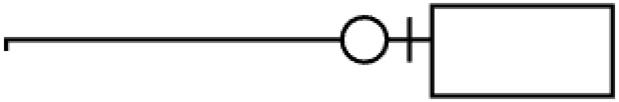
Continuous Assessment I

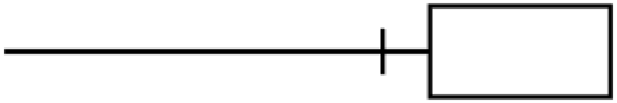
- **CA1 – 20%** must be submitted this **Friday (18/10/2019) before 5pm**
    - The descriptor was updated recently
    - Can be found here (https://moodle.cct.ie/mod/resource/view.php?id=49275)
- **Job Title** clarification
- A **relationship** can have **attributes**
    - These **must be converted into relations** (i.e. tables) for the **Relational Data Model (Logical)**.
- **Crow's Foot** should be used for the **Relational Data Model (Logical)**.
- Your **Logical Data Model** should include **primary** & **foreign** keys, **data types**, i.e. should be very close to a **Physical Data Model**

Logical/Physical Modelling

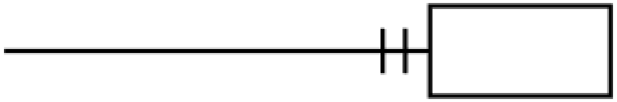
Crow’s Foot Notation:



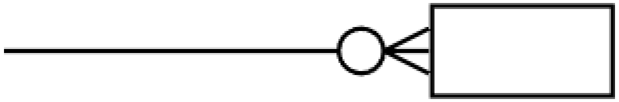
Zero or One
an optional relationship



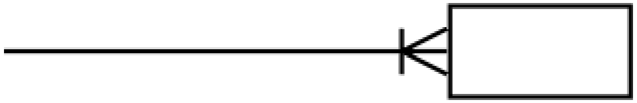
One



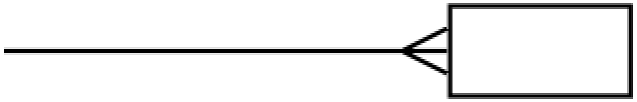
One and Only One



Zero or Many
an optional relationship



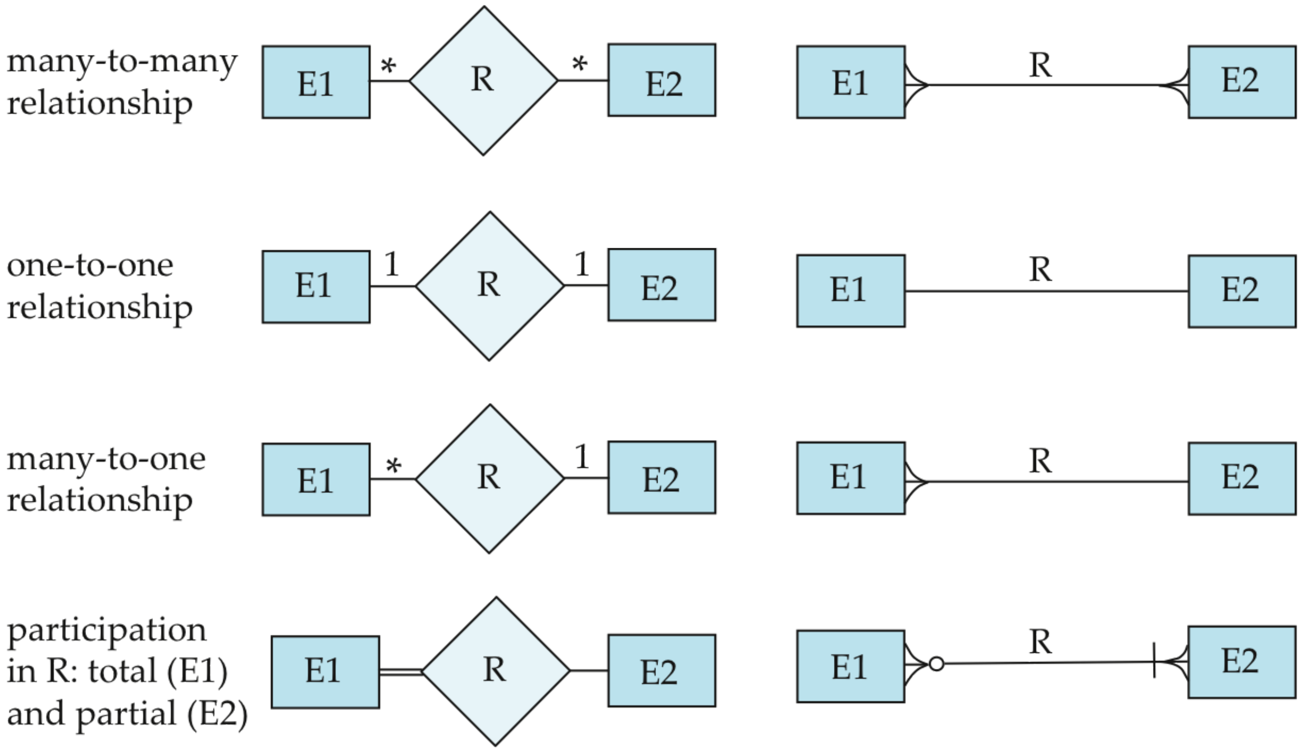
One or Many



Many

More info on **Crow's Foot notation** can be found **here (https://www.vertabelo.com/blog/crow-s-foot-notation/)**

Chen vs Crow's Foot



many-to-many relationship

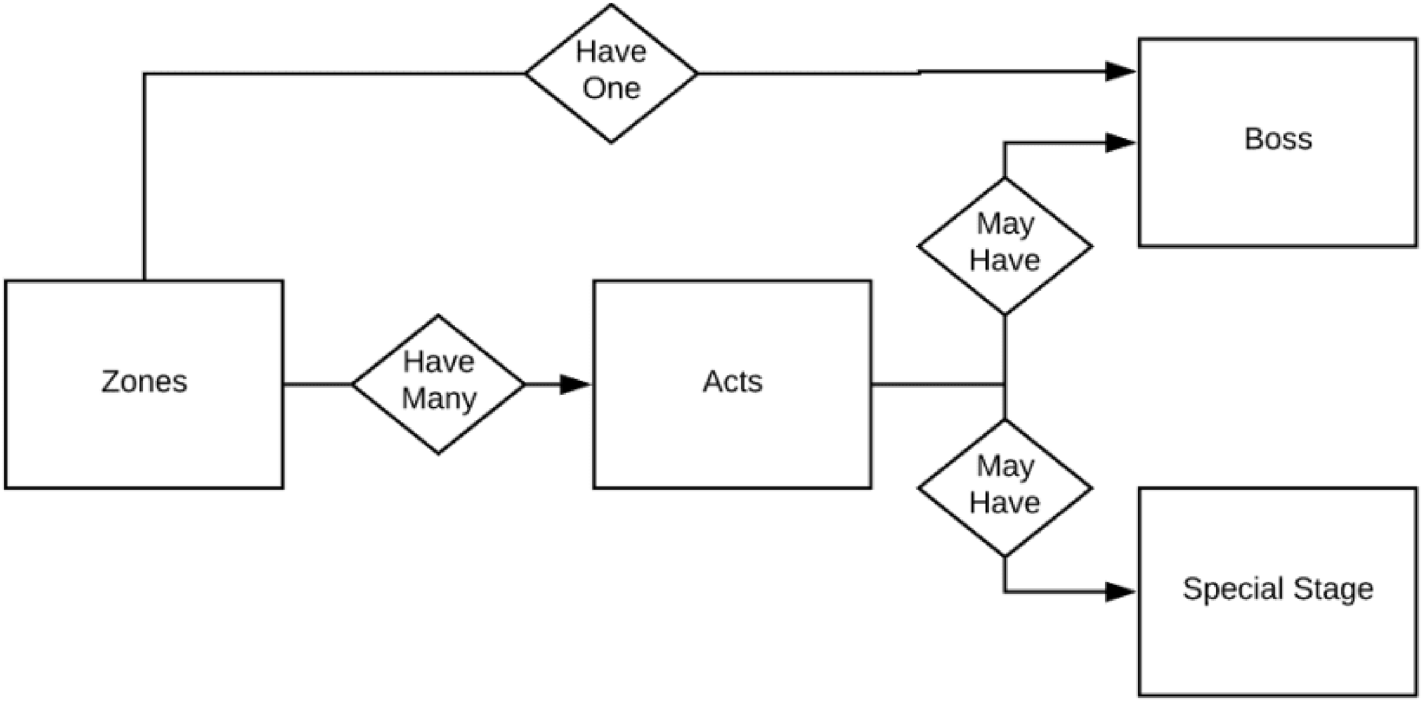
one-to-one relationship

many-to-one relationship

participation in R: total (E1) and partial (E2)

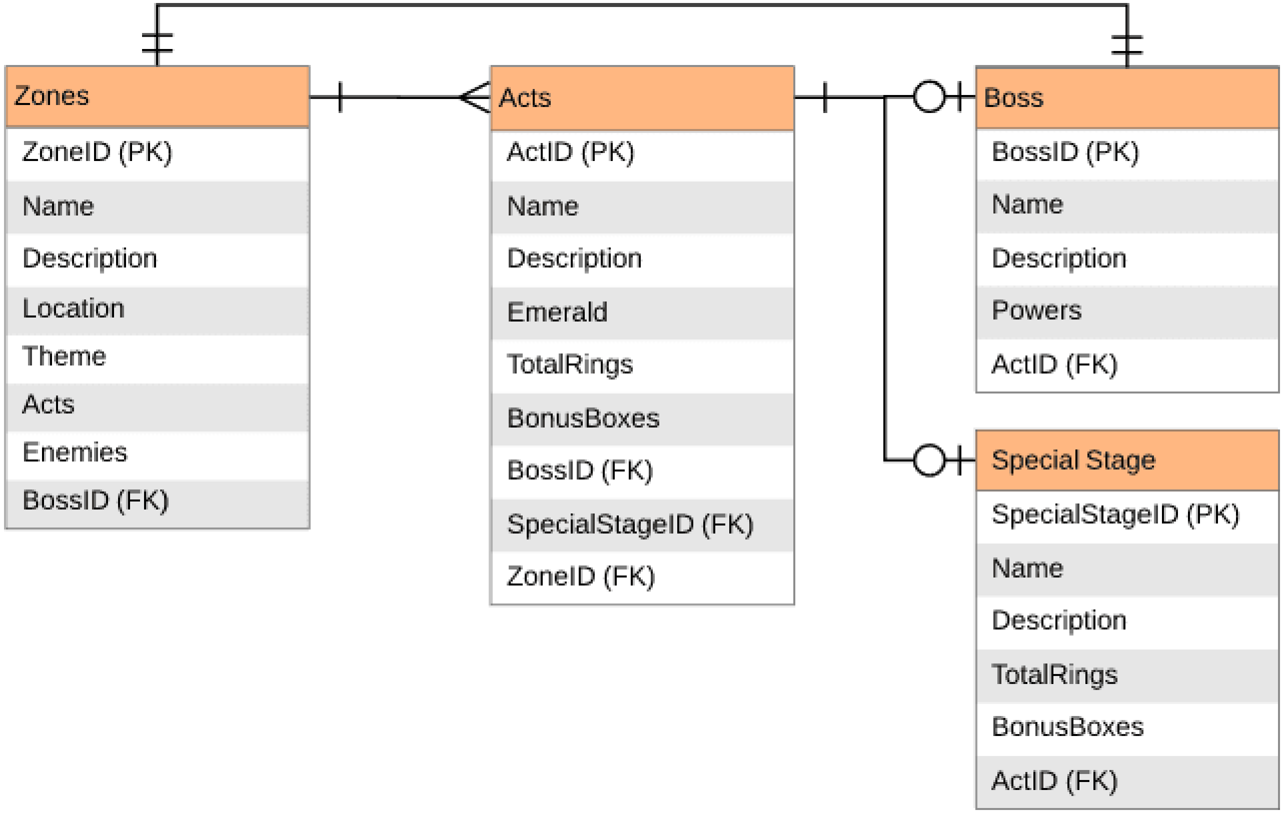
Example: Conceptual Model

- In **Sonic the Hedgehog 1** game, **Zones** can have many **Acts**, these levels may have **Special Stages**, may have **Bosses**, but there is only one **Boss** per **Zone**



- This example can be found here (https://dev.to/helenanders26/entity-relationship-diagrams-explained-by-sonic-the-hedgehog-1m68)

Example: Logical Model



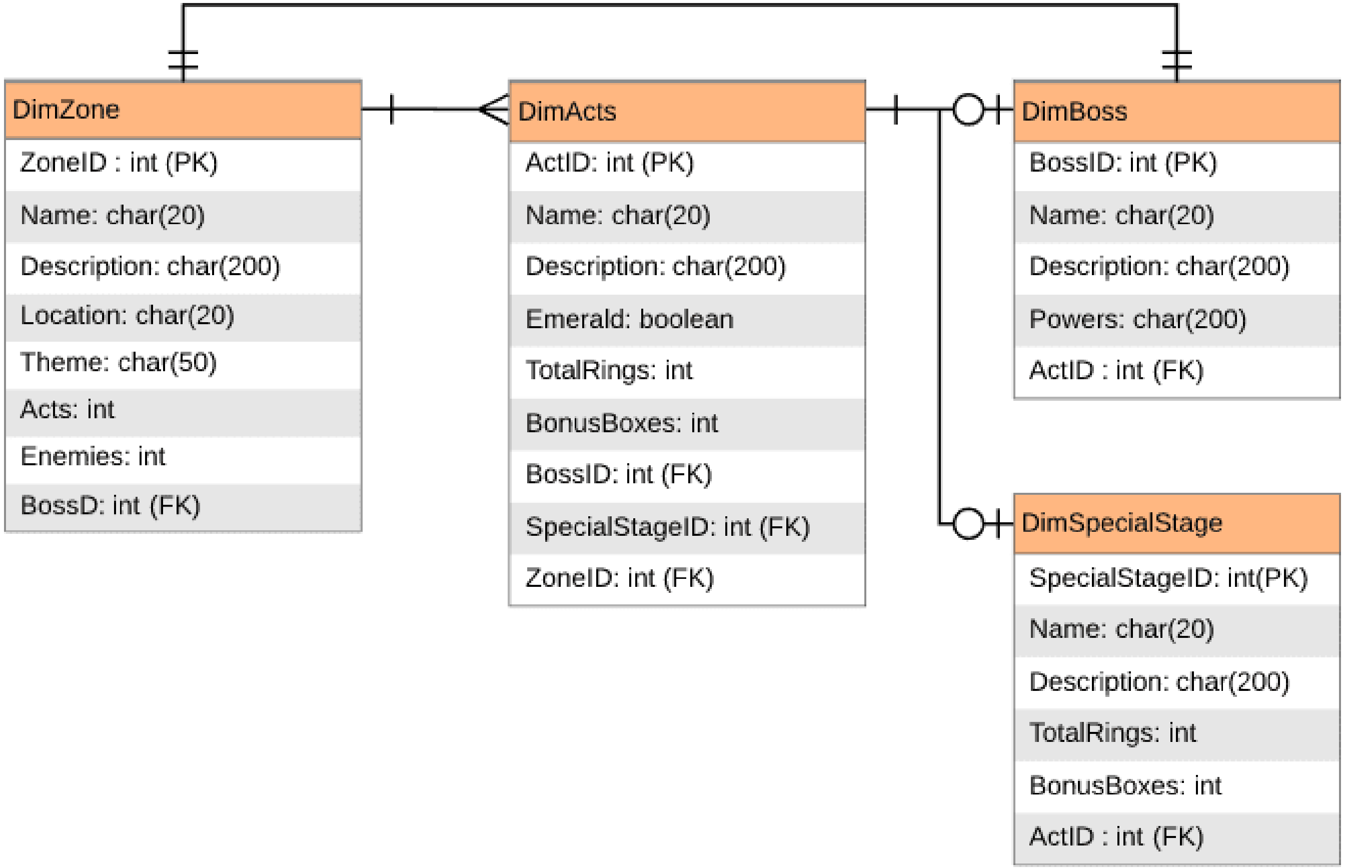
Zones
ZoneID (PK)
Name
Description
Location
Theme
Acts
Enemies
BossID (FK)

Acts
ActID (PK)
Name
Description
Emerald
TotalRings
BonusBoxes
BossID (FK)
SpecialStageID (FK)
ZoneID (FK)

Boss
BossID (PK)
Name
Description
Powers
ActID (FK)

Special Stage
SpecialStageID (PK)
Name
Description
TotalRings
BonusBoxes
ActID (FK)

Example: Physical Model



DimZone
ZoneID : int (PK)
Name: char(20)
Description: char(200)
Location: char(20)
Theme: char(50)
Acts: int
Enemies: int
BossD: int (FK)

DimActs
ActID: int (PK)
Name: char(20)
Description: char(200)
Emerald: boolean
TotalRings: int
BonusBoxes: int
BossID: int (FK)
SpecialStageID: int (FK)
ZoneID: int (FK)

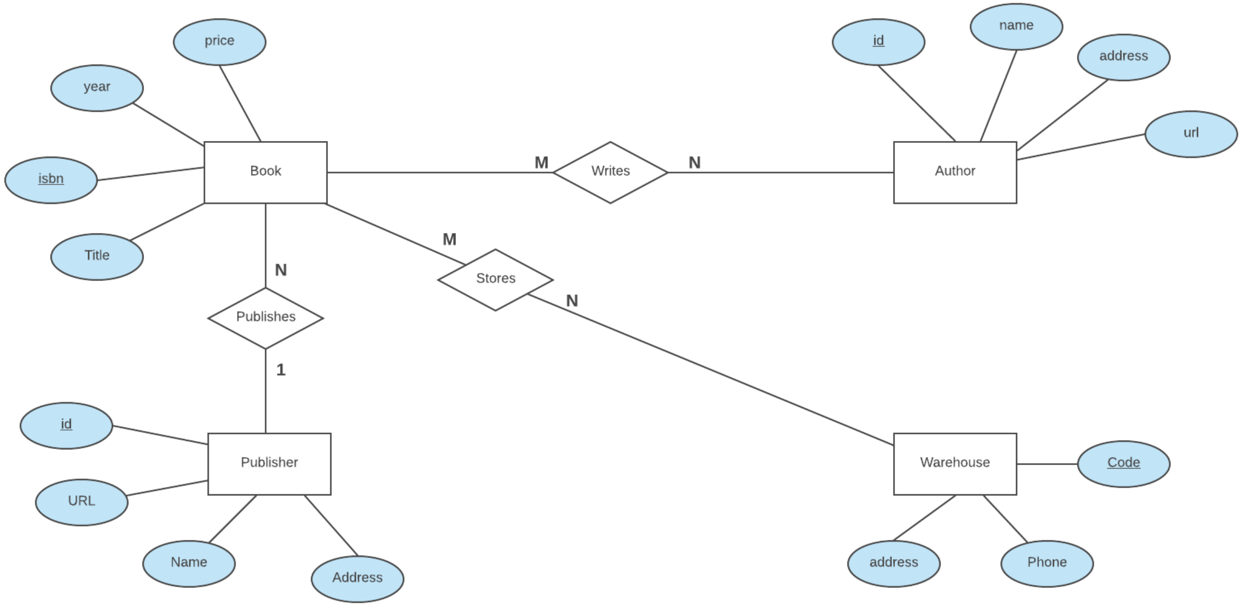
DimBoss
BossID: int (PK)
Name: char(20)
Description: char(200)
Powers: char(200)
ActID : int (FK)

DimSpecialStage
SpecialStageID: int(PK)
Name: char(20)
Description: char(200)
TotalRings: int
BonusBoxes: int
ActID : int (FK)

Exam Question – January Exam 2013

- **Design an ERD according to the provided specifications:**
    - The database must store **book**, **author**, **publisher** and **warehouse** information.
    - For every **book** you must capture the **title**, **isbn**, **year** and **price information**. The isbn value is unique for a book.
    - For every **author** you must store an **id**, **name**, **address** and the **URL** of their homepage.
    - Each **author** can write ***many*** **books**, and each **book** can have ***many*** **authors**, for example.
    - For every **publisher** you must store an **id**, **name**, **address**, **phone number** and an **URL** of their website.
    - **Books** are stored at ***several*** **warehouses**, each of which has a **code**, **address** and **phone number**.
    - A **book** has only **one publisher**.
    - The **warehouse** stocks ***many*** **different books**. A book may be stocked at multiple warehouses.
    - The database records the **number of copies** of a book stocked at various warehouses.

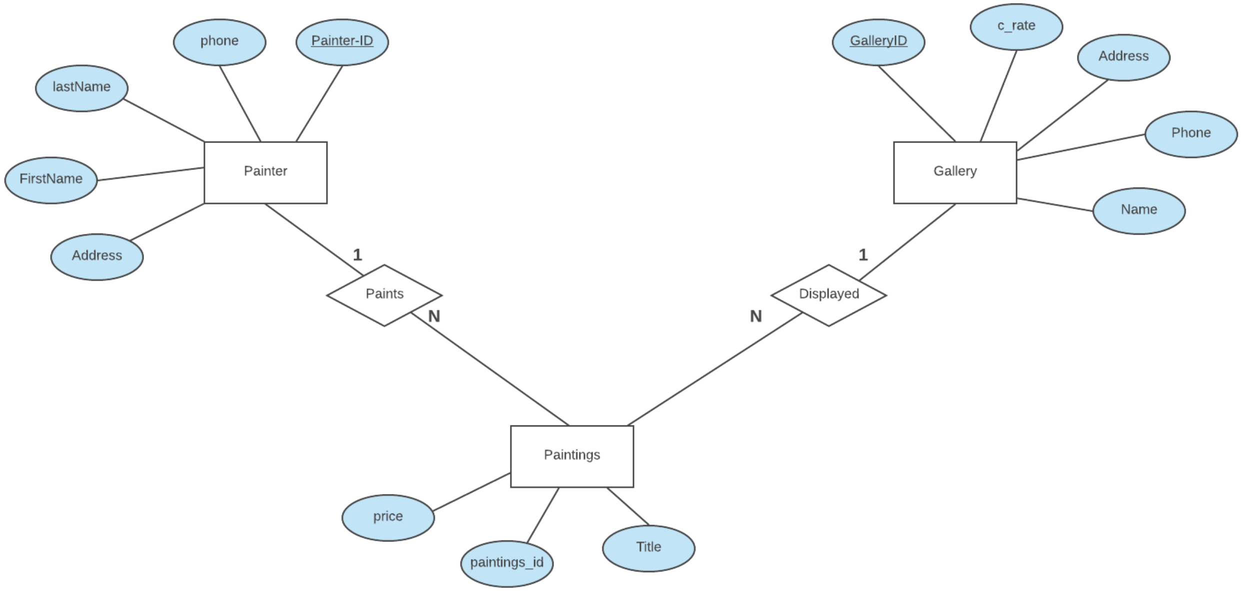
Exam Question – January Exam 2013



Exam Question – August Exam 2014

- **Design an ERD according to the provided specifications:**
    - **United Direct Artists (UDA)** is an insurance broker that specialise in insuring paintings for galleries. You are required to design a database for this company.
    - The database must store **painters**, **paintings** and **galleries** information.
    - Painters have a **unique number**, **name** and **phone number**
    - Paintings have **unique number**, **title** and **price**
    - Galleries have **unique number**, **owner**, **phone number**, **commission rate** and **address**
    - A painting is painted by a particular artist, and that **painting** is exhibited in **a particular gallery**.
    - A **gallery** can exhibit ***many*** **paintings**, but each **painting** can be exhibited in **only one gallery**.
    - Similarly, a **painting** is painted by **a single painter**, but each **painter** can paint ***many*** **paintings**.

Exam Question – August Exam 2014



Introduction to SQL

- Press `Space` to navigate through the slides
- Use `Shift+Space` to go back

Quick Recap

- **ERD Notations**
- **Components**
  - **Entity**
  - **Attribute**
  - **Relationship**
    - Cardinality
    - Degree
    - Participation constraints
    - Optionality
- **Chen Notation**
  - Limitations, Tips, Drawing, Naming, Refinment

Structured Query Language

- **Structured Query Language (SQL)** is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS)

- Ideally, database language should allow user to:

    - **create** the database and relation structures
    - perform **insertion**, **modification**, **deletion** of data from relations
    - perform simple and complex **queries**

- Must perform these tasks with minimal user effort and command structure/syntax must be **easy to learn**.

- It should be **portable** (use commands that allows one to move from one DBMS to another easily), have an easy to learn command structure and syntax

Structured Query Language

- **SQL** is a language with 2 major components:
  - A **DDL** for defining database structure.
  - A **DML** for retrieving and updating data.
- It is non-procedural – you specify what information you require, rather than how to get it
- SQL is free-format which means that parts of statements do not have to be typed at particular locations on the screen
- Consists of standard English words:

```
SELECT staffNo, lName, salary FROM Staff WHERE salary > 10000;
```

- An ISO standard now exists for SQL, making it both the formal and de facto standard language for relational databases

SQL Syntax

- SQL statement consists of reserved words and user-defined words.
- **Reserved words** are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
    - Certain keywords, such as `SELECT`, `DELETE` are reserved in RDMBS
- **User-defined** words are made up by user and represent names of various database objects such as relations, columns, views. Most components of an SQL statement are case insensitive, except for literal character data.
- **Many dialects of SQL require the use of a statement terminator to mark the end of each SQL statement (usually the semicolon `;` is used)**

SQL Syntax

- SQL is more readable with indentation and lineation:
  - Each clause should begin on a new line.
  - Start of a clause should line up with start of other clauses.
  - If clause has several parts, should each appear on a separate line and be indented under start of clause. E.g

```
SELECT *
FROM Branch
WHERE city = 'London';
```

SQL Syntax

- Literals are constants used in SQL statements.
- All **non-numeric literals** must be enclosed in single quotes (e.g. `'London'`).
- All **numeric literals** must NOT be enclosed in quotes (e.g. `650.00`).

SQL Syntax

For numeric data types (e.g. INT, DOUBLE), we do not need to put quotes around the value:

```
SELECT * FROM houses WHERE price > 300000;
```

- For other data types (e.g. VARCHAR, DATE, TEXT), it is required to use single quotes to quote the value:

```
SELECT * FROM houses WHERE town = 'Dublin'
SELECT * FROM people WHERE dob < '1980-01-01'
```

- Some RDBMSs (including MySQL) will also allow you to use double quotes but this is not recommended as it is not part of the ANSI SQL standard and is not guaranteed to work with all systems.

Select Statement

```
SELECT [DISTINCT | ALL]
    {* | [fieldExpression [AS newName]}
FROM         tableName [alias]
[WHERE     condition]
[GROUP BY   columnList]
[HAVING   condition]
[ORDER BY   columnList]
```

Select Statement

```
SELECT
```

is the SQL keyword that lets the database know that you want to retrieve data.

```
[DISTINCT | ALL]
```

are optional keywords that can be used to fine tune the results returned from the SQL SELECT statement. If nothing is specified then ALL is assumed as the default.

```
{* | [fieldExpression [AS newName]}
```

at least one part must be specified, `*` selected all the fields from the specified table name, `fieldExpression` performs some computations on the specified fields such as adding numbers or putting together two string fields into one.

Select Statement

```
FROM tableName
```

is mandatory and must contain at least one table, multiple tables must be separated using commas or joined using the `JOIN` keyword.

```
WHERE
```

condition is optional, it can be used to specify criteria in the result set returned from the query.

```
GROUP BY
```

is used to put together records that have the same field values.

```
HAVING
```

condition is used to specify criteria when working using the `GROUP BY` keyword.

```
ORDER BY
```

is used to specify the sort order of the result set.

Basic Query Structure

- A typical SQL query has the form:
    - **select** A_1 , A_2 , ..., A_n
    - **from** R_1 ,R_2 ,...,R_m
    - **where** P
- A_1 represents an **attribute**
- R_1 represents a **relation**
- P is a predicate (**condition**)
- **The result of an SQL query is a relation (table).**

Example 1: All Columns, All Rows

- List full details of all staff.

```
SELECT staffNo, fName, lName, address, position, sex, DOB, salary, branchN
o FROM Staff;
```

- Can use * as an abbreviation for ‘all columns’: (quick way/shortcut)

```
SELECT * FROM Staff;
```

Example 1: All Columns, All Rows

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-601	2000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Example 2: Specific Columns, All Rows

- Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary FROM Staff;
```

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Example 3: Use of Distinct

- List the property numbers of all properties that have been viewed.

```
SELECT propertyNo FROM Viewing;
```

propertyNo
PA14
PG4
PG4
PA14
PG36

Example 3: Use of Distinct

- Use DISTINCT to **eliminate duplicates**:

```
SELECT DISTINCT propertyNo FROM Viewing;
```

propertyNo
PA14
PG4
PG36

Example 4: Calculated Fields

- Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.

```sql
SELECT staffNo, fName, lName, salary
FROM Staff;
```

- To name column, use `AS` clause:

```
SELECT staffNo, fName, lName, salary/12 AS monthlySalary
FROM Staff;
```

staffNo	fName	lName	monthlySalary
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

Five Basic Search Conditions

- **Comparison**
  - Compare the value of one expression to the value of another expression.
- **Range**
  - Test whether the value of an expression falls within a specified range of values.
- **Set membership**
  - Test whether the value of an expression equals one of a set of values.
- **Pattern match**
  - Test whether a string matches a specified pattern.
- **Null**
  - Test whether a column has a null (unknown) value.

Comparison Search Condition

- List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary FROM Staff
WHERE salary > 10000;
```

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SG5	Susan	Brand	24000.00

Comparison Search Condition

- In SQL, the following simple comparison operators are available:
    - = equals
    - <> is not equal to (ISO standard)
    - `!`= is not equal to (allowed in some dialects)
    - < is less than
    - <= is less than or equal to
    - > is greater than
    - >= is greater than or equal to

Comparison Search Condition

- List addresses of all branch offices in London or Glasgow.

```sql
SELECT *
FROM Branch
WHERE city = 'London' OR city = 'Glasgow';
```

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Range Search Condition

- List all staff with a salary between 20,000 and 30,000.

```sql
SELECT staffNo, fName, lName, position, salary FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

- BETWEEN test includes the endpoints of range.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

Range Search Condition

- Also a negated version `NOT BETWEEN`.
- `BETWEEN` does not add much to SQL's expressive power.
- You could also write:

```
SELECT staffNo, fName, lName, position, salary FROM Staff
WHERE salary >=20000 AND salary <= 30000;
```

- Useful, though, for a range of values.
- `BETWEEN` test includes the endpoints of range.

Set Membership

- List all managers and supervisors:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

- There is a negated version as well – `NOT IN`

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Set Membership

- Could have expressed this as:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position='Manager' OR position='Supervisor';
```

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Pattern Matching

- Find all owners with the string 'Glasgow' in their address.

```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE ‘%Glasgow%’;
```

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Pattern Matching

- SQL has two special pattern matching symbols:
  - `%` – sequence of zero or more characters;
  - _ (underscore) – any single character.
  - `LIKE '%Glasgow%'` – means a sequence of characters of any length containing 'Glasgow'.

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Null

- List details of all viewings on property PG4 where a comment has not been supplied.
- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using special keyword `IS NULL`:

```
SELECT clientNo, viewDate FROM Viewing
WHERE propertyNo = ‘PG4’ AND comment IS NULL;
```

Summary

- SQL is not case sensitive so commands can be upper or lower case
- Convention, however, has it that we write SQL operators in upper case and the operands in lower case, e.g.
- `SELECT name FROM employee;`
- Each SQL command ends in a semicolon

Tutorial 💻

Practice SQL with SQLite and Jupyter Notebook (GitHub) (https://github.com/royalosyin/Practice-SQL-with-SQLite-and-Jupyter-Notebook)

- Intoduction (tutorial/ex00-Introduction.ipynb)
- Quick Start (tutorial/ex01-Quick%20Start.ipynb)
- Quary Table Information (tutorial/ex02-Query%20Table%20Information.ipynb)
- Retrieving Data with SELECT (tutorial/ex03-Retrieving%20Data%20with%20SELECT
- Constrain the Number of Rows Returned by a SELECT Query (tutorial/ex04-Constrain%20the%20Number%20of%20Rows%20Returned%20by%20a%20SELECT
- Filtering a Query with WHERE (tutorial/ex05-Filtering%20a%spo20Query%20with%
- Doing Math Across Table Columns (tutorial/ex06-Doing%20Math%20Across%20Table%20Columns.ipynb)
- Aggregating data with GROUP BY and ORDER BY (tutorial/ex07-Aggregating%20data%20with%20GROUP%20BY%20and%20ORDER%20BY.ipynb)
- Dealing with NULL Values (tutorial/ex11-Dealing%20with%20NULL%20Values.ipynb