approximations, so that storing and printing back out a value may show slight discrepancies.
    - If you require exact storage and calculations (such as for monetary amounts), use the ***numeric*** type instead.
    - If you want to do complicated calculations with these types for anything important, you should evaluate the implementation carefully.
    - Comparing two floating-point values for equality may or may not work as expected

## Numerical Data Types

- Serials
  - Not a true type, but convenient for setting up identifier columns (similar to the AUTO_INCREMENT property supported by some other databases).

```
CREATE TABLE tablename (
    colname SERIAL
);
```

- This creates an integer column and arranged for its default values to be assigned from a sequence generator.
- A NOT NULL constraint is applied to ensure that a null value cannot be explicitly inserted
- In most cases you would also want to attach a UNIQUE or PRIMARY KEY constraint to prevent duplicate values from being inserted by accident, but this is not automatic.

## Character Data Types

- SQL defines two primary character types:
  - character varying(n)
  - character(n)
- Both types store strings up to *n* characters in length.
- An attempt to store a longer string into a column of these types will result in an error, unless the excess characters are all spaces, in which case the string will be truncated to the maximum length.
- If the string to be stored is shorter than the declared length, values of type character will be space-padded; values of type character varying will simply store the shorter string.

## Date/Time Data Types

| | |
|---|---|
| January 8, 1999 | unambiguous |
| 1999-01-08 | ISO-8601 format |
| 1/8/1999 ; 8/1/1999 | U.S. vs European mode |
| 1999.008 ; 99008 | year and day of year |
| 19990108 | ISO-8601 year, month, day |
| 990108 | ISO-8601 year, month, day |
| J2451187 | Julian day |
| January 8, 99 BC | year 99 before the Common Era |

## Boolean Data Types

- TRUE
- 't'
- 'true'
- 'y'
- 'yes'
- '1'

- FALSE
- 'f'
- 'false'
- 'n'
- 'no'
- '0'

## Normalisation and DB Design

- Should be part of the design process
- Entity Relation diagrams provide a high-level view
- Normalisation provides for a much lower-level view of the interaction between entities
- ER diagramming and Normalisation are closely related

## Stages of Normalisation

- 1NF – First Normal Form
- 2NF – Second Normal Form
- 3NF – Third Normal Form
- 4NF – Fourth Normal Form

## Conversion of DATA to 1NF

- Repeating groups to be eliminated
- Proper primary key must be selected/created
  - uniquely identifies entity rows
- Dependencies can be identified
  - Most desirable - those based on Primary Key
  - Less desirable
    - Partial – Based on part of Primary Key
    - Transitive – one non key attribute depends on another

## Conversion to 2NF

- Start with 1NF data
- Write each key component on a separate line
- Write original Key on the last line
- Each Key component should be used as a new table
- Write dependant attributes after the Key on each line

## Conversion to 3NF

- Start with 2NF data
- Create separate tables in order to remove transitive dependencies