**YR 11 & 12 ATAR COMPUTER SCIENCE**

**REFERENCE BOOKLET**

This document has been created to assist WA Computer Science teachers and students understand some of the key concepts for the Year 11 and 12 ATAR Computer Science Syllabus. It is not a textbook and does not include content for the entire course.

This document is NOT provided or endorsed by SCSA and does not mandate the standards that teachers and students must follow. The content is an interpretation of the SCSA Syllabus by the author who has many years of experience teaching in this area.

Table of Contents

[Data Flow Diagrams (DFDs) 3](#_Toc58875470)

[Context Diagram (Year 11 and 12) 3](#_Toc58875471)

[Level 0 DFD (Year 11 and 12) 5](#_Toc58875472)

[Level 1 DFD (Year 12 ONLY) 7](#_Toc58875473)

[Normalisation 8](#_Toc58875474)

[Update Anomaly 8](#_Toc58875475)

[Insert Anomaly 8](#_Toc58875476)

[Deletion Anomaly 8](#_Toc58875477)

[Un-normalised Data 9](#_Toc58875478)

[First Normal Form - 1NF 10](#_Toc58875479)

[Second Normal Form - 2NF 11](#_Toc58875480)

[Third Normal Form - 3NF 12](#_Toc58875481)

[ERDs 13](#_Toc58875482)

[Cardinality 15](#_Toc58875483)

[SQL 18](#_Toc58875484)

[SELECT 18](#_Toc58875485)

[INSERT 19](#_Toc58875486)

[UPDATE 20](#_Toc58875487)

[DELETE 21](#_Toc58875488)

[Joining Tables 22](#_Toc58875489)

[Programming 24](#_Toc58875490)

[Flowchart Symbols 24](#_Toc58875491)

[Sequence 24](#_Toc58875492)

[Selection 25](#_Toc58875493)

[Iteration (repetition) 26](#_Toc58875494)

[Modularisation 28](#_Toc58875495)

[Structure Chart 30](#_Toc58875496)

[Cisco Network Diagram 31](#_Toc58875497)

[CISCO Icons 31](#_Toc58875498)

[Types of Routers 32](#_Toc58875499)

[Order of Devices 33](#_Toc58875500)

[Complete Cisco Network Diagram 36](#_Toc58875501)

## Data Flow Diagrams (DFDs)

### Context Diagram (Year 11 and 12)

A context diagram provides a visual tool to demonstrate the flows of data to and from a system to people or organisation outside of the system (known as external entities).

Context diagrams show the external entities (in rectangles), the system (in a circle) and the flows of data (lines with arrow indicated direction of data flow).

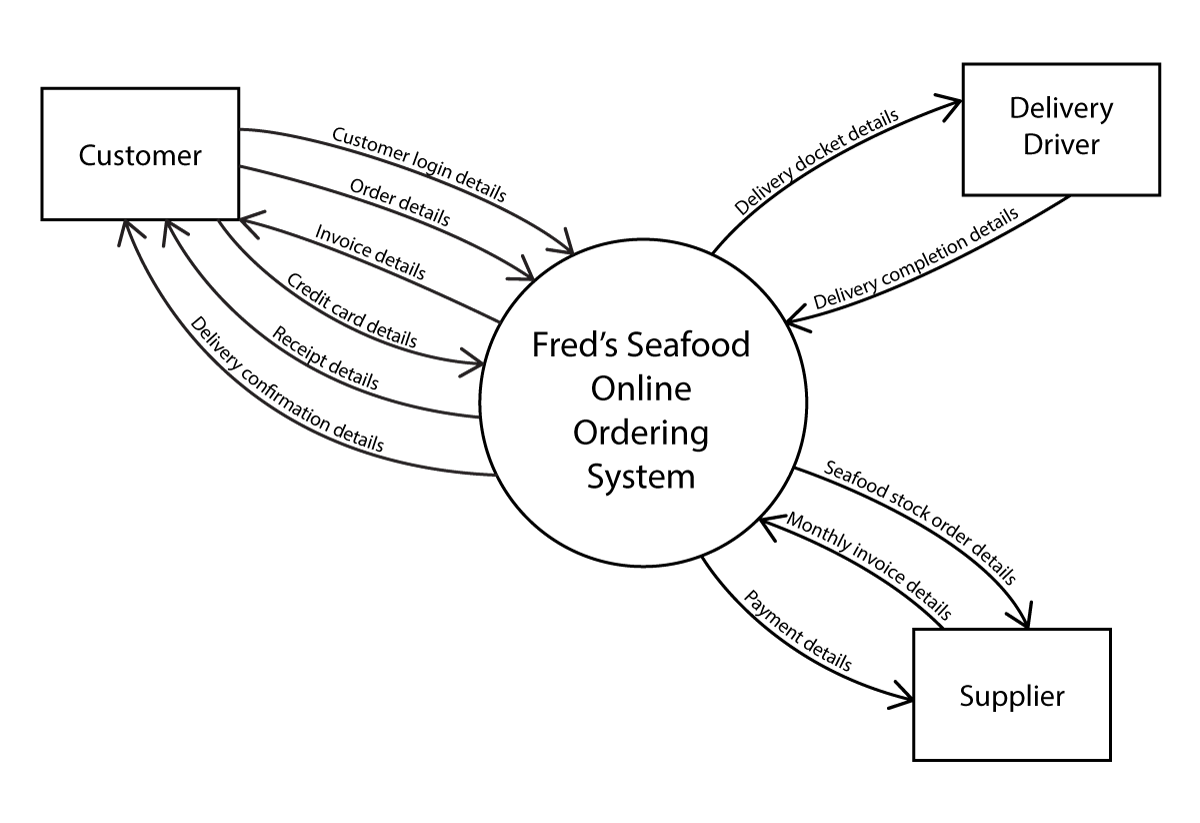
The context diagram allows us to understand the part of the system we have control over. We look further into the system when we look at the level 0 DFD (explained below). We can control aspects within the system but have no control over the external entities that interact with our system.

**Example:**

Fred’s Seafood has decided to implement a new digital online ordering system. It will allow their customers easy access to their products as well as provide automation for their deliveries and seafood ordering processes.

1. Customers go to Fred’s Seafood website and provide their login details to access the website. These details are checked against existing customers from the Customer database. If their login is successful, they can proceed on to order their seafood.
2. Customers then browse through the selection of seafood and place an order using the website. The system generates an invoice which is then displayed to the customer on the screen. A copy of the order is saved into the Order database.
3. The customer provides payment for the order using their credit card and the system sends a receipt of payment to the customer via email. The order database is updated with the customer payment details.
4. Fred’s Seafood kitchen staff read a screen showing the current order. They prepare the order as requested. Once completed they verbally inform the waitstaff the food is ready to be delivered.
5. Fred’s Seafood use an external delivery driver to get the food to the customer. When the driver arrives, staff hand them a delivery list with the order details including the address of the customer retrieved from the customer database. The delivery driver emails Fred’s Seafood once the delivery has been completed. Fred’s Seafood staff then send an email through to the customer confirming that the order has been delivered. The order database is updated with the delivery confirmation details and the stock levels are updated in the stock level database.
6. At the end of each day, the manager of Fred’s Seafood checks the stock levels from the stock level database and places an order for any seafood required. The manager sends the seafood order through to their supplier and saves a copy of the seafood stock order in the Stock Order database.
7. Each month the seafood supplier sends an invoice to Fred’s Seafood detailing the orders for the month. Fred’s Seafood checks the invoice against the orders it has on record in the Stock Order database and, if correct, sends through payment to the supplier.

Context Diagram (Year 11 and 12) *continued*



|  |  |  |
| --- | --- | --- |
| External Entity |  | An external entity is something that is outside of the control of the system. External entities are a person or an organisation and is represented as a singular noun. |
| System |  | The system is represented by a circle and always ends in the word ’system’. The circle indicates the system boundary. The processes exist within the scope of the project while the external entities are outside of the control of the system. |
| Data Flows |  | Data flows are represented by a line with an arrowhead. The arrow indicates the direction of flow of the data. The label on the line MUST indicate data rather than the actual physical representation. The easiest way to satisfy this is to use the keyword ‘details’ after each data flow. E.g. ‘receipt’ is not valid as it is a real, physical piece of paper where ‘receipt details’ indicates the data that is on the receipt. |

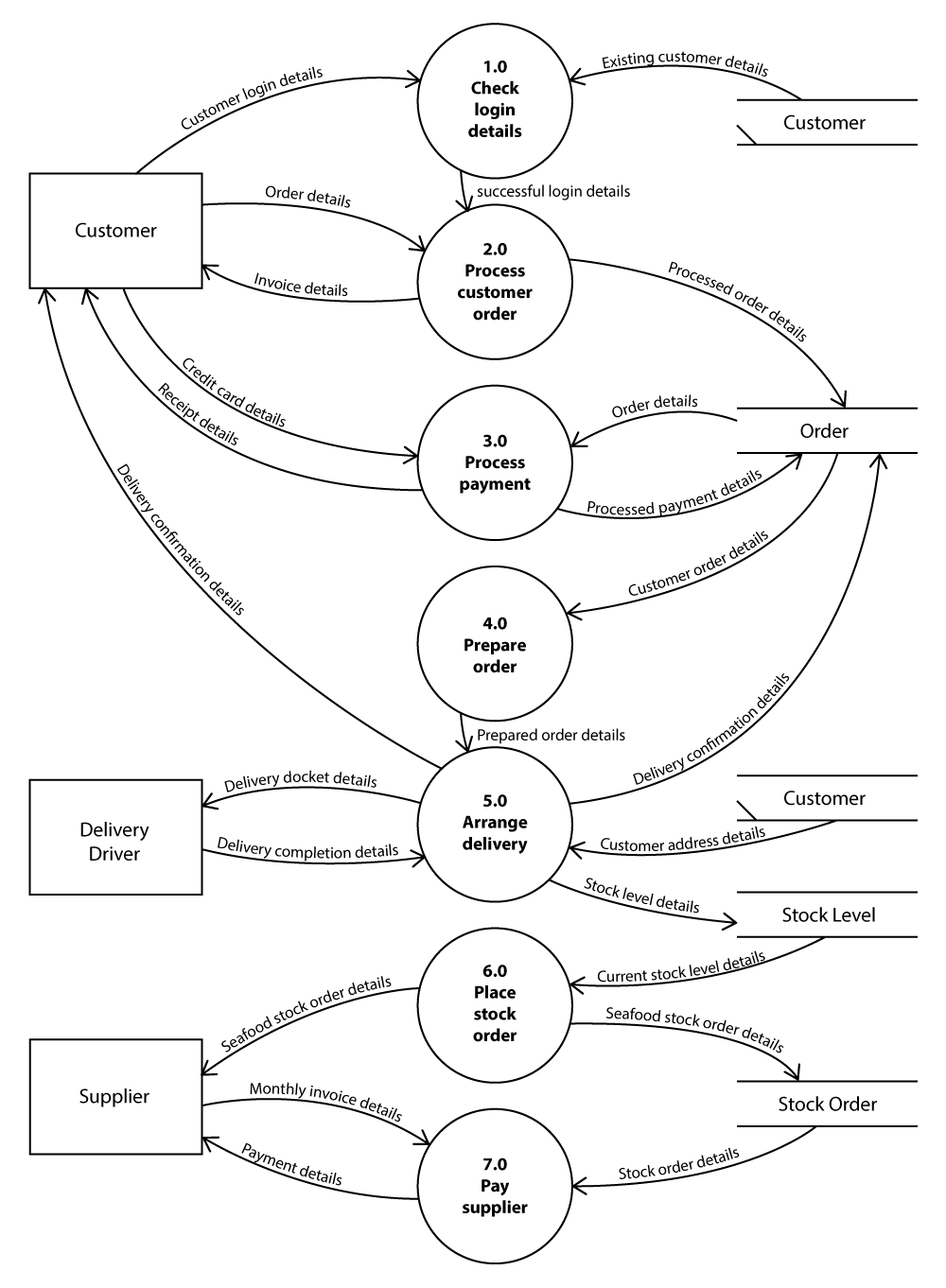
**Rules**

* MUST include the word ‘system’ when labeling the system
* End each data flow with the word ‘details’
* Context diagram DOES NOT show processes or data stores inside the system
* The circle for the system represents the system boundary

### Level 0 DFD (Year 11 and 12)

Level 0 DFDs allow us to look at the processes within the system. When we created a context diagram previously, we represented the entire system in a circle and did not look at what was happening inside the system at all.

A level 0 DFD shows the **same external entities that exist in the context diagram**. It also shows the **same flows of data to and from the system** that have been identified in the context diagram.



Level 0 DFD (Year 11 and 12) *continued*

|  |  |  |
| --- | --- | --- |
| External Entity  \*Same as Context Diagram\* |  | An external entity is something that is outside of the control of the system. External entities are a person or an organisation and is represented as a singular noun. |
| Process |  | Processes in a level 0 DFD are represented using the circle. Processes are actions that are happening to data within our system. They MUST start with a verb and must have a number used to represent the process number. Process numbering needs to be unique for each process. Process numbering does not need to be chronological, but it is most common to do so. |
| Data Store | A close up of a logo  Description automatically generated | A data store is represented by two parallel horizontal lines. The name for the data store is placed within the lines and represents what is being stored. Data store names are usually represented in the singular and may represent various types of data stores including a database, filing cabinet, pinup board etc. |
| Data Flows  \*Same as Context Diagram\* |  | Data flows are represented by a line with an arrowhead. The arrow indicates the direction of flow of the data. The label on the line MUST indicate data rather than the actual physical representation. The easiest way to satisfy this is to use the keyword ‘details’ after each data flow. E.g. ‘receipt’ is not valid as it is a real, physical piece of paper where ‘receipt details’ indicates the data that is on the receipt. |

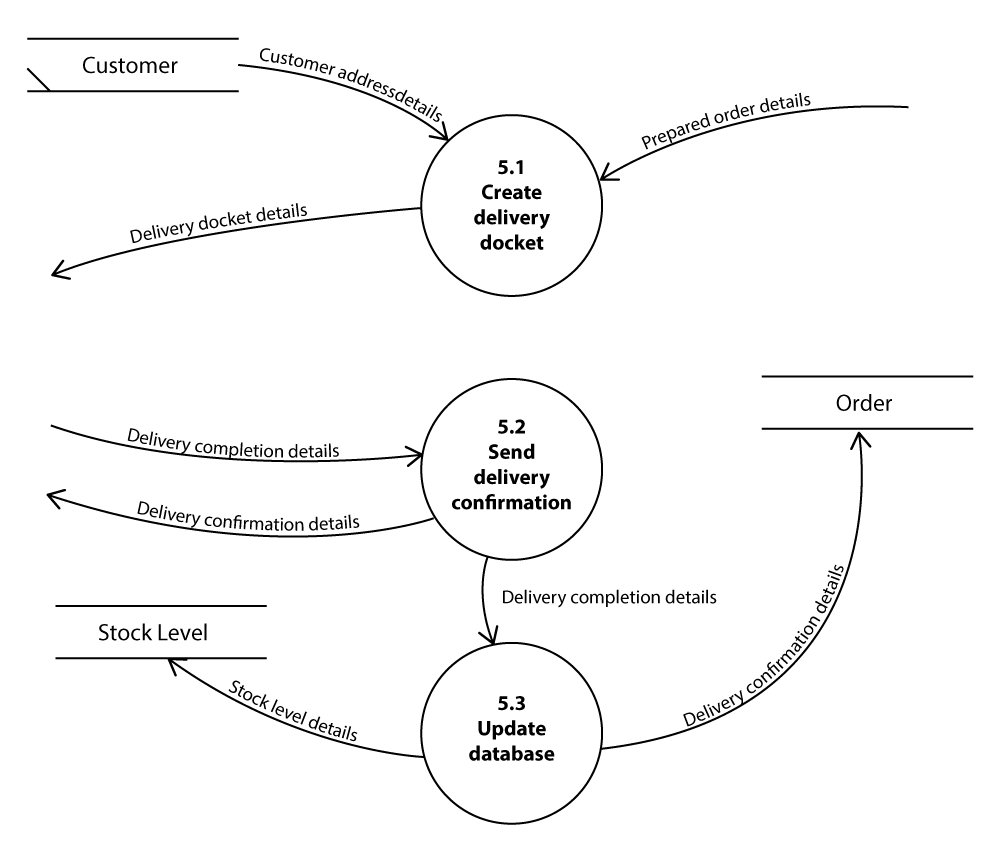
**Rules**

* Processes MUST have at least one data flow in AND at least one data flow out
* Process names MUST start with a verb and be numbered (1.0 etc.)
* The process should be named relational to the system – not the external entity (eg “Process payment” rather than “Pay money”)
* End each data flow with the word ‘details’
* Data flows CAN go from
  + External entity to and from a process
  + Data store to and from a process
  + Process to process
* Data flows CAN NOT go from
  + External entity to or from a data store
  + External entity to external entity
  + Data store to data store
* Level 0 DFDs MUST be balanced with the Context Diagram for the same system
  + MUST have the same External Entities in Level 0 DFD and Context Diagram
  + MUST have the same Data Flows to and from the External Entities and the System

### Level 1 DFD (Year 12 ONLY)

Level 1 DFD’s explore one of the processes from our level 0 DFD in more detail. We will look further into process 5.0. Level 1 DFD questions sometimes provide additional information.

Fred’s Seafood use an external delivery driver to get the food to the customer. When one of the waitstaff is told an order has been prepared and ready to go by the kitchen staff, the waitstaff creates a delivery docket. The delivery docket includes the order details as well as the address for delivery which is retrieved from the Customer database. When the driver arrives, staff hand them a delivery docket and the food. When the delivery driver has completed the delivery, they email Fred’s Seafood to inform them that the delivery has taken place. Fred’s Seafood immediately send an email through to the customer confirming that the order has been delivered. The order database is updated with the delivery confirmation details and the stock levels are updated in the stock level database.



**Rules**

* Processes MUST have at least one data flow in AND at least one data flow out
* Process numbers must match the process number from the level 0 DFD
* External entities are NOT shown – just the data flows to and from them
* Data stores are shown
* Level 1 DFDs MUST be balanced with the Level 0 DFD for the same system
  + MUST have the same Data Flows in and out of the Level 1 DFD as the Level 0 DFD

## Normalisation

Purpose

When data is put together, it is sometimes just placed all in one large table encompassing fields from a wide variety of areas. This can result in redundant data in the relations and may affect the integrity of the data in your database. We use normalisation to separate the data into relations (tables) that store related fields. Removing duplicate data results in a more efficient database taking up less space and increased reliability and integrity.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SaleID | Date | Customer | Phone | Item | Cost | Supplier | SuppAddress | SuppSuburb |
| 002315 | 23/8/19 | Jack Haddow | 0454 778 352 | 12ft Fishing Rod | $49.00 | Rods R Us | 1254 Canning Hwy | Como |
| 002316 | 24/8/19 | Sam Smith | 0487 662 325 | Tackle Box | $29.00 | WA Tackle Supplies | 125 Herald Ave | Riverton |
| 002317 | 26/8/19 | Juliette Relpin | 0411 321 888 | Fishing Hat | $29.00 | Stompies Hats | 89 Henderson St | Osborne Park |
| 002318 | 28/8/19 | Jack Haddow | 0454 778 352 | Octopus Circle Hooks | $9.90 | WA Tackle Supplies | 125 Herald Ave | Riverton |
| 002319 | 30/8/19 | Jack Haddow | 0454 778 352 | Fishing Gloves | $8.90 | WA Tackle Supplies | 125 Herald Ave | Riverton |

### Update Anomaly

Anomalies of this type occur when a relation has multiple instances of the same data. In situations where this data must be changed it is essential to ensure that all instances of the old data is changed to the new data. An example would be if Jack Haddow changed his mobile phone number to 0455 111 555. In the relation above, we would need to change the number in three locations. As we go through the process of normalisation we find that data duplication is removed, and updates then only need to be completed once ensuring the validity and integrity of the data.

### Insert Anomaly

An insert anomaly occurs when we need to add a new record but only have part of the information available to us. It is a result of data from different relations being grouped into the same relation. In this example we are storing sale information as well as customer information and supplier information. If we were to attempt to include a new supplier without having sale or customer information, we would be unable to as we would need all the information to create the new record.

### Deletion Anomaly

A deletion anomaly results when we delete information intentionally from our relation but accidentally lose additional information as a result of the data being stored together. An example would be if we no longer deal with Stompies Hats and want to remove them from our relation we would have to delete SaleID 2317. If we did this, we would unfortunately also lose all our data associated with Juliette Relpin.

The different forms for normalisation and the process for going through them is discussed over the next few pages.

### Un-normalised Data

Un-normalised data does not satisfy any of the requirements for it to be in first normal form.

Order

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OrderNo** | **Date** | **CustID** | **CustName** | **CustPhone** | **Item1ID** | **Item1Desc** | **Item1Cost** | **Item1Qty** | **Item1Total** | **Item2ID** | **Item2Desc** | **Item2Cost** | **Item2Qty** | **Item2Total** | **Item3ID** | **Item3Desc** | **Item3Cost** | **Item3Qty** | **Item3Total** | **Total** |
| 801 | 05/06/2019 | 201 | Jackson McIntosh | 0423 056 347 | 5 | Pink Snapper | $14.50 | 2 | $29.00 | 6 | Grilled Surcharge | $1.00 | 2 | $2.00 | 7 | Chips (paid per $) | $1.00 | 5 | 5.00 | $36.00 |
| 802 | 05/06/2019 | 202 | Richard Marst | 0407 489 133 | 8 | Hake and Chips | $9.50 | 4 | $38.00 | 13 | Tartare Sauce | $2.00 | 4 | $8.00 |  |  |  |  |  | $46.00 |
| 803 | 06/06/2019 | 202 | Richard Marst | 0407 489 133 | 3 | Shark | $9.00 | 1 | $9.00 | 11 | Crab Stick | $1.50 | 2 | $3.00 | 13 | Tartare Sauce | $2.00 | 1 |  | $14.00 |
| 804 | 06/06/219 | 205 | Sally Parker | 0402 155 264 | 2 | Spanish Mackerel | $10.50 | 1 | $10.50 | 1 | Hake | $8.50 | 2 | $17.00 |  |  |  |  |  | $27.50 |
| 805 | 06/06/2019 | 206 | Peter Smethelstein | 0489 966 477 | 12 | Squid (serve of 5) | $3.50 | 2 | $7.00 | 13 | Tartare Sauce | $2.00 | 2 | $4.00 |  |  |  |  |  | $11.00 |
| 806 | 06/06/2019 | 204 | Martin McLeuth | 0435 665 986 | 9 | Fish and Chips | $11.50 | 2 | $23.00 | 10 | Kids Fish and Chips | $7.50 | 3 | $22.50 |  |  |  |  |  | $45.50 |
| 807 | 07/06/2019 | 206 | Peter Smethelstein | 0489 966 477 | 5 | Pink Snapper | $14.50 | 4 | $58.00 |  |  |  |  |  |  |  |  |  |  | $58.00 |
| 808 | 07/06/2019 | 205 | Sally Parker | 0402 155 264 | 7 | Chips (paid per $) | $1.00 | 5 | $5.00 | 13 | Tartare Sauce | $2.00 | 2 | $4.00 |  |  |  |  |  | $9.00 |
| 809 | 07/06/2019 | 201 | Jackson McIntosh | 0423 056 347 | 9 | Fish and Chips | $11.50 | 2 | $23.00 |  |  |  |  |  |  |  |  |  |  | $23.00 |
| 810 | 07/06/2019 | 204 | Martin McLeuth | 0435 665 986 | 3 | Shark | $9.00 | 3 | $27.00 | 7 | Chips (paid per $) | $1.00 | 5 | $5.00 |  |  |  |  |  | $32.00 |

**Order** (**OrderNo**, Date, CustID, CustName, CustPhone, Item1ID, Item1Desc, Item1Qty, Item1Total, Item2ID, Item2Desc, Item2Qty, Item2Total, Item3ID, Item3Desc, Item3Qty, Item3Total, Total)

This table DOES NOT satisfy the requirement for 1NF

* MUST have a primary key – OrderNo is a primary key for this table
* All values be atomic – CustName is not atomic – the fields should be separated into first name and surname fields
* No repeating groups – fields for items have been repeated 3 times – {item1ID, item1Desc, item1Qty, item1Total}. {item2ID… etc.

### First Normal Form - 1NF

To get this relation into 1NF we must ensure all fields are atomic and remove repeating groups.

Step 1 – Make all values atomic

**Order** (OrderNo, Date, CustID, **CustFirstName, CustSurname**, CustPhone, Item1ID, Item1Desc, Item1Qty, Item1Total, Item2ID, Item2Desc, Item2Qty, Item2Total, Item3ID, Item3Desc, Item3Qty, Item3Total, Total)

This is still not in 1NF as it has repeating groups.

Step 2 – Remove repeating groups

Repeating groups in the order relation cause issues as each order may only include 1,2 or 3 items. This is an issue for any orders over 3 items. It also leaves NULL values in the relation if the order does not include three items.

We will remove the repeating information (item) and place it in a new relation. This relation will link to the original order relation on the foreign key OrderNo.

**Order** (**OrderNo**, Date, CustID, CustFirstName, CustSurname, CustPhone)

**Order\_Item** (**OrderNo FK**, **ItemID**, ItemDesc, ItemQty, ItemCost)

Order

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **OrderNo** | **Date** | **CustID** | **CustFirstName** | **CustSurname** | **CustPhone** |
| 801 | 05/06/2019 | 201 | Jackson | McIntosh | 0423 056 347 |
| 802 | 05/06/2019 | 202 | Richard | Marst | 0407 489 133 |
| 803 | 05/06/2019 | 202 | Richard | Marst | 0407 489 133 |
| 804 | 06/06/219 | 205 | Sally | Parker | 0402 155 264 |
| 805 | 06/06/2019 | 206 | Peter | Smethelstein | 0489 966 477 |
| 806 | 06/06/2019 | 204 | Martin | McLeuth | 0435 665 986 |
| 807 | 07/06/2019 | 206 | Peter | Smethelstein | 0489 966 477 |
| 808 | 07/06/2019 | 205 | Sally | Parker | 0402 155 264 |
| 809 | 07/06/2019 | 201 | Jackson | McIntosh | 0423 056 347 |
| 810 | 07/06/2019 | 204 | Martin | McLeuth | 0435 665 986 |

Order\_Item

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderNo** | **ItemID** | **ItemDesc** | **ItemQty** | **ItemCost** |
| 801 | 5 | Pink Snapper | 2 | $14.50 |
| 801 | 6 | Grilled Surcharge | 2 | $1.00 |
| 801 | 7 | Chips (paid per $) | 5 | $1.00 |
| 802 | 8 | Hake and Chips | 4 | $9.50 |
| 802 | 13 | Tartare Sauce | 4 | $2.00 |
| 803 | 3 | Shark | 1 | $8.50 |
| 803 | 11 | Crab Stick | 2 | $1.50 |
| 803 | 13 | Tartare Sauce | 1 | $2.00 |
| 804 | 2 | Spanish Mackerel | 1 | $10.50 |
| 804 | 1 | Hake | 2 | $8.50 |
| 805 | 12 | Squid (serve of 5) | 2 | $3.50 |
| 805 | 13 | Tartare Sauce | 2 | $2.00 |
| 806 | 9 | Fish and Chips | 2 | $11.50 |
| 806 | 10 | Kids Fish and Chips | 3 | $7.50 |
| 807 | 5 | Pink Snapper | 4 | $14.50 |
| 808 | 7 | Chips (paid per $) | 5 | $1.00 |
| 808 | 13 | Tartare Sauce | 2 | $2.00 |
| 809 | 9 | Fish and Chips | 2 | $11.50 |
| 810 | 3 | Shark | 3 | $8.50 |
| 810 | 7 | Chips (paid per $) | 5 | $1.00 |

These relations are now in 1NF as they satisfy all the requirements:

* MUST have a primary key
* All values be atomic
* No repeating groups

### Second Normal Form - 2NF

To be in second normal form, the relation(s) must satisfy the following requirements:

* MUST be in 1NF
* No partial dependencies

**What is a Partial Dependency?**

All non-key fields in a relation (i.e. those fields that are not a primary key or a foreign key) must be functionally dependent on the WHOLE KEY. If any non-key fields only depend on part of a key (which is only possible if there is a composite primary key) then there is a partial dependence.

The relations below are in 1NF. Partial dependencies can only occur when a composite primary key is used, so we know that the Order relation does not have any partial dependence.

The primary key for the Order\_Item relation is a composite primary key as on each order there can only be one instance of a particular item - (OrderNo, ItemID). However, the non-key fields in the Order\_Item relation do not all depend on the WHOLE key. ItemDesc and ItemCost depend on ItemID only and not on OrderNo. ItemQty depends on both ItemID and OrderNo.

**Order** (**OrderNo**, Date, CustID, CustFirstName, CustSurname, CustPhone)

**Order\_Item** (**OrderNo FK**, **ItemID**, ItemDesc, ItemQty, ItemCost)

To remove the partial dependence, we must move the fields that depend on ItemID to a new relation called Item.

**Order** (**OrderNo**, Date, CustID, CustFirstName, CustSurname, CustPhone)

**Order\_Item** (**OrderNo FK**, **ItemID FK**, ItemQty)

**Item** (**ItemID**, ItemDesc, ItemCost)

Order

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **OrderNo** | **Date** | **CustID** | **CustFirstName** | **CustSurname** | **CustPhone** |
| 801 | 05/06/2019 | 201 | Jackson | McIntosh | 0423 056 347 |
| 802 | 05/06/2019 | 202 | Richard | Marst | 0407 489 133 |
| 803 | 05/06/2019 | 202 | Richard | Marst | 0407 489 133 |
| 804 | 06/06/219 | 205 | Sally | Parker | 0402 155 264 |
| 805 | 06/06/2019 | 206 | Peter | Smethelstein | 0489 966 477 |
| 806 | 06/06/2019 | 204 | Martin | McLeuth | 0435 665 986 |
| 807 | 07/06/2019 | 206 | Peter | Smethelstein | 0489 966 477 |
| 808 | 07/06/2019 | 205 | Sally | Parker | 0402 155 264 |
| 809 | 07/06/2019 | 201 | Jackson | McIntosh | 0423 056 347 |
| 810 | 07/06/2019 | 204 | Martin | McLeuth | 0435 665 986 |

These relations are now in 2NF as they satisfy all the requirements:

* MUST be in 1NF
* No partial dependencies

Order\_Item

|  |  |  |
| --- | --- | --- |
| **OrderNo** | **ItemID** | **ItemQty** |
| 801 | 5 | 2 |
| 801 | 6 | 2 |
| 801 | 7 | 5 |
| 802 | 8 | 4 |
| 802 | 13 | 4 |
| 803 | 3 | 1 |
| 803 | 11 | 2 |
| 803 | 13 | 1 |
| 804 | 2 | 1 |
| 804 | 1 | 2 |
| 805 | 12 | 2 |
| 805 | 13 | 2 |
| 806 | 9 | 2 |
| 806 | 10 | 3 |
| 807 | 5 | 4 |
| 808 | 7 | 5 |
| 808 | 13 | 2 |
| 809 | 9 | 2 |
| 810 | 3 | 3 |
| 810 | 7 | 5 |

Item

|  |  |  |
| --- | --- | --- |
| **ItemID** | **ItemDesc** | **ItemCost** |
| 1 | Hake | $8.50 |
| 2 | Spanish Mackerel | $10.50 |
| 3 | Shark | $9.00 |
| 4 | Barramundi | $11.50 |
| 5 | Pink Snapper | $14.50 |
| 6 | Grilled Surcharge | $1.00 |
| 7 | Chips (paid per $) | $1.00 |
| 8 | Hake and Chips | $9.50 |
| 9 | Fish and Chips | $11.50 |
| 10 | Kids Fish and Chips | $7.50 |
| 11 | Crab Stick | $1.50 |
| 12 | Squid (serve of 5) | $3.50 |
| 13 | Tartare Sauce | $2.00 |

### Third Normal Form - 3NF

To be in third normal form, the relations must satisfy the following requirements:

* MUST be in 2NF
* No transitive dependencies

**What is a Transitive Dependency?**

All non-key fields in a relation must be fully functionally dependent on the NOTHING BUT THE KEY. Transitive dependence occurs when a non-key field depends on another non-key field rather than the primary key.

While the Order\_Item and Item tables do not have any transitive dependencies, the Order table does.

**Order** (**OrderNo**, Date, CustID, CustFirstName, CustSurname, CustPhone)

**Order\_Item** (**OrderNo FK**, **ItemID FK**, ItemQty)

**Item** (**ItemID**, ItemDesc, ItemCost)

Date and CustID and both fully functionally dependent on OrderID but CustFirstName, CustSurname and CustPhone are dependent on CustID which is a non-key field. To fix this, we will create a new relation that will link to the Order table on CustID.

**Customer (CustID**, CustFirstName, CustSurname, CustPhone)

**Order** (**OrderNo**, Date, CustID FK)

**Order\_Item** (**OrderNo FK**, **ItemID FK**, ItemQty)

**Item** (**ItemID**, ItemDesc, ItemCost)

Customer

|  |  |  |  |
| --- | --- | --- | --- |
| **CustID** | **CustFirstName** | **CustSurname** | **CustPhone** |
| 201 | Jackson | McIntosh | 0423 056 347 |
| 202 | Richard | Marst | 0407 489 133 |
| 204 | Martin | McLeuth | 0435 665 986 |
| 205 | Sally | Parker | 0402 155 264 |
| 206 | Peter | Smethelstein | 0489 966 477 |

Order

These relations are now in 3NF as they satisfy all the requirements:

* MUST be in 2NF
* No transitive dependencies

|  |  |  |
| --- | --- | --- |
| **OrderNo** | **Date** | **CustID** |
| 801 | 05/06/2019 | 201 |
| 802 | 05/06/2019 | 202 |
| 803 | 05/06/2019 | 202 |
| 804 | 06/06/219 | 205 |
| 805 | 06/06/2019 | 206 |
| 806 | 06/06/2019 | 204 |
| 807 | 07/06/2019 | 206 |
| 808 | 07/06/2019 | 205 |
| 809 | 07/06/2019 | 201 |
| 810 | 07/06/2019 | 204 |

Order\_Item

|  |  |  |
| --- | --- | --- |
| **OrderNo** | **ItemID** | **ItemQty** |
| 801 | 5 | 2 |
| 801 | 6 | 2 |
| 801 | 7 | 5 |
| 802 | 8 | 4 |
| 802 | 13 | 4 |
| 803 | 3 | 1 |
| 803 | 11 | 2 |
| 803 | 13 | 1 |
| 804 | 2 | 1 |
| 804 | 1 | 2 |
| 805 | 12 | 2 |
| 805 | 13 | 2 |
| 806 | 9 | 2 |
| 806 | 10 | 3 |
| 807 | 5 | 4 |
| 808 | 7 | 5 |
| 808 | 13 | 2 |
| 809 | 9 | 2 |
| 810 | 3 | 3 |
| 810 | 7 | 5 |

Item

|  |  |  |
| --- | --- | --- |
| **ItemID** | **ItemDesc** | **ItemCost** |
| 1 | Hake | $8.50 |
| 2 | Spanish Mackerel | $10.50 |
| 3 | Shark | $9.00 |
| 4 | Barramundi | $11.50 |
| 5 | Pink Snapper | $14.50 |
| 6 | Grilled Surcharge | $1.00 |
| 7 | Chips (paid per $) | $1.00 |
| 8 | Hake and Chips | $9.50 |
| 9 | Fish and Chips | $11.50 |
| 10 | Kids Fish and Chips | $7.50 |
| 11 | Crab Stick | $1.50 |
| 12 | Squid (serve of 5) | $3.50 |
| 13 | Tartare Sauce | $2.00 |

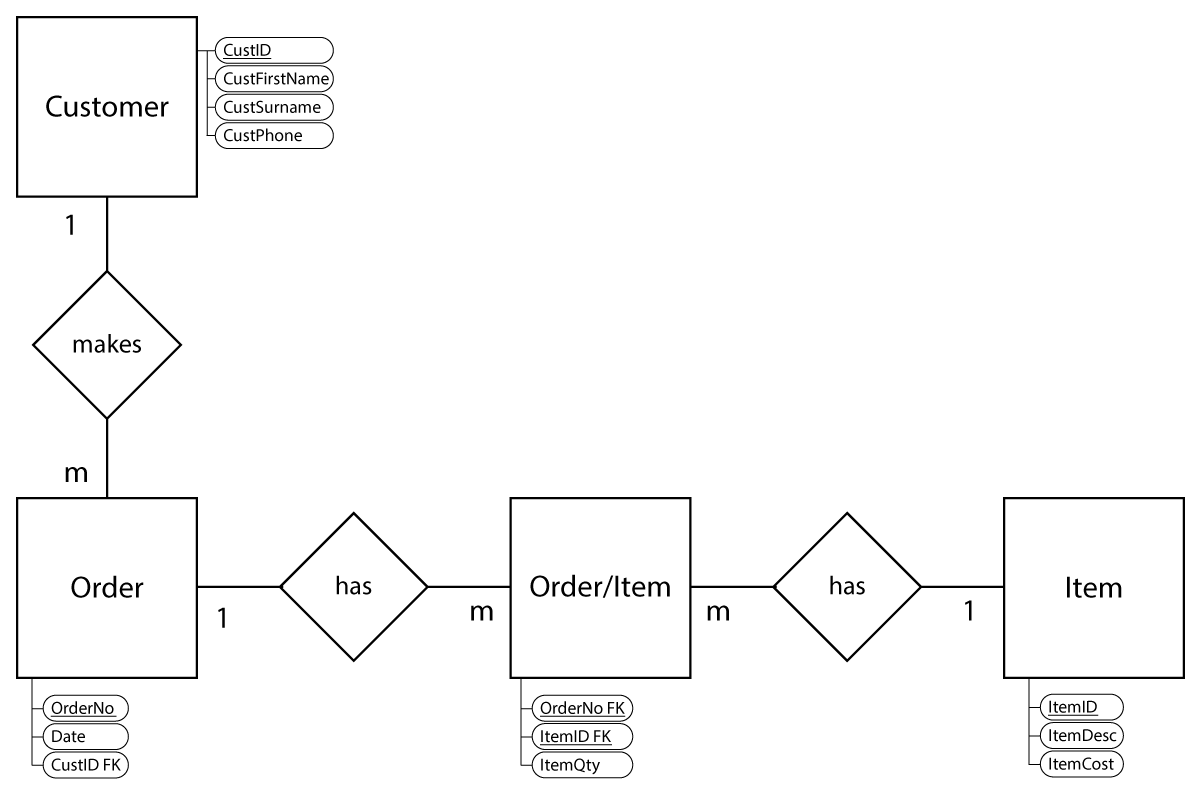
## ERDs

Purpose

Entity Relationship Diagrams (ERDs) are a visual tool for displaying the table layout for a planned database, along with the relationships and cardinality for the tables (relations).

Based on the relations from the previous page representing the database for Fred’s Seafood:

* One Customer may make many orders
* Each order has many items
* An item may appear on many orders



Shapes

|  |  |  |
| --- | --- | --- |
| Entity | A picture containing bird, tree, flower  Description automatically generated | Entities in an ERD are represented in a square or rectangle. Entities represent the relations (tables) in your database. |
| Relationships | A close up of a logo  Description automatically generated | Relationships are displayed in a diamond. Relationships connect two relations (tables). The wording used in relationship represents how the two relations are linked (e.g. Customer makes many orders). It is commonly accepted that if no clear wording can be established, “has” may be used. |
| Cardinality | A close up of a logo  Description automatically generated | The cardinality represents how many times a record in a relation (table) is able to link together with items in another relation (table). I.e. One customer can make many orders represents a one-to-many (1:M) relationship. One order can have many items and each item can be on many different orders. This represents a many-to-many (M:N) relationship. Cardinalities may be 1:1, 1:M, M:1 or M:N. All many-to-many relationships must be resolved when designing ERDs and creating databases (explained in next section). |
| Primary Key | A drawing of a person  Description automatically generated | A unique attribute used to identify each record. Primary keys are demonstrated in Chen Notation by underlining the key. |
| Foreign Key | A picture containing drawing  Description automatically generated | The foreign key is the link between two relations. A foreign key in a relation(table) links to a primary key in the relation(table) it is linked to. Foreign keys have ‘FK’ after their field name. |
| Non-key attributes |  | These are any other attributes for the relation other than the primary key and any foreign keys. |

### Cardinality

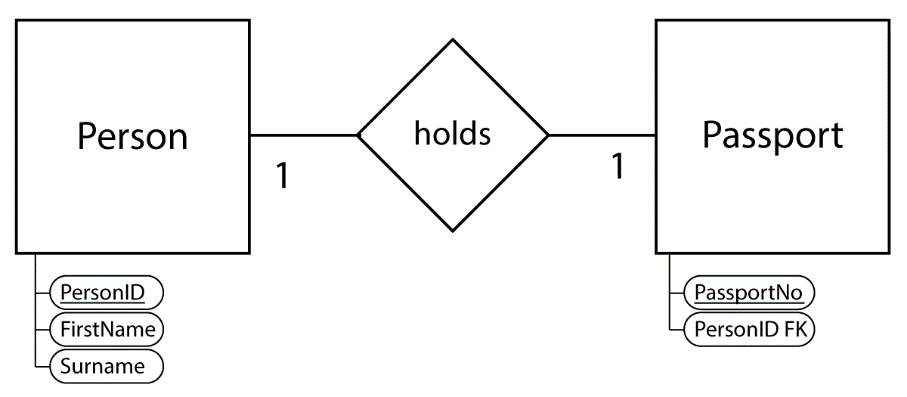
When establishing cardinality, we need to determine the relationship type between two entities. The possible cardinalities are:

* 1:1 – One-to-one
* 1:M – One-to-many
* M:1 – Many-to-one
* M:N – Many-to-many

**1:1**

A one-to-one relationship exists when for each record in a relation there can only be one possible record in a linked relation for that record. This is the least common database relationship.

An example for this is an Australian passport. Each Australian citizen may only hold one Australian passport.



**1:M or M:1**

A one-to-many or many-to-one relationship is the most common database relationship. This occurs when one record of a relation is able to relate to more than one record in a linked relation.

A one-to-many relationship from our previous ERD example is a situation where one customer can make many orders from Fred’s Seafood. For each order there can only be one customer.

A screenshot of a cell phone

Description automatically generated

Important Note: The foreign key in a 1:M relationship ALWAYS exists on the MANY side. In our example above the foreign key (CustID) is in the Order relation.

**M:N**

Many-to-many relationships exists in situations where a 1:M relationship exists in both directions for a relationship.

An example for this is in our previous ERD. One each order there can be many different items. This indicates a 1:M cardinality in one direction between order and item.

A picture containing clock

Description automatically generated

BUT… Each item can be on more than one order. This indicates a 1:M cardinality in the other direction between item and order.

A picture containing clock

Description automatically generated

If 1:M cardinalities exist in both directions, a M:N cardinality exists as per the final diagram below.

A picture containing clock

Description automatically generated

NOTE: M:N cardinality cannot be entered into a database and MUST be resolved in all complete ERDs.

**Resolving Many-to-Many Relationships**

Many to many relationships exist in systems where there is a one-to-many relationship in both directions between the entities.

Many to many relationships CAN NOT exist in a final ERD. They must be resolved using the process below.

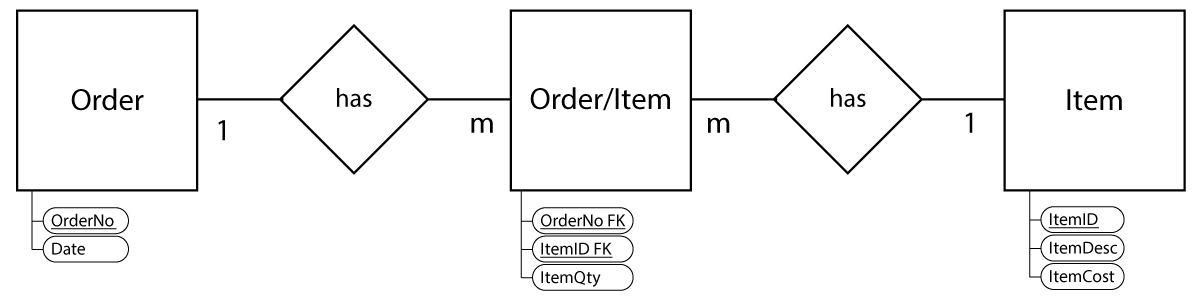
In our example we have a many-to-many relationship between the order entity and the item entity.

A picture containing clock

Description automatically generated

To resolve this, we need to create a new entity in between these two called an associative entity (also sometimes called a linking entity or joining entity).

If there is an obvious name for the associative entity we would use that to name the entity. An example would be a M:N relationship between student and class could name the associative entity Class List. If there is no obvious name as in our example above, it is common practice to join the names of either original entity. In our example we would name our new associative entity – order/item.



There are some important things to note about our resolved ERD:

* The cardinality always becomes 1:M - M:1 with the “Many” on the new associative entity
* The associative entity has foreign keys from both original entities (as it is on the “many” side)
* The associative entity needs a primary key. This may sometimes be a composite primary key using both foreign keys (as show above) but in some situations this is not a unique record. The other option (and sometimes recommended) is to create a new primary key such as orderItemID.

## SQL

Structured Query Language is the common 4G language that is utilised in many RDBMS. It provides a written way of interacting with a database and allows text-based lines to perform database operations. The provides an easier way to interact with the database rather than having to use the database software.

Note: there are a few conventions that will be used in the examples below, but they are not essential for SQL to run or essential for the Computer Science course.

* Using upper case for the keywords
* Splitting the SQL over multiple lines
* Ending the SQL statement with a semicolon (;)

### SELECT

SELECT is the most commonly used keyword in SQL as it is used to run a query on a database to retrieve data.

The basic structure of SELECT is:

SELECT <field>

FROM <table>

WHERE <filter criteria>

AND <filter criteria 2> (optional);

Customer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** |
| 2022 | Alison | McKay | Kelmscott | WA |
| 2023 | Ronald | Jefferies | Clarkson | WA |
| 2024 | Jackson | Albury | Churchlands | WA |
| 2025 | Rachael | Smith | Queens Park | WA |
| 2026 | Mary | Jefferies | Clarkson | WA |
| 2027 | Harry | Zipkins | St Kilda | VIC |

SELECT Examples

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SELECT \*  FROM Customer; | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** | | 2022 | Alison | McKay | Kelmscott | WA | | 2023 | Ronald | Jefferies | Clarkson | WA | | 2024 | Jackson | Albury | Churchlands | WA | | 2025 | Rachael | Smith | Queens Park | WA | | 2026 | Mary | Jefferies | Clarkson | WA | | 2027 | Harry | Zipkins | St Kilda | VIC | |
| SELECT FirstName, Surname  FROM Customer; | |  |  | | --- | --- | | **FirstName** | **Surname** | | Alison | McKay | | Ronald | Jefferies | | Jackson | Albury | | Rachael | Smith | | Mary | Jefferies | | Harry | Zipkins | |
| SELECT \*  FROM Customer  WHERE Suburb = ‘Clarkson’; | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** | | 2023 | Ronald | Jefferies | Clarkson | WA | | 2026 | Mary | Jefferies | Clarkson | WA | |
| SELECT FirstName, Surname, State  FROM Customer  WHERE Surname = ‘Jefferies’ AND FirstName = ‘Ronald’; | |  |  |  | | --- | --- | --- | | **FirstName** | **Surname** | **State** | | Ronald | Jefferies | WA | |

### INSERT

The syntax to add a new row of data to a table. You may include data for all fields in the new row or only partial data.

There are two possible variations for INSERT. One is for when values are known for all fields in the row. The second form is used when only some of the values are known.

Form 1 – Values for all fields are known

INSERT INTO <table>

VALUES (value1, value2, value3….);

Example 1 – adding an extra row of data into our Customer table above.

INSERT INTO Customer

VALUES (2028, ‘Kathryn’, ‘Jackovich’, ‘Warnbro’, ‘WA’);

Customer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** |
| 2022 | Alison | McKay | Kelmscott | WA |
| 2023 | Ronald | Jefferies | Clarkson | WA |
| 2024 | Jackson | Albury | Churchlands | WA |
| 2025 | Rachael | Smith | Queens Park | WA |
| 2026 | Mary | Jefferies | Clarkson | WA |
| 2027 | Harry | Zipkins | St Kilda | VIC |
| 2028 | Kathryn | Jackovich | Warnbro | WA |

Example 2 – adding a new record without values for all fields. This will be added to the updated table above.

INSERT INTO Customer (CustomerID, FirstName, Surname)

VALUES (2029, ‘Ralph’, ‘Norris’);

Customer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** |
| 2022 | Alison | McKay | Kelmscott | WA |
| 2023 | Ronald | Jefferies | Clarkson | WA |
| 2024 | Jackson | Albury | Churchlands | WA |
| 2025 | Rachael | Smith | Queens Park | WA |
| 2026 | Mary | Jefferies | Clarkson | WA |
| 2027 | Harry | Zipkins | St Kilda | VIC |
| 2028 | Kathryn | Jackovich | Warnbro | WA |
| 2029 | Ralph | Norris |  |  |

### UPDATE

This is used when values in an existing row need to be changed (updated).

The basic structure of UPDATE is:

UPDATE <table>

SET columnname1 = value1, columnname2 = value 2, …

WHERE <filter criteria> (optional but highly recommended – otherwise all records will be updated)

AND <filter criteria 2> (optional);

Example 1 – Fixing a spelling mistake in the table above to correct Ralph Norris to Ralph Morris.

UPDATE Customer

SET Surname = ‘Morris’

WHERE CustomerID = 2029;

Customer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** |
| 2022 | Alison | McKay | Kelmscott | WA |
| 2023 | Ronald | Jefferies | Clarkson | WA |
| 2024 | Jackson | Albury | Churchlands | WA |
| 2025 | Rachael | Smith | Queens Park | WA |
| 2026 | Mary | Jefferies | Clarkson | WA |
| 2027 | Harry | Zipkins | St Kilda | VIC |
| 2028 | Kathryn | Jackovich | Warnbro | WA |
| 2029 | Ralph | Morris |  |  |

### DELETE

Used to remove one or more records from a table.

The basic structure of DELETE is:

DELETE FROM <table>

WHERE <filter criteria> (optional but highly recommended – otherwise all records will be deleted)

AND <filter criteria 2> (optional);

Example 1 – Remove all records for people who live in Clarkson

DELETE FROM Customer

WHERE Suburb = ‘Clarkson’;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** |
| 2022 | Alison | McKay | Kelmscott | WA |
| 2024 | Jackson | Albury | Churchlands | WA |
| 2025 | Rachael | Smith | Queens Park | WA |
| 2027 | Harry | Zipkins | St Kilda | VIC |
| 2028 | Kathryn | Jackovich | Warnbro | WA |
| 2029 | Ralph | Morris |  |  |

Example 2 – Intentionally delete all records from table OR forget to include a filter using WHERE.

DELETE FROM Customer;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **Surname** | **Suburb** | **State** |

### Joining Tables

When using SQL with relational databases there is often the requirement to source data from more than one table – linked on the foreign key.

There are a number of SQL statements that may be used to perform joins depending on the required data to be returned. These include INNER JOIN, LEFT JOIN, RIGHT JOIN and FULL JOIN. Each of these commands results in a slightly different. There is a less common but easy technique to join tables. We will discuss the easy technique and INNER JOIN – both which produce the same result.

Driver

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DriverID** | **FirstName** | **Surname** | **Address** | **Suburb** | **State** | **PostCode** | **Phone** |
| 1 | Chloe | Apollo | 56 Holden Rd | Roleystone | WA | 6111 | 0422 023 652 |
| 2 | Olivia | Diaz | 86 Lakeview Dr | Edgewater | WA | 6027 | 0400 231 547 |
| 3 | Emily | Mahoney | 12 Kirby Way | Samson | WA | 6163 | 0431 005 569 |
| 4 | Lachlan | Piletro | 1230 Albany Hwy | Gosnells | WA | 6112 | 0408 062 379 |
| 5 | Riley | Hinley | 98 Hickson St | Mt Lawley | WA | 6050 | 0412 365 986 |
| 6 | Marissa | De Cruz | 22 Busby Ln | Gosnells | WA | 6024 | 0431 564 834 |

Car

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CarID** | **Year** | **Make** | **Model** | **FuelType** | **BodyType** | **Transmission** | **DriverID** |
| 1 | 2019 | Toyota | Camry | Petrol | Sedan | Automatic | 2 |
| 2 | 2004 | Mitsubishi | Pajero | Diesel | Wagon | Automatic | 6 |
| 3 | 2010 | Hyundai | i30 | Diesel | Hatch | Automatic | 6 |
| 4 | 2004 | Nissan | 350z | Petrol | Coupe | Manual | 4 |
| 5 | 2019 | Volkswagon | Amarok | Petrol | Wagon | Manual | 5 |
| 6 | 2016 | Toyota | Prius | Hybrid | Wagon | Automatic | 3 |
| 7 | 2013 | Mazda | 3 | Petrol | Hatch | Manual | 1 |

**Example 1**

List the FirstName, Surname, Year, Make and Model for all drivers who live in Gosnells

Method # 1 (Easy method)

SELECT Driver.FirstName, Driver.Surname, Car.Year, Car.Make, Car.Model

FROM Driver, Car

WHERE Driver.DriverID = Car.DriverID

AND Driver.Suburb = ‘Gosnells’

Method # 2 (Better method)

SELECT Driver.FirstName, Driver.Surname, Car.Year, Car.Make, Car.Model

FROM Driver INNER JOIN Car ON Driver.DriverID = Car.DriverID

WHERE Driver.Suburb = ‘Gosnells’

Both SQL methods result in the same way:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Driver.FirstName** | **Driver.Surname** | **Car.Year** | **Car.Make** | **Car.Model** |
| Lachlan | Piletro | 2004 | Nissan | 350z |
| Marissa | De Cruz | 2004 | Mitsubishi | Pajero |
| Marissa | De Cruz | 2010 | Hyundai | i30 |

**Example 2**

List all the fields from the driver table for all drivers who drive cars with a manual transmission

Method # 1 (Easy method)

SELECT Driver.\*

FROM Driver, Car

WHERE Driver.DriverID = Car.DriverID

AND Car.Transmission = ‘Manual’

Method # 2 (Better method)

SELECT Driver.\*

FROM Driver INNER JOIN Car ON Driver.DriverID = Car.DriverID

WHERE Car.Transmission = ‘Manual’

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DriverID** | **FirstName** | **Surname** | **Address** | **Suburb** | **State** | **PostCode** | **Phone** |
| 1 | Chloe | Apollo | 56 Holden Rd | Roleystone | WA | 6111 | 0422 023 652 |
| 4 | Lachlan | Piletro | 1230 Albany Hwy | Gosnells | WA | 6112 | 0408 062 379 |
| 5 | Riley | Hinley | 98 Hickson St | Mt Lawley | WA | 6050 | 0412 365 986 |

## Programming

Flowcharts – A flowchart is an easy visual way to demonstrate the flow through the algorithm.

Pseudocode – An informal high-level language that is used to demonstrate the algorithm prior to then coding the program in a set programming language such as C# or Java.

### Flowchart Symbols

|  |  |
| --- | --- |
| Symbol | Meaning |
|  | Terminal: Begin or End  Used to determine the starting point and end point of the algorithm |
|  | Input or Output  Used to show data that is entered and defined as a variable in the algorithm or data that is being displayed to the end user. |
|  | Process  Used to display a process that is performed on the data in the algorithm. This may be such things as a mathematic calculation or concatenating a string etc. |
|  | Decision  A decision shows the direction of flow of data depending on a logical comparison. The logical comparison is written in the diamond and, depending on the type of selection statement being used, will have either one, two or many possible directions to take. |

### Sequence

|  |  |  |
| --- | --- | --- |
| **Sequence**  Used when a sequence of lines in the algorithm flow line by line |  | Output(‘What is your name? ‘)  Input(name)  Output(‘Hello ’, name) |

### Selection

|  |  |  |
| --- | --- | --- |
| **One-way selection –**  **If..Then statement**  Used to test a condition and if the condition is TRUE then some additional is run, otherwise no code is run, and the statement finishes |  | Input (deliveryDistance)  If deliveryDistance >= 30 then  surcharge 🡨 0.1  End if  orderPrice 🡨 orderPrice + orderPrice \* surcharge  Output (orderPrice) |
| **Two-way selection – If..Then..Else statement**  Used to test a condition and if the condition is TRUE then some additional code is run. If the condition is FALSE, an alternate code is run before the statement finishes |  | Output (‘Do you have a loyalty card?’)  Input(hasLoyalty)  If (hasLoyalty = ‘y’) or (hasLoyalty = ‘yes’) then  discount 🡨 0.1  else  discount 🡨 0  End if  orderPrice 🡨 orderPrice – orderPrice \* discount  Output (orderPrice) |
| **Multi-way selection –**  **Case statement**  Used to test a condition against a variety of possible outcomes. The case statement considers each condition one line at a time. If the condition of the line being considered is TRUE, then the corresponding code is run before the statement ends. If the condition is FALSE, it then considers the next line of the case statement. |  | Input(orderPrice)  Input(deliveryDistance)  Case deliveryDistance of  <30: surcharge 🡨 0  <100: surcharge 🡨 0.1  >=100: surcharge 🡨 0.2  End case  orderPrice 🡨 orderPrice + orderPrice \* surcharge  Output (orderPrice) |

### Iteration (repetition)

|  |  |  |
| --- | --- | --- |
| **Counted loop – For loop**  The counted loop is used in situations where the number of instances for the iteration is known. A local variable is initialised in the for-loop declaration and then increments until it reaches the final value declared at the end of the for loop. |  | numOrders 🡨 5  totalOrder 🡨 0  For count 🡨 1 to numOrders  Input (orderValue)  totalOrder 🡨 totalOrder + orderValue  End For  orderAverage 🡨 totalOrder / numOrders  Output(orderAverage) |
| **Test first loop –**  **While loop**  A test first loop tests a condition at the beginning of the loop. If the condition is true then the code within the structure is run. The loop is then rechecked at the beginning of each loop until the condition is false when the loop is broken.  An important aspect of the test first loop is that it is possible that the condition may be false the first time the condition is checked so it is possible for the code in the loop to not run at all. |  | totalOrder 🡨 10.00  Output (‘Please enter payment amount’)  Input (payment)  While payment < totalOrder  Output (‘Insufficient payment amount. Try again’)  Input (payment)  End While  change 🡨 payment – totalOrder  Output (change) |
| **Test-last loop – Repeat…Until loop**  The test last loop checks for the condition at the end of each iteration of the loop.  This difference means that the test-last loop will always run at least once. | A close up of a map  Description automatically generated | totalOrder 🡨 0  Repeat  Output (‘Please enter the item you wish to add to the order or exit to end’)  Input (item)  Case item of  = ‘Hake’: itemCost 🡨 8.50  = ‘Shark’: itemCost 🡨 9.00  = ‘Barramundi’: itemCost 🡨 11.50  End Case  If item <> ‘exit’ then  Output (‘How many of this item?’)  Input(itemQuantity)  totalOrder 🡨 totalOrder + itemCost\*itemQuantity  End if  Until item = ‘exit’  Output (totalOrder) |

**Beginning and End of Algorithms**

It is common to use different statements for the beginning and end of algorithms in Year 11 and in Year 12. This is because modularisation (discussed in the next section) is introduced in Year 12.

**Year 11**

Begin

Output(‘What is your name? ‘)

Input(name)

Output(‘Hello ’, name)

End

**Year 12**

Main Module

Output(‘What is your name? ‘)

Input(name)

Output(‘Hello ’, name)

EndModule

### Modularisation

Modularisation is the process of breaking down your code into smaller, manageable sections of code in order to increase the efficiency and allows for better teamwork on a project.

The following code demonstrates the use of Modularisation.

**Without Modules**

Main Module

output(‘Please enter a temp in fahrenheit:’)

input(fahrenheit1)

celsius1 🡨 (farenheit1-32)\*5/9

output(‘Please enter a temp in fahrenheit:’)

input(fahrenheit2)

celsius2 🡨 (fahrenheit2-32)\*5/9

    celsiusAverage 🡨 (celsius1+celsius2)/2

End Module**With Modules**

Module GetCelsius (celsius)

output(‘Please enter a temp in fahrenheit:’)

input(fahrenheit)

celsius 🡨 (fahrenheit-32)\*5/9

End Module

Main Module

Call GetCelsius (celcius1)

Call GetCelsius (celcius2)

celsiusAverage 🡨 (celsius1+celsius2)/2

End Module

In the original code without modules, there is repetition of code when the algorithm performs the same three lines on two occasions to convert two values from Fahrenheit to Celsius

To reduce repetition, we place the three lines that are being repeated into its own Module *GetCelsius.* This new module is called in the main module using the keyword *Call*. Each time it is called in the main module, the code in Module GetCelsius is run. The value is calculated and then passed back to the main module through the parameter (Celsius). It is then assigned to a local variable celcius1 in the first line called and celcius2 when it is called in the second line. This solution increases efficiency, reduces duplication and increases scalability for our program.

The benefits of modularisation are as follows:

* Makes code more efficient by reducing redundancies
* Allows code to be easier to read by separating into named modules rather than all being in the main module
* Allows members of a programming team to work on separate parts of the program at the same time
* Reduces the scope of variables which allows for easier understanding of assigned values

**Modules and Functions**

Modules and Functions are both instances of modularisation. A function performs a very similar role to what we have seen so far but there are some clear differences between the two structures.

**Module**

* Must be given an identifier name
* Can return none, one or many values
* Formal parameters may include both input and output parameters
* Value(s) returned through parameters
* Must use keyword ‘Call’ when calling a module

**Function**

* Must be given an identifier name
* MUST return one value
* Formal parameters ONLY include input parameters
* Value is returned through the function name
* DO NOT use ‘Call’ when calling a function but retuned value must be utilised\*\*

The following algorithm to calculate the power for a base and exponent has been demonstrated using a module and a function. This is possible as there is only one result needed from this algorithm. Both a function and a module are capable of returning one value.

Module CalculatePower (cp\_base, cp\_exponent, cp\_result)

cp\_result 🡨 1

for count 🡨 1 to cp\_exponent

cp\_result 🡨 cp\_base \* cp\_result

end for

End Module

Main Module

input(base)

input(exponent)

**Call CalculatePower (base, exponent, result)**

output(base,‘ to the power of ‘,exponent,’ equals ‘, result)

End ModuleFunction CalculatePower (cp\_base, cp\_exponent)

CalculatePower 🡨 1

for count 🡨 1 to cp\_exponent

CalculatePower 🡨 cp\_base \* CalculatePower

end for

End Module

Main Module

input(base)

input(exponent)

**result 🡨 CalculatePower (base, exponent)**

output(base,‘ to the power of ‘,exponent,’ equals ‘, result)

End Module

In the module example, the module is called directly from the main module using the keyword *Call.* When calling a module from the main module, we pass variables into the other module (in this case Module *CalculatePower*) through parameters. The parameters in the bold line in our main module have both sending parameters (*base* and *exponent*) and a receiving parameter (*result*). When the module is called in the main module (the bold line), we send through values for *base* and *exponent*. These values pass to *CalculatePower* and are then use the code in that module to calculate the answer. The answer is allocated to the parameter *cp\_result* which is transferred back to the main module and into the local variable *result*.

The function works slightly different. The function does not return any values through parameters, the value is returned through the function name. If we look at the highlighted line in the function example the answer is returned through the function name – *CalculatePower(base, exponent).* The two values (*base* and *exponent*) are passed through to the function name similar to the module example, but in functions we only use sending parameters and no receiving parameters. As such, the two parameters are sent through to the function for calculation. The function completes the calculation and then assigns the answer to its own name - *CalculatePower 🡨 cp\_base \* CalculatePower*. This is then returned to the main module through its own name *CalculatePower(base, exponent).* In our example, the local variable *result* is assigned the value that has been calculated from the function - *result 🡨 CalculatePower (base, exponent).*

### Structure Chart

A structure chart is used to show a diagram of how parameters are passed back and forth between the main module and any other modules for functions in a program.

The following program inputs two values and returns a 10% discount on the larger of the two values.

Module CalculateDiscount (cd\_bigValue, cd\_discount, cd\_total)

cd\_discount 🡨 cd\_bigValue \* 0.1

cd\_total 🡨 cd\_bigValue – cd\_discount

End Module

A close up of a map

Description automatically generatedFunction IsBigger(ib\_num1, ib\_num2)

If (ib\_num1 > ib\_num2) then

IsBigger 🡨 True

Else

IsBigger 🡨 False

End If

End Function

Module GetValues (gv\_value1, gv\_value2)

Input(gv\_value1)

Input(gv\_value2)

End Module

Main Module

Call GetValues(value1, value2)

If IsBigger(value1, value2) then

bigValue 🡨 value1

Else

bigValue 🡨 value2

End If

Call CalculateDiscount (bigValue, discount, total)

output(‘The largest order $,’ bigValue,’ has been discounted by $‘, discount,’.’)

output(‘The discounted price is $’, total)

End Module

## Cisco Network Diagram

### CISCO Icons

|  |  |  |  |
| --- | --- | --- | --- |
| Router |  | IP Phone |  |
| Switch |  | Notebook/Laptop |  |
| Firewall |  | Desktop computer |  |
| Modem |  | Mobile phone |  |
| Wireless Access Point |  | Handheld/Tablet |  |
| Bridge  A bridge separates two parts of the same network into multiple network segments to reduce collisions. |  | Network Printer |  |
| Repeaters |  | File server |  |
| Wireless Router  NOTE: This device DOES NOT have a built-in modem. It is a combination of a router and a wireless access point. |  | Cloud |  |

### Types of Routers

There are two distinct types of routers available –

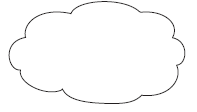
|  |  |
| --- | --- |
| Home Router | Corporate Router |
|  |  |
| These devices are what we usually use to connect to the internet in our homes. While we nearly always call these “routers”, they are a combination of a number of networking devices. These consumer devices may vary in their functions, but most include functionality for the following networking devices:   * Modem (not all include a modem) * Router * Firewall * Switch * Wireless access point (WAP)   This device is represented by the following CISCO icon. The icon refers to a wireless router without an inbuilt modem so we must therefore include a modem icon.    We can also represent this in a diagram including all the inbuilt devices (sometimes with a line around them all to show they are grouped in one device). | Corporate routers are used by businesses, corporations and ISPs. These are often situated on the edge of the corporate network and may provide a connection to the WAN. They may also be used in a corporate network if the corporate network consists of more than one network.  These devices are represented using a single CISCO icon.    These routers usually only have limited number of ports (usually ethernet or fibre). As such they are not designed to connect directly to end devices (like printers, computers, servers, WAPs etc.). They nearly always connect to a switch that then connects to the end devices. |

### Order of Devices

The order of devices in a network is as follows:

1. Cloud (WAN / Internet)
2. Modem (optional – see below)
3. Router / firewall
4. Firewall / router
5. Switch
6. Devices / WAP / Servers

**Cloud**

When demonstrating a wide area network (WAN) in a Cisco diagram, we do not generally show all of the devices that exist in the WAN as it is generally unimportant. The most common WAN used is the internet and this is way too complex to show. As such, the WAN is represented using the cloud icon.

**Modem**

Modems are used to connect to the internet in many situations and should be the first Cisco icon to connect to the cloud. However, there are situations where a modem is not required. This all depends on the internet connection type being used.

Internet connections that travel (even a small bit) over a copper line require a modem. The copper line is usually the copper telephone line (PSTN) but can also be copper coaxial cable used for cable networks.

The following table represents the internet connection types and their corresponding type of modem used.

|  |  |
| --- | --- |
| **Internet Type** | **Modem** |
| Dialup | Dialup modem |
| Asynchronous Digital Subscriber Line (ADSL) | ADSL Modem |
| Fibre to the Node (FTTN) | VDSL2 Modem |
| Cable (HFC) | Cable Modem |

All these internet connections require a modem to modulate and demodulate the signal from digital to analogue. As such you MUST include a modem icon in your network diagrams connected through to the WAN. All the modems above are represented using the same Cisco icon pictured to the right.

**Router / Firewall**

The firewall and router are interchangeable in their position in the network. The firewall can appear between the modem and router or between the router and switch. The most common configuration is to have the router connecting to the modem, followed by the firewall, then the switch.

**Router**

The router routes traffic between two (or more) different networks. In the most common form, they allow an internal local area network (LAN) to connect to an external wide area network (WAN), such as the internet. Routers analyse each inbound packet and checks the destination IP address. If the router knows the network the IP address belongs to, it will forward the packet out one of its ports destined for that network.

Corporate routers generally have a limited number of ports and as such you generally **DO NOT** connect end devices to the router – including workstations, wireless access points or server. The router connects to a switch (often with a firewall in between) and the switch then provides a large array of ports for these end devices to connect to.

Home routers, however, include a switch in the device so end devices may be connected directly to a home router.

**Firewall**

The firewall is a device to restrict inbound and outbound network traffic based on the type of packet. Firewalls may be software based or hardware based but if being included on a network diagram we are representing a hardware-based solution. Firewalls block traffic based on the port number associated with each packet. This port number determines the type of packet. An HTTPS request is based on port # 443 and would be allowed inbound access. Port 23 (Telnet) is commonly blocked as an inbound port to restrict anyone trying to login using Telnet from outside of the LAN).

**Switch**

A switch allows multiple wired networking devices to connect to it. Switches have a set number of ports and may use a variety of wired ports such as Ethernet (RJ45 connections), Fibre optic or a combination of these.

End devices are generally plugged into one of the ports of a switch. This includes wired workstations, servers and wireless access points (WAPs). Switches can also connect to other switches to provide more ports for end devices. In a corporate structure they are often installed in a hierarchical structure with one fast switch connecting to the router and then that fast switch providing multiple connections to additional switches in various locations around the organisation. These switches then then provide access to end devices.

Switches are smart devices. Hubs were the devices used prior to switches and, while they looked the same as a switch, they were less efficient. If 20 devices were connected to a hub, if a device wanted to send a packet to another device on the network the packet would travel into the hub and then the hub would broadcast it to the other 19 ports other than the sending device. The 19 end devices would look at the packet and if it wasn’t destined for the devices, the packet would be dropped. The one it was destined for would receive the packet correctly. Switches work on the process of developing a switching table which learns which port each end device uses. Once it has built this switching table, it can then directly send packets to the appropriate ports rather than flooding the network.

Unlike routers, generally, switches are not interested in the IP address of the packet. They direct traffic based on the MAC address of the packet. They read in the destination MAC address and then send it out the appropriate port to the device with that MAC address.

**WAP**

Wireless access points (WAPs) connect directly to one of the ports in the switch to provide wireless communication to wireless devices on the network.



**Wireless End Devices**

Any device with a wireless NIC such as notebook computers, mobile phones and handheld/tablet devices can connect to the network through the WAP. Once the data travels from the wireless device to the WAP it then travels through the wired network as it continues onto its destination.

**End Devices / Servers**

All these devices connect to a switch – not to a modem, firewall or router. End devices include desktop computers, notebook computers with a wired NIC, network printers, IP phones and servers. These devices connect to the switch using either UTP/STP or Fibre optic cables.



### Complete Cisco Network Diagram

Fred’s Seafood have decided to setup a new office in Sydney to supplement their existing Perth office. Both offices will be able to communicate with each other through a VPN.

The Perth office consists of one PC in the manager’s office as well as a PC setup on the counter used as a point of sale machine to take orders. They also have a web server on-site and a wired network printer.

The Sydney office has decided to implement a wireless solution whereby all staff will connect to the network through a wireless access point. There will however be a wired network printer in one of the offices.

Both offices connect to the internet through a FTTN internet connection.

Chart, box and whisker chart

Description automatically generated