COMP160: Software Engineering

Databases and SQL

Learning objectives

- By the end of this section, you should understand
  - The concepts of databases, tables, columns, rows, primary keys and relations
  - The analogy and differences between OOP and relational databases
  - How to use web resources to start exploring SQL query syntax

Database basics

- A database consists of several **tables**
  - E.g. Students, courses, modules
- A table has several **columns**
  - E.g. Name, student number, email address
- A table also has several **rows**
  - E.g. A row in the Students table holds the information about one student
- The tables and columns are the **schema** of the database

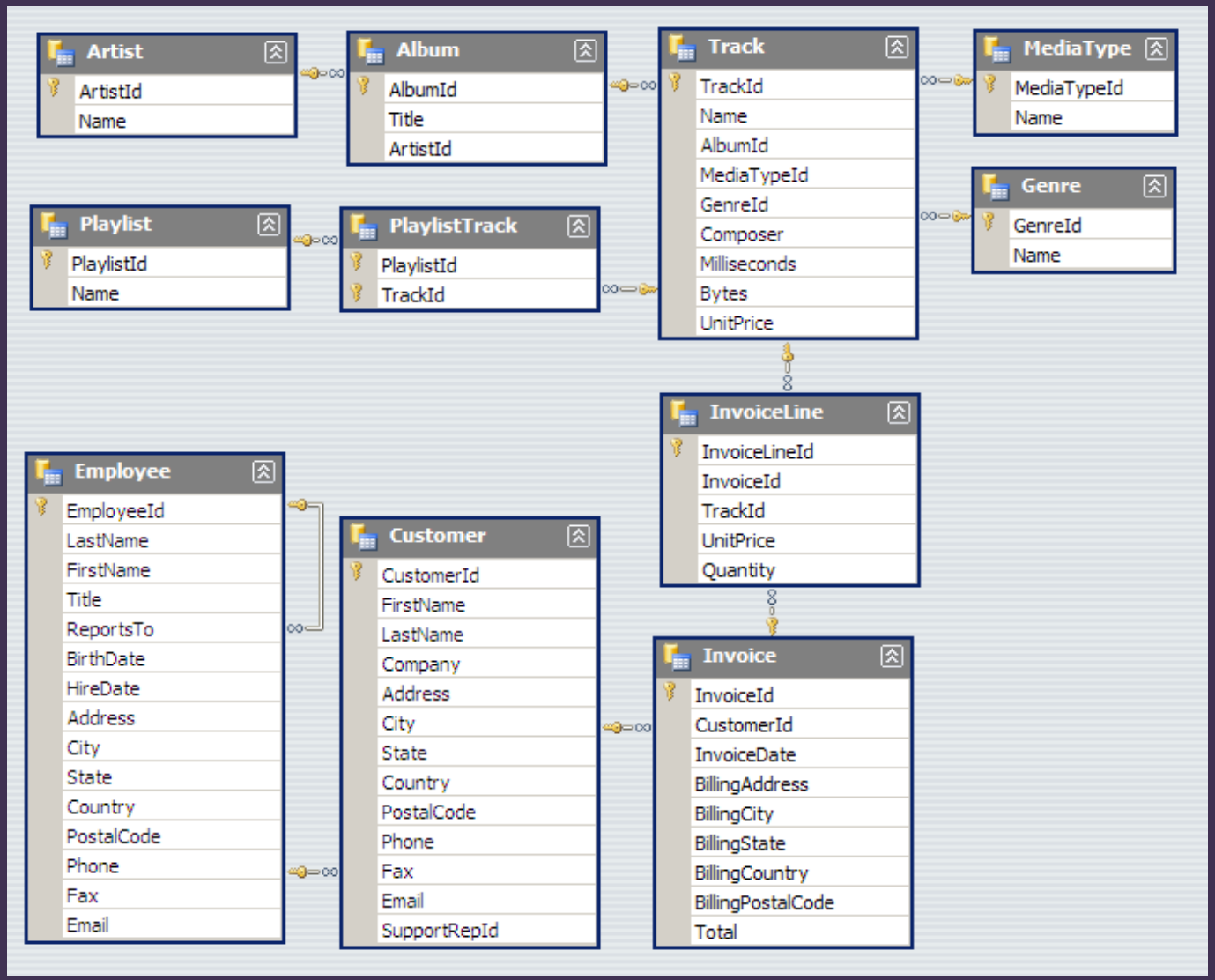
Relational databases

- Database columns can refer to other tables
- Every table has a **primary key** column, which is unique for every row
  - E.g. Student ID number, course code
- Tables can have columns which reference a **foreign key** – the primary key of another table
  - E.g. Which course a student is enrolled on

FALMOUTH UNIVERSITY

Relational database example

http://chinookdatabase.codeplex.com/



"One to many" relationship
← Primary key
Foreign key →

Socrative room code:
6E8NSW3IN

OOP and databases

- Not the same, but there are analogies

Object Oriented Programming	Relational databases
Module or program	Database
Class	Table
Field	Column
Method	*No direct analogy*
Instance	Row
Reference	Primary key
Field holding a reference to an instance of another class	Foreign key

Structured Query Language

- Language for writing database **queries**
- Examples:

```
SELECT FirstName, LastName FROM Students
  WHERE Course = 'BSCCOMP'
INSERT INTO Students (StudentId,
  FirstName, LastName, Course) VALUES
  (123456, 'John', 'Smith', 'BSCCOMP')
```

Activity

Read

https://www.w3schools.com/sql/sql_intro.asp

(30 minutes)

Activity

http://sqlzoo.net

(45 minutes)

Game Platform Studies

ENTITY-RELATIONSHIP DIAGRAMS

Original Slides: Mudasir Qazi

Definition

- The Relational Database Model forms on the basis of an ERD. The ERD represents the conceptual database as viewed by users.

- ERDs depict the database's main components: *entities*, *attributes*, and *relationships*.

Basic Components of ERD

- Entities:
  - An entity is an object of interest to the end user.
  - At the ER modeling level, an entity actually refers to the entity set and not to a single instance of an entity. In other words, the word entity in the ERM corresponds to a table—not to a row—in the relational environment.

Basic Components of ERD

- Attributes:
  - Attributes are characteristics of entities. For example, firstname, lastname, age, etc.
  - Attributes always belong to an entity.

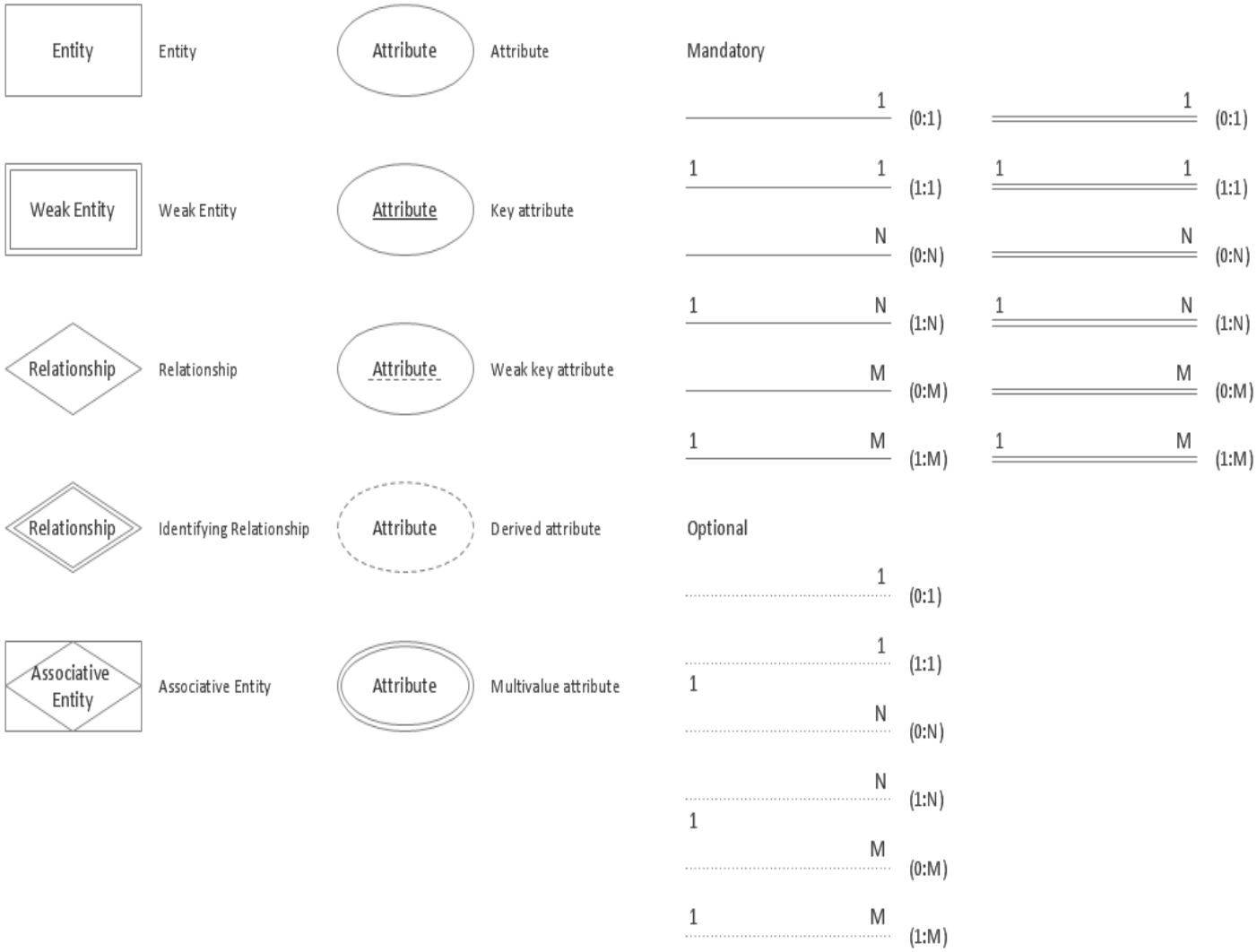
Basic Components of ERD

- Relationships:
  - A relationship is an association between entities. The entities that participate in a relationship are also known as *participants*, and each relationship is identified by an active and passive verb that describes the relationship.
  - Relationships between entities always operate in both directions:
    - A CUSTOMER may generate many INVOICEs.
    - Each INVOICE is generated by one CUSTOMER.

ERD Representations

- There are three main notations to represent and ERD
  - The *Chen notation* favors conceptual modeling.
  - The *Crow's Foot notation* favors a more implementation-oriented approach.
  - The *UML notation* can be used for both conceptual and implementation modeling.
- Because of its implementation emphasis, the Crow's Foot notation can represent only what could be implemented.

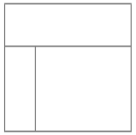
Chen Notation Symbols



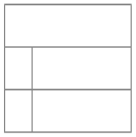
Crows Foot Notation Symbols

Entity
(with no attributes)

Entity
(with attributes field)



Entity
(attributes field with columns)



Entity
(attributes field with columns and variable number of rows)

Relationships
(Cardinality and Modality)

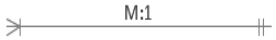
Zero or More

One or More

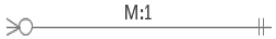
One and only One

Zero or One

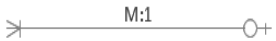
Many - to - One



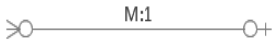
a one through many notation on one side of a relationship and a one and only one on the other



a zero through many notation on one side of a relationship and a one and only one on the other



a one through many notation on one side of a relationship and a zero or one notation on the other



a zero through many notation on one side of a relationship and a zero or one notation on the other

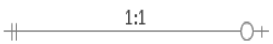
Many-to-Many

a zero through many on both sides of a relationship

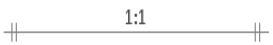
a one through many on both sides of a relationship

a zero through many on one side and a one through many on the other

Many-to-Many



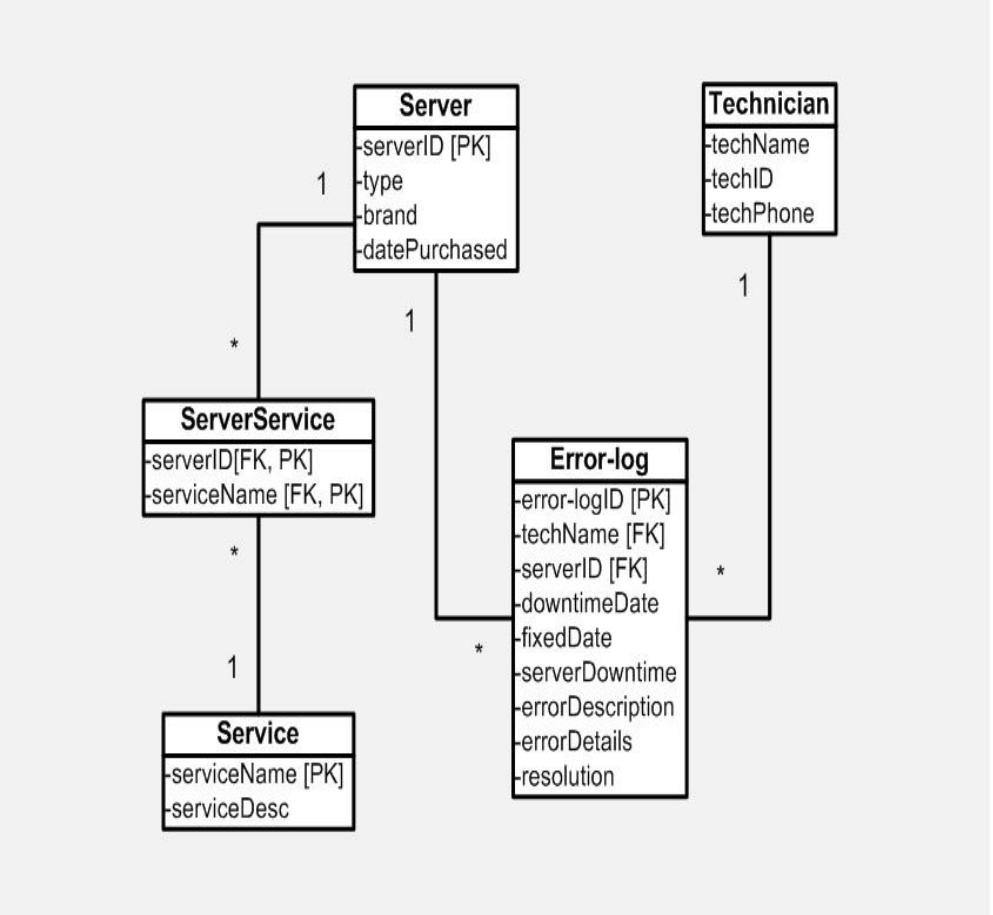
a one and only one notation on one side of a relationship and a zero or one on the other



a one and only one notation on both sides

UML Notation Symbols

Primary Keys are followed by [PK]
Foreign Keys are followed by [FK]
Relationships:
Cardinality 1 for 'One'.
Cardinality * for 'Many'.



Advantages

- Advantages:
  1. ERD tells us that how many tables you need and what would be the relationship between them (you also have to do Normalization to know finally how many tables would be in your database but still first step is ERD).
  2. ERD is simple and understandable representation of a database. It helps a lot to understand the whole database.

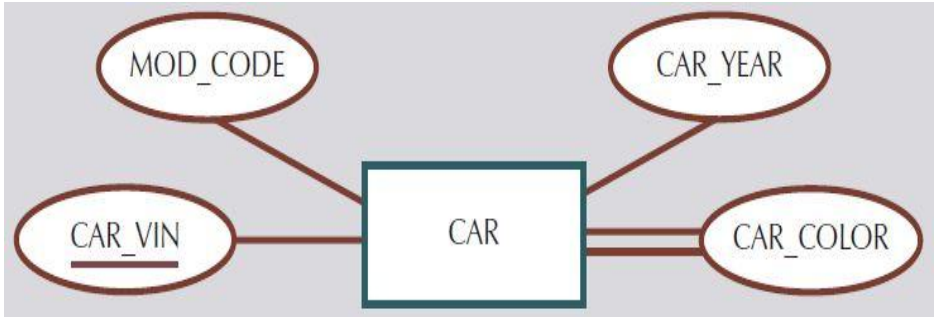
Usage

- Usage:
  1. An ERD leads to ERM, means when ever you need to build a database with tables, firstly, you need to create an ERD.
  2. Crow's foot notation is used most of all because its easy to understand in implementation point of view.

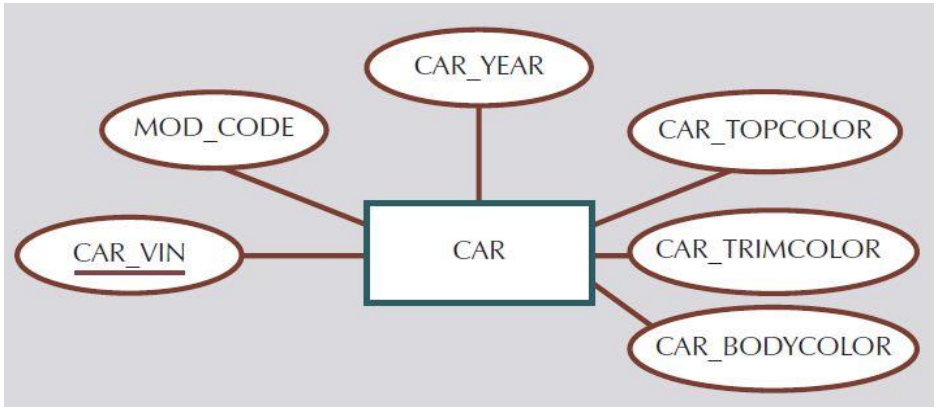
How to Solve Multi-Valued attributes (1)

- Although the conceptual model can handle M:N relationships and multivalued attributes, you should not implement them in the RDBMS.
- The relational table, each column/row intersection represents a single data value. So if multivalued attributes exist, the designer must decide on one of two possible courses of action:
  1. Within the original entity, create several new attributes, one for each of the original multivalued attribute's components.
  2. Create a new entity composed of the original multivalued attribute's components.

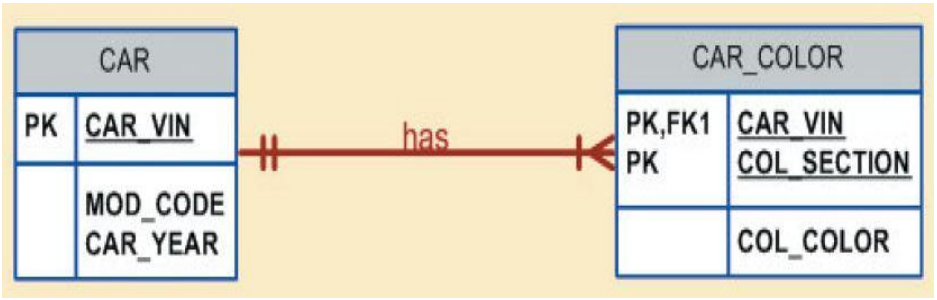
How to Solve Multi-Valued attributes (2)



Entity CAR contains a multivalued attribute CAR_COLOR.



Solution 1: (Not good)
Solution to Multi-Valued attribute by adding new attributes to CAR entity.



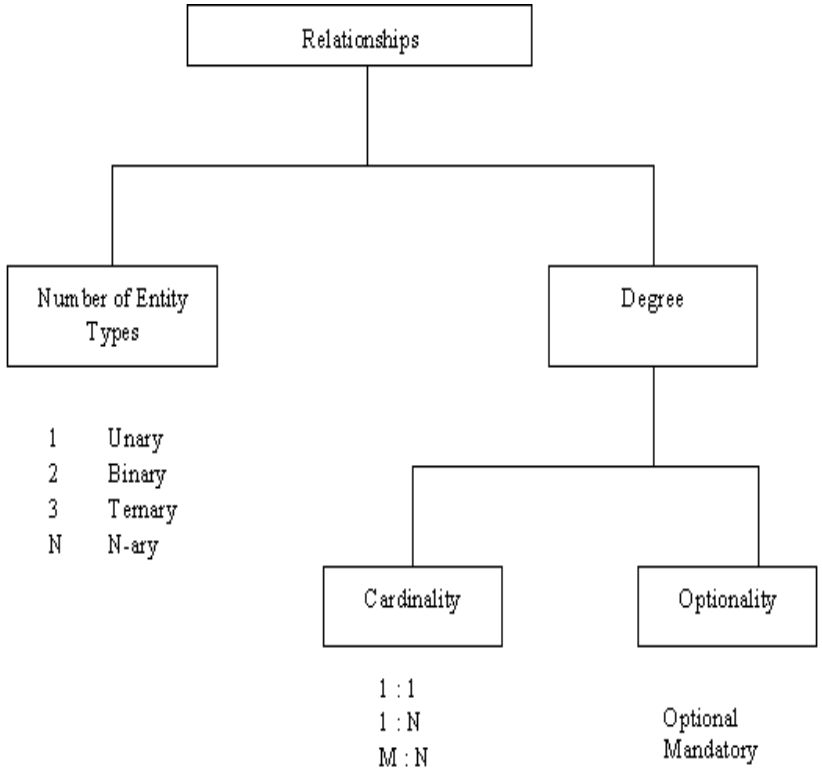
Solution 2: (Best)
Solution to Multi-Valued attribute by adding new entity with 1:M relation to CAR entity.

Relationships

- Ways of Classifying Relationships Types

  A relationship type can be classified by the *number of entity types involved*, and by the *degree* of the relationship type.

  Following is a brief picture showing all types of relationships.



Connectivity and Cardinalities

- Connectivity:

  The term *connectivity* is used to describe the relationship classification. Just to show the relation existence and type of relation between entities.

Connectivity and Cardinalities

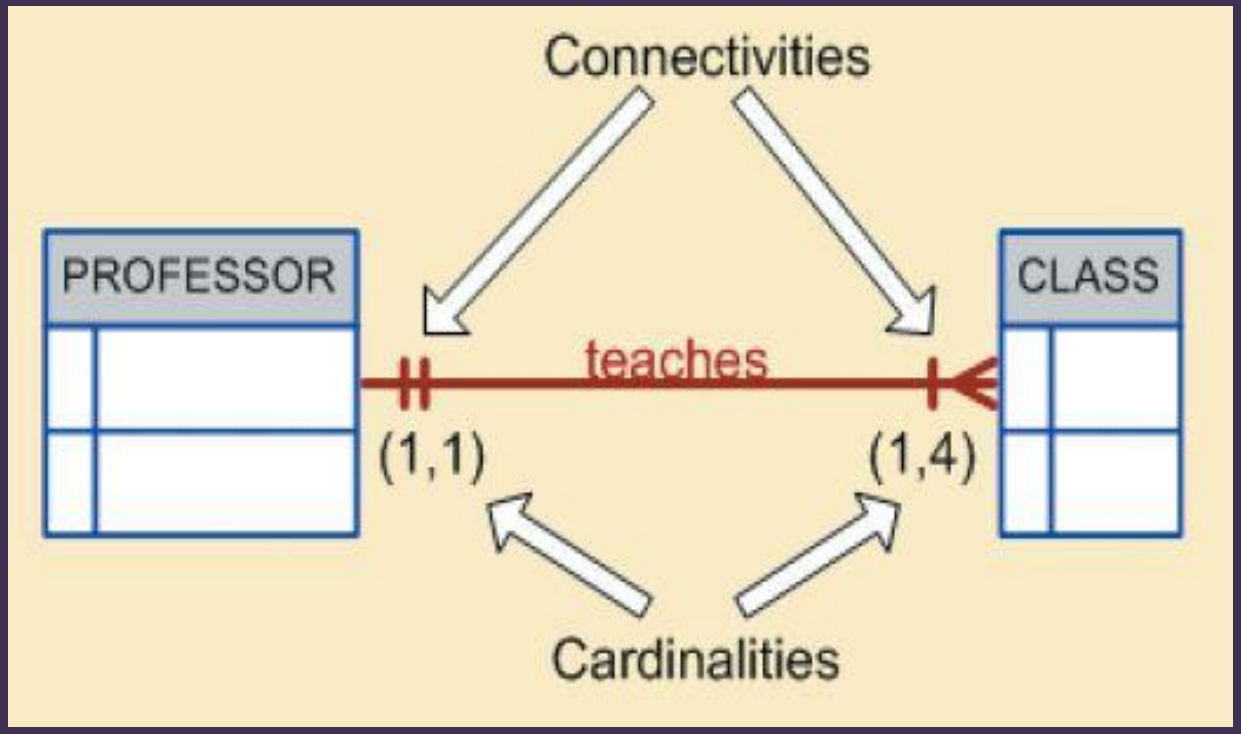
- Cardinalities:

  Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In the ERD, cardinality is indicated by placing the appropriate numbers beside the entities, using the format (min , max).

  1. Cardinality / mandatory:
     maximum cardinality.
  2. Modality / optional:
     minimum cardinality or optionality.

Connectivity and Cardinalities

- One PROFESSOR can teach one (min 1) or more (max 4) CLASSes but each single (min 1) CLASS can be taught by one (max 1) PROFESSOR at time.



Software Engineering

DATABASES IN SQLITE & PYTHON

Original Slides: Mudasir Qazi

Databases in Python

- Python supports several database systems
- Most use a standard API so can be interchanged relatively easily
  - https://www.python.org/dev/peps/pep-0249/
- We will use SQLite (for now)

- Setting up a SQLite database as a data source in PyCharm (this enables autocompletion of table and field names when writing SQL queries)
  - View -> Tool windows -> Database
  - + -> Data Source -> SQLite -> Xerial
    - (What's the difference between Xerial and Zentus? I don't know, but Xerial seems to work well enough)
  - Click "Download" at the bottom (one-time setup)
  - Choose the database file
  - Click OK

A simple example

```
import sqlite3

database = sqlite3.connect("Chinook_Sqlite.sqlite")
cursor = database.cursor()

cursor.execute("""SELECT FirstName, LastName FROM Customer""")

for row in cursor:
    print row
```

What's wrong with this code?

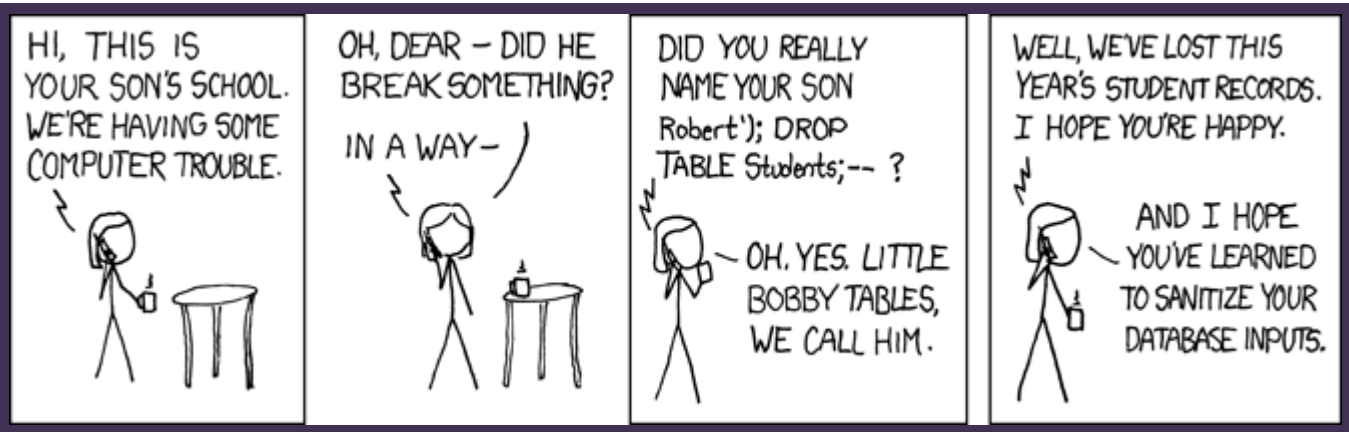
```
import sqlite3

database = sqlite3.connect("Chinook_Sqlite.sqlite")
cursor = database.cursor()

last_name = raw_input("Enter last name: ")

cursor.execute("""SELECT CustomerId, FirstName, LastName FROM Customer
                  WHERE LastName = '""" + last_name + "'")

for row in cursor:
    print row
```



Avoiding SQL injection

```
import sqlite3

database = sqlite3.connect("Chinook_Sqlite.sqlite")
cursor = database.cursor()

last_name = raw_input("Enter last name: ")

query_params = (last_name,)   # tuple of length 1

cursor.execute("""SELECT CustomerId, FirstName, LastName FROM Customer
                  WHERE LastName = ?""", query_params)

for row in cursor:
    print row
```

Accessing columns by name

```
import sqlite3

database = sqlite3.connect("Chinook_Sqlite.sqlite")
database.row_factory = sqlite3.Row
cursor = database.cursor()

cursor.execute("""SELECT Artist.Name AS ArtistName, Album.Title
                  FROM Album INNER JOIN Artist ON Artist.ArtistId = Album.ArtistId""")

for row in cursor:
    print row["Title"], "by", row["ArtistName"]
```

Activity

In your COMP130 groups:

- Design and implement a high score table for your game
  - Add scores
  - Update scores
- Use Python
- (45 minutes)

Game Platform Studies

DATABASE NORMALISATION

Normalization

- Main objective in developing a logical data model for relational database systems is to create an accurate representation of the data, its relationships, and constraints.

- To achieve this objective, must identify a suitable set of relations.

Normalization

- Four most commonly used normal forms are first (1NF), second (2NF) and third (3NF) normal forms, and Boyce–Codd normal form (BCNF).
- Based on functional dependencies among the attributes of a relation.
- A relation can be normalized to a specific form to prevent possible occurrence of update anomalies.

Data Redundancy

- Major aim of relational database design is to group attributes into relations to minimize data redundancy and reduce file storage space required by base relations.

- Problems associated with data redundancy are illustrated by comparing the following Staff and Branch relations with the StaffBranch relation.

Data Redundancy

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Data Redundancy

- StaffBranch relation has redundant data: details of a branch are repeated for every member of staff.

- In contrast, branch information appears only once for each branch in Branch relation and only branchNo is repeated in Staff relation, to represent where each member of staff works.

Update Anomalies

- Relations that contain redundant information may potentially suffer from update anomalies.

- Types of update anomalies include:
  - Insertion
  - Deletion
  - Modification.

Properties of Decomposition

- Two important properties of decomposition:

  - ***Lossless-join property*** enables us to find any instance of original relation from corresponding instances in the smaller relations.

  - ***Dependency preservation property*** enables us to enforce a constraint on original relation by enforcing some constraint on each of the smaller relations.

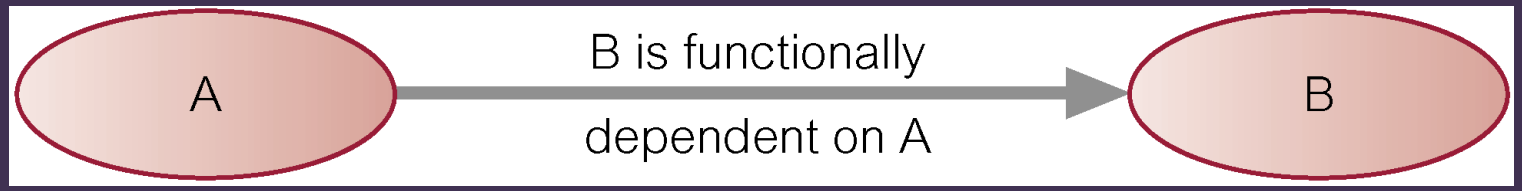
Functional Dependency

- Main concept associated with normalization.

- Functional Dependency
  - Describes relationship between attributes in a relation.
  - If A and B are attributes of relation R, B is functionally dependent on A (denoted A ⮝ B), if each value of A in R is associated with exactly one value of B in R.

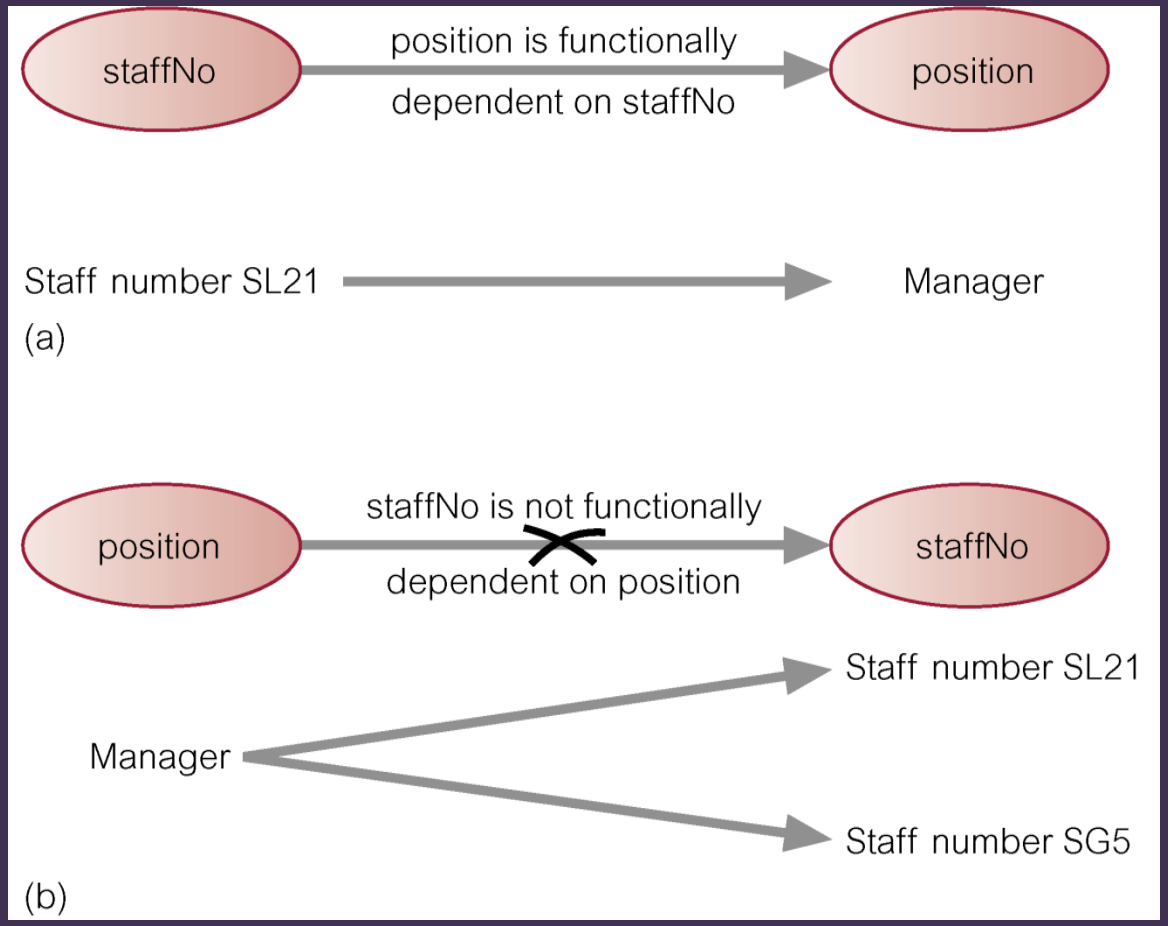
Functional Dependency

- Property of the meaning (or semantics) of the attributes in a relation.
- Diagrammatic representation:



  - *Determinant* of a functional dependency refers to attribute or group of attributes on left-hand side of the arrow.

Example - Functional Dependency



Functional Dependency

- Main characteristics of functional dependencies used in normalization:
  - have a 1:1 relationship between attribute(s) on left and right-hand side of a dependency;
  - hold for all time;
  - are nontrivial.

Functional Dependency

- Complete set of functional dependencies for a given relation can be very large.
- Important to find an approach that can reduce set to a manageable size.
- Need to identify set of functional dependencies (X) for a relation that is smaller than complete set of functional dependencies (Y) for that relation and has property that every functional dependency in Y is implied by functional dependencies in X.

Functional Dependency

- Set of all functional dependencies implied by a given set of functional dependencies X called closure of X (written X^+).

- Set of inference rules, called Armstrong's axioms, specifies how new functional dependencies can be inferred from given ones.

Functional Dependency

- Let A, B, and C be subsets of the attributes of relation R. Armstrong's axioms are as follows:
- 1. Reflexivity
  - If B is a subset of A, then $A \rightarrow B$
- 2. Augmentation
  - If $A \rightarrow B$, then $A,C \rightarrow B,C$
- 3. Transitivity
  - If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

The Process of Normalization

- Formal technique for analyzing a relation based on its primary key and functional dependencies between its attributes.
- Often executed as a series of steps. Each step corresponds to a specific normal form, which has known properties.
- As normalization proceeds, relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies.

Unnormalized Form (UNF)

- A table that contains one or more repeating groups.

- To create an unnormalized table:
  - transform data from information source (e.g. form) into table format with columns and rows.

First Normal Form (1NF)

- A relation in which intersection of each row and column contains one and only one value.

UNF to 1NF

- Nominate an attribute or group of attributes to act as the key for the unnormalized table.

- Identify repeating group(s) in unnormalized table which repeats for the key attribute(s).

UNF to 1NF

- Remove repeating group by:
  - entering appropriate data into the empty columns of rows containing repeating data ('flattening' the table).

  Or by
  - placing repeating data along with copy of the original key attribute(s) into a separate relation.

Second Normal Form (2NF)

- Based on concept of full functional dependency:
  - A and B are attributes of a relation,
  - B is fully dependent on A if B is functionally dependent on A but not on any proper subset of A.
- 2NF - A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on the primary key.

1NF to 2NF

- Identify primary key for the 1NF relation.
- Identify functional dependencies in the relation.
- If partial dependencies exist on the primary key remove them by placing them in a new relation along with copy of their determinant.

Third Normal Form (3NF)

- Based on concept of transitive dependency:
  - A, B and C are attributes of a relation such that if A → B and B → C,
  - then C is transitively dependent on A through B. (Provided that A is not functionally dependent on B or C).
- 3NF - A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

2NF to 3NF

- Identify the primary key in the 2NF relation.
- Identify functional dependencies in the relation.
- If transitive dependencies exist on the primary key remove them by placing them in a new relation along with copy of their determinant.

General Definitions of 2NF and 3NF

- Second normal form (2NF)
  - A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on *any candidate key*.
- Third normal form (3NF)
  - A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on *any candidate key*.

Activity

In your COMP130 groups:

- Normalise the high score table for your game
- Update you implementation
- (45 minutes)