DegreeLive

Bachelor of Commerce (B2001) | Level 2 • Part B Major | Monash University

Accounting Information Systems

Course Code: ACC2400 • Module 1 Notes

Module 1: AIS Architecture, Business Processes, Data Modelling & Relational Databases

1.1 Foundations of Accounting Information Systems and Value Creation

An Accounting Information System (AIS) is a structured, socio-technical system that collects, records, stores, processes, and communicates financial and non-financial data to produce reliable information for organisational decision-making, stewardship, and statutory compliance. In contemporary Australian enterprises operating under the *Corporations Act 2001 (Cth)* and Australian Accounting Standards Board (AASB) pronouncements, the AIS is not merely an automated general ledger; it is the transactional nervous system linking operational business events to external financial reporting and internal managerial control.

To analyse how an AIS generates organisational value, accountants distinguish between raw **data** and decision-useful **information**. Data are unprocessed facts collected from business activities, such as an individual sales order timestamp, a stock-keeping unit (SKU) identifier, or a unit price of $145.00. Information is data that has been validated, classified, aggregated, and contextualised so that it reduces uncertainty and improves the expected payoff of a decision. Under the AASB Conceptual Framework for Financial Reporting, accounting information achieves utility when it possesses the fundamental qualitative characteristics of *relevance* and *faithful representation*, reinforced by the enhancing characteristics of *comparability*, *verifiability*, *timeliness*, and *understandability*.

The Six Core Components of an AIS

Every functioning AIS—whether deployed on-premises by an ASX-listed manufacturer or operated through cloud software by a Melbourne professional services firm—comprises six interdependent components:

- **People:** Users who capture transactions, maintain master records, authorise economic commitments, and interpret outputs, including accounts clerks, controllers, internal auditors, and operational managers.
- **Procedures and Instructions:** Both manual and automated rules governing how transactions are initiated, validated, recorded, and reconciled across business cycles.
- **Data:** Structured tables, event logs, and unstructured supporting artefacts (such as supplier tax invoices and shipping manifests) describing the organisation's economic activities.
- **Software:** Application programs, database management systems (DBMS), enterprise resource planning (ERP) modules, and reporting engines that transform stored data into ledgers and analytical views.
- **Information Technology Infrastructure:** Computing hardware, cloud servers, local and wide-area networks, point-of-sale terminals, and peripheral capture devices.
- **Internal Controls and Security Measures:** Preventive, detective, and corrective safeguards designed to protect data integrity, asset custody, and system availability against error or malicious compromise.

degreelive

Value Chain Positioning and the Economics of Information

Using Michael Porter's value chain framework, an AIS supports both *primary activities* (inbound logistics, operations, outbound logistics, marketing and sales, and customer service) and *support activities* (firm infrastructure, human resource management, technology development, and procurement). Specifically, accounting belongs to firm infrastructure, yet its transaction capture engines sit directly inside inbound logistics, operations, and outbound sales. When an AIS integrates operational events directly with financial posting, organisations eliminate duplicate data entry, compress order-to-cash cycle times, and improve inventory turnover.

Because information is costly to acquire, store, and verify, system designers evaluate AIS investments using information economics. The net economic value of an enhanced AIS signal equals the expected decision payoff with the information minus the expected payoff without it, net of the incremental cost of producing the information:

$$V_I = E[U \mid I] - E[U \mid I_0] - C_I$$

Where: V_I is the net economic value of the new information signal (in AUD); $E[U \mid I]$ is the expected monetary payoff when management acts with the improved AIS information; $E[U \mid I_0]$ is the expected monetary payoff under baseline (prior) information; and C_I is the incremental cost of acquiring, processing, and maintaining the information signal.

Worked Example 1.1: Expected Value of Real-Time Inventory Information

A Sydney wholesale distributor, Southern Cross Supplies Pty Ltd, sells industrial components and faces uncertain monthly demand. Without real-time AIS stock-level alerts, management orders standard replenishment batches yielding an expected monthly contribution margin of $182,000. Management is evaluating an automated barcode-integrated AIS module that provides daily reorder alerts. Under the new system, expected monthly contribution rises to $196,500 due to fewer stockouts and lower emergency freight charges. The cloud subscription and maintenance cost of the module is $6,200 per month.

1. **Identify baseline and informed payoffs:** $E[U \mid I_0] = 182{,}000$ AUD and $E[U \mid I] = 196{,}500$ AUD.
2. **Compute gross value of information:** $196{,}500 - 182{,}000 = 14{,}500$ AUD per month ($14,500).
3. **Subtract incremental system cost:** $V_I = 14{,}500 - 6{,}200 = 8{,}300$ AUD net monthly gain ($8,300/month, or $99,600 annually).
4. **Decision rule:** Because $V_I > 0$, adopting the real-time inventory signal is economically justified.

Information Attribute	Operational Definition in AIS	Failure Consequence in Practice
Relevance	Capable of influencing user resource-allocation or control decisions through predictive or confirmatory value.	Information overload obscures overdue receivables and delays credit holds on defaulting customers.

Degreelive

Information Attribute	Operational Definition in AIS	Failure Consequence in Practice
Faithful Representation	Complete, neutral, and free from material error in depicting underlying economic events.	Unrecorded supplier liabilities understate expenses and breach AASB 101 presentation standards.
Timeliness	Available to decision-makers before the opportunity to intervene expires.	Monthly batch updates allow over-limit sales to occur before credit exposure is flagged.
Verifiability	Independent observers can trace aggregated balances back to authentic source events.	External auditors cannot substantiate revenue cut-off at 30 June, risking a modified audit opinion.
Accessibility	Authorised personnel can retrieve formatted views without technical bottlenecks.	Managers export unverified data to ad-hoc spreadsheets, introducing shadow-IT formula errors.

1.2 Business Process Documentation: DFDs, Flowcharts, and BPMN

Under Australian auditing standards (ASA 315 *Identifying and Assessing the Risks of Material Misstatement*), accountants and auditors must obtain and document a rigorous understanding of how transactions originate, flow through IT applications, and enter the general ledger. Three complementary graphical techniques dominate AIS practice: Data Flow Diagrams (DFDs), System Flowcharts, and Business Process Model and Notation (BPMN 2.0).

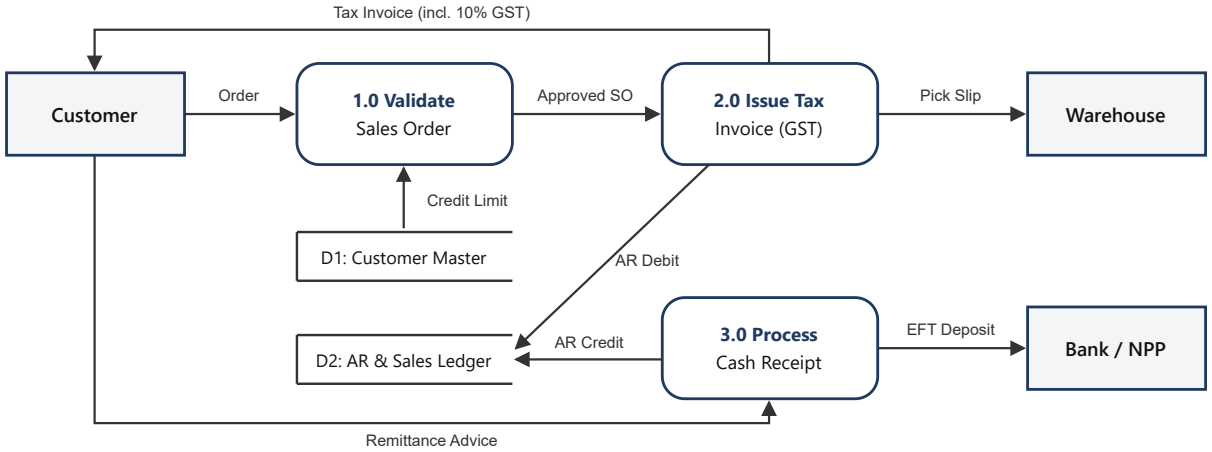
Data Flow Diagrams (DFDs)

A Data Flow Diagram depicts the logical flow of data through an organisation, emphasizing *what* transformations occur rather than *who* executes them or *which* physical media carry them. DFDs employ exactly four standard symbols: external entities (squares, representing data sources or destinations outside the system boundary), processes (rounded rectangles or circles, transforming incoming data into outgoing data), data flows (labelled directed arrows), and data stores (open-ended rectangles, representing repositories at rest).

DFDs are structured hierarchically to manage complexity. The highest level is the **Context Diagram**, which contains a single process bubble representing the entire system surrounded by external entities and major boundary flows. Decomposing the context bubble yields a **Level-0 DFD** showing major subprocesses (e.g., 1.0 Validate Order, 2.0 Fulfil Delivery, 3.0 Bill Customer, 4.0 Collect Cash), and each numbered process can be further expanded into Level-1 and Level-2 DFDs. Crucially, parent and child diagrams must remain *balanced*: every data flow crossing the boundary of a parent process must appear identically on its child decomposition.

DegreeLive

Figure 1.1: Logical Level-0 Data Flow Diagram (Revenue Cycle)



System Flowcharts and Document Flowcharts

While DFDs abstract away physical implementation, **Document Flowcharts** and **System Flowcharts** explicitly map physical media, organisational departments, and control checkpoints. A Document Flowchart uses vertical columns (departments or roles) to trace physical or digital documents from origination to archival, making it ideal for evaluating *segregation of duties*. A System Flowchart extends this perspective across IT components—identifying terminal entry, batch validation scripts, magnetic/cloud disk storage, and printed exception reports.

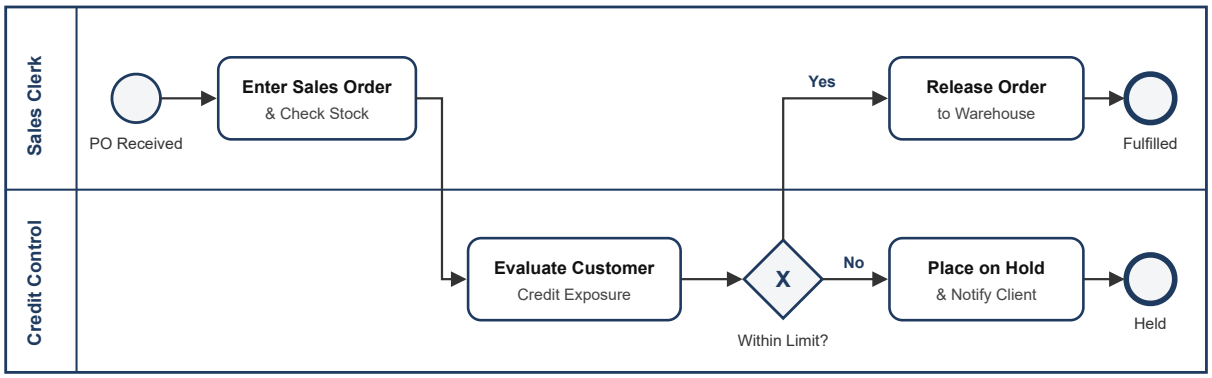
Business Process Model and Notation (BPMN 2.0)

BPMN 2.0 has become the international industry standard (ISO/IEC 19510) for documenting end-to-end enterprise workflows because it bridges business analysis and executable ERP workflow engines. BPMN models organise activities inside **Pools** (independent organisations or systems) partitioned into **Lanes** (internal departments, roles, or software modules). Flows progress chronologically from left to right using four core element families:

- **Flow Objects:** *Events* (circles: thin-bordered Start, double-bordered Intermediate, thick-bordered End), *Activities* (rounded rectangles representing atomic tasks or collapsible subprocesses), and *Gateways* (diamonds controlling branching and merging, such as exclusive XOR X, parallel AND +, or inclusive OR 0).
- **Connecting Objects:** Solid sequence flows within a pool, dashed message flows between distinct pools, and dotted associations linking data artefacts.
- **Swimlanes:** Pools and Lanes clarifying departmental accountability and internal control boundaries.
- **Artefacts:** Data objects (representing records such as an electronic tax invoice) and text annotations.

DegreeLive

Figure 1.2: BPMN 2.0 Swimlane Model — Sales Order & Credit Approval



Documentation Method	Primary Analytical Focus	Key Symbols Used	Optimal Audit / Design Use Case
Data Flow Diagram (DFD)	Logical transformation and movement of data independent of technology.	External entity, process, data flow, data store.	Scoping system boundaries and designing database inputs/outputs.
Document Flowchart	Physical document routing across organisational departments.	Department columns, document, manual operation, file, connector.	Evaluating segregation of duties and physical paper-trail controls.
System Flowchart	Hardware, software, batch/online processing, and storage media.	Display, manual input, process, magnetic/cloud storage, report.	Assessing IT general and application controls in automated jobs.
BPMN 2.0 Diagram	Chronological sequence, role accountability, and exception branching.	Pools, lanes, events, activities, XOR/AND gateways, message flows.	Process re-engineering, ERP workflow configuration, and ASA 315 walkthroughs.

1.3 Conceptual Data Modelling: ER Diagrams and the REA Ontology

Before an organisation creates database tables, analysts construct a conceptual data model representing the entities, attributes, and business rules of the domain. Two complementary modelling paradigms govern AIS design: general **Entity-Relationship (ER) modelling** and the accounting-specific **Resources-Events-Agents (REA) ontology** developed by William E. McCarthy (1982).

Entity-Relationship (ER) Modelling and Cardinality

An ER diagram identifies *entities* (distinct things or occurrences about which data are stored), *attributes* (properties describing each entity), and *relationships* (meaningful associations among entities). In accounting databases, precision depends on specifying **cardinalities**—the minimum and maximum number of instances of one entity that can associate with a single instance of a related entity, written as (min, max) pairs.

- **One-to-One (1:1):** Maximum cardinality is 1 on both sides (e.g., one corporate tax invoice corresponds to at most one electronic Peppol dispatch record).
- **One-to-Many (1:N):** One parent instance associates with zero, one, or many child instances, whereas each child associates with at most one parent (e.g., one Customer places $0..N$ Sales, and each Sale

belongs to $(1, 1)$ Customer).

- **Many-to-Many (M:N):** Maximum cardinality is N on both sides (e.g., one Sale includes $1..N$ Inventory Items, and each Inventory Item appears on $0..N$ Sales). In relational databases, M:N relationships cannot be implemented directly and must be resolved via an associative (junction/bridge) entity such as `Sale_Line_Item`.

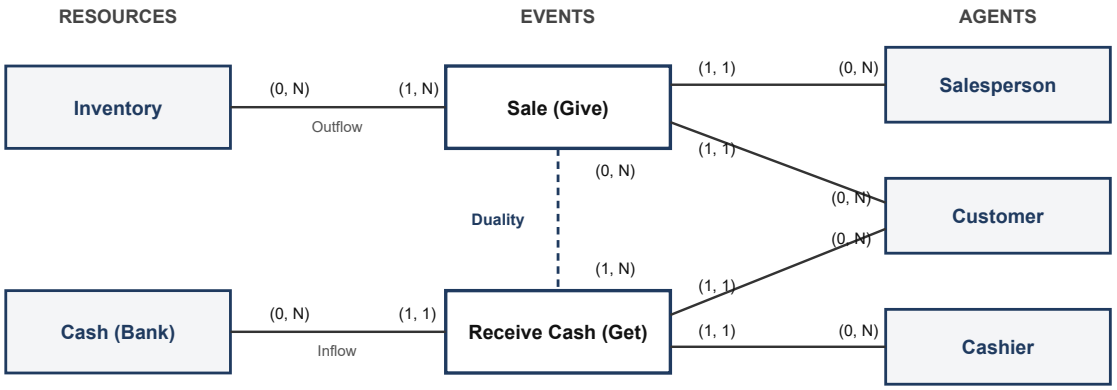
The REA Data Model (Resources, Events, Agents)

Traditional double-entry bookkeeping stores only monetary summaries (account balances), stripping away operational details such as salesperson identity, delivery time, or warehouse location. The **REA ontology** overcomes this limitation by modelling economic reality directly around three fundamental entity classes:

1. **Resources (R):** Identifiable economic assets that have scarcity and value to the enterprise, such as `Inventory`, `Cash`, and `Plant_Equipment`.
2. **Events (E):** Time-stamped business activities that increase or decrease resources. Events fall into two subclasses: *economic events* that alter resource quantities (e.g., `Sale`, `Receive_Cash`, `Purchase_Inventory`) and *commitment events* that precede economic exchanges (e.g., `Customer_Order`, `Purchase_Order`).
3. **Agents (A):** Internal employees and external counterparties (`Customer`, `Supplier`, `Salesperson`, `Cashier`) who participate in events and assume accountability.

Every REA diagram obeys strict structural pattern rules. First, each economic event connects to at least one resource via a **stockflow relationship** (giving or taking a resource). Second, every economic event links to at least two agents via **participation relationships**—one internal employee accountable for the transaction and one external counterparty. Third, every resource-decreasing ("give") event must pair with a corresponding resource-increasing ("get") event through an **economic duality relationship** (for example, `Sale` pairs with `Receive_Cash`, and `Receive_Inventory` pairs with `Disburse_Cash`).

Figure 1.3: REA Conceptual Diagram — Revenue Cycle with (min, max) Cardinalities



Notice how the REA structure naturally derives financial statements without requiring separate journal tables. At any cut-off date, Accounts Receivable is simply the cumulative value of `Sale` events minus cumulative matched `Receive_Cash` events, while Inventory on hand equals cumulative purchase inflows less cumulative sale outflows.

1.4 Relational Database Design, Integrity Rules & Normalisation (1NF–3NF)

Once the conceptual REA or ER model is verified, it is mapped into a physical **relational database** governed by Edgar F. Codd's relational model. In a relational DBMS (such as PostgreSQL, SQL Server, or Oracle), data are stored in two-dimensional tables (relations) consisting of rows (tuples) and columns (attributes).

Keys and Relational Integrity Constraints

- **Primary Key (PK):** An attribute or minimal set of attributes that uniquely identifies every row in a table (e.g., `Invoice_No` in `Sales`). Under the *Entity Integrity Rule*, no component of a primary key may ever contain a `NULL` value or duplicate value.
- **Foreign Key (FK):** An attribute in one table that references the primary key of another (or the same) table to establish a relational link (e.g., `Customer_ID` in `Sales` referencing `Customer_ID` in `Customers`). Under the *Referential Integrity Rule*, every non-null foreign key value must match an existing primary key value in the referenced parent table, preventing "orphan" transactions such as recording a sale to a non-existent customer.
- **Composite Key:** A primary key formed by combining two or more attributes, typically used in associative bridge tables created to resolve M:N relationships (e.g., `(Invoice_No, SKU_ID)` in `Sale_Line_Items`).

Poorly structured tables suffer from three classic **data anomalies**:

1. **Update Anomaly:** Changing one fact (such as a customer's Melbourne street address) requires editing dozens of invoice rows; missing even one row leaves contradictory addresses in the database.
2. **Insert Anomaly:** Inability to record a valid new entity (such as a newly approved supplier) until a transaction event occurs, because the supplier fields live inside a purchase transaction table whose primary key requires a `PO_No`.
3. **Delete Anomaly:** Deleting a single transaction inadvertently destroys the only stored record of an underlying master entity (such as erasing the sole invoice for a regional customer and losing their ABN and contact details).

Step-by-Step Normalisation from UNF to 3NF

Normalisation is the formal mathematical process of decomposing unnormalised relations into smaller, well-structured tables to eliminate redundancy and anomalies while preserving lossless joins.

- **First Normal Form (1NF):** Eliminate repeating groups so every cell holds a single atomic value, and designate a primary key.
- **Second Normal Form (2NF):** The table must be in 1NF, and every non-key attribute must be *fully functionally dependent* on the entire primary key (eliminating *partial dependencies* where an attribute depends on only part of a composite key).
- **Third Normal Form (3NF):** The table must be in 2NF, and no non-key attribute may be *transitively dependent* on the primary key through another non-key attribute (every non-key attribute must depend strictly on "the key, the whole key, and nothing but the key").

DegreeLive

Worked Example 1.2: Normalising an Australian Wholesale Invoice Register (UNF to 3NF)

Consider a flat spreadsheet register maintained by Yarra Hardware Distributors Pty Ltd containing repeating line items inside each invoice:

Inv_No	Inv_Date	Cust_ID	Cust_Name	Cust_ABN	SKU_List (Repeating Group)	Item_Desc	Qty	Unit_Price
INV-401	2027-03-10	C-108	Bayside Builders	48 123 456 789	SK-11, SK-25	Timber Beam, Steel Bracket	10, 40	$45.00, $8.50
INV-402	2027-03-11	C-109	Geelong Civil	91 987 654 321	SK-11	Timber Beam	25	$45.00

Step 1 — Achieve 1NF (Remove Repeating Groups): Flatten multi-valued cells so each row represents one invoice line, forming a composite key `(Inv_No, SKU_ID)`. Now `Item_Desc`, `Qty`, and `Unit_Price` sit atomically, but anomalies persist because customer details repeat for every line.

Step 2 — Achieve 2NF (Eliminate Partial Dependencies): Inspect dependencies against the composite PK `(Inv_No, SKU_ID)`:

- `Inv_Date`, `Cust_ID`, `Cust_Name`, and `Cust_ABN` depend only on `Inv_No` (partial dependency).
- `Item_Desc` and standard `Catalog_Price` depend only on `SKU_ID` (partial dependency).
- `Qty` and historical `Unit_Price_Sold` depend on the full composite key `(Inv_No, SKU_ID)`.

Decomposing produces three 2NF relations: `Invoices(Inv_No, Inv_Date, Cust_ID, Cust_Name, Cust_ABN)`, `Inventory(SKU_ID, Item_Desc, Catalog_Price)`, and `Invoice_Lines(Inv_No, SKU_ID, Qty, Unit_Price_Sold)`.

Step 3 — Achieve 3NF (Eliminate Transitive Dependencies): In the 2NF `Invoices` table, `Cust_Name` and `Cust_ABN` depend on `Cust_ID`, which in turn depends on `Inv_No` ($Inv_No \rightarrow Cust_ID \rightarrow \{Cust_Name, Cust_ABN\}$). Moving customer attributes to a dedicated table yields four clean 3NF relations:

- `Customers(Cust_ID, Cust_Name, Cust_ABN)`
- `Invoices(Inv_No, Inv_Date, Cust_ID)` — where *Cust_ID* is FK referencing `Customers`
- `Inventory(SKU_ID, Item_Desc, Catalog_Price)`
- `Invoice_Lines(Inv_No, SKU_ID, Qty, Unit_Price_Sold)` — where *Inv_No* and *SKU_ID* are FKs

Notice an important accounting nuance: `Unit_Price_Sold` remains inside `Invoice_Lines` rather than being looked up solely from `Inventory.Catalog_Price`. Because catalog prices change over time, storing the historical transaction price in `Invoice_Lines` preserves faithful representation of past revenue under AASB 15.

DegreeLive

1.5 Structured Query Language (SQL) Foundations for Accounting

Relational databases are interrogated and enforced through **Structured Query Language (SQL)**, partitioned into Data Definition Language (DDL) for schema creation and Data Manipulation Language (DML) for querying and updating records. Accountants use SQL both to design integrity constraints at table creation and to extract audit-ready financial summaries directly from transaction tables.

DDL Schema Definition with Integrity Constraints

The following DDL statements translate our 3NF Australian revenue schema into executable constraints, enforcing positive prices and valid 11-digit Australian Business Numbers (ABNs):

```
CREATE TABLE Customers (
    Cust_ID     VARCHAR(10) PRIMARY KEY,
    Cust_Name   VARCHAR(100) NOT NULL,
    Cust_ABN    CHAR(11) NOT NULL UNIQUE
);

CREATE TABLE Invoices (
    Inv_No      VARCHAR(12) PRIMARY KEY,
    Inv_Date    DATE NOT NULL,
    Cust_ID     VARCHAR(10) NOT NULL,
    FOREIGN KEY (Cust_ID) REFERENCES Customers(Cust_ID)
);

CREATE TABLE Invoice_Lines (
    Inv_No          VARCHAR(12),
    SKU_ID          VARCHAR(10),
    Qty             INTEGER NOT NULL CHECK (Qty > 0),
    Unit_Price_Sold DECIMAL(10,2) NOT NULL CHECK (Unit_Price_Sold >= 0),
    PRIMARY KEY (Inv_No, SKU_ID),
    FOREIGN KEY (Inv_No) REFERENCES Invoices(Inv_No)
);
```

DML Analytical Queries: Joins, Grouping, and GST Extraction

Using `INNER JOIN`, aggregation functions (`SUM`, `COUNT`), `GROUP BY`, and `HAVING`, an accountant can extract total GST-exclusive revenue and 10% Australian GST liability per customer for a quarterly Business Activity Statement (BAS) period:

```
SELECT c.Cust_ID,
       c.Cust_Name,
       COUNT(DISTINCT i.Inv_No) AS Invoice_Count,
       SUM(l.Qty * l.Unit_Price_Sold) AS Net_Sales_Ex_GST,
       ROUND(SUM(l.Qty * l.Unit_Price_Sold) * 0.10, 2) AS GST_Collected
FROM Customers c
INNER JOIN Invoices i ON c.Cust_ID = i.Cust_ID
INNER JOIN Invoice_Lines l ON i.Inv_No = l.Inv_No
WHERE i.Inv_Date BETWEEN '2027-01-01' AND '2027-03-31'
GROUP BY c.Cust_ID, c.Cust_Name
HAVING SUM(l.Qty * l.Unit_Price_Sold) >= 10000.00
ORDER BY Net_Sales_Ex_GST DESC;
```

In contrast to `INNER JOIN` (which returns only matching rows across tables), a `LEFT JOIN` from `Customers` to `Invoices` retains customers with zero transactions during the quarter—an essential audit query for identifying dormant master accounts or potential duplicate customer profiles.

1.6 Corporate Modelling Theory and Effective Spreadsheet Design

Although enterprise transactions reside in relational databases and ERP systems, management accountants routinely extract data into spreadsheet models (such as Microsoft Excel) for budgeting, cash-flow forecasting, sensitivity analysis, and capital appraisal. Empirical studies of corporate spreadsheets consistently reveal high error rates—often stemming from hardcoded constants inside formulas, undocumented logic, and tangled layouts. Corporate modelling theory establishes rigorous architectural rules so spreadsheet models operate as reliable decision-support tools rather than fragile sources of misstatement.

Types, Uses, Benefits, and Limitations of Models

A corporate model is a simplified mathematical representation of an organisation's economic relationships. Depending on purpose, models are classified as **deterministic** (single-point inputs producing one output, such as a static annual budget), **stochastic / probabilistic** (incorporating probability distributions via Monte Carlo simulation to model uncertain commodity prices or AUD exchange rates), or **optimisation** models (using linear programming to maximise contribution subject to capacity constraints).

Models offer three major benefits: they permit low-cost experimentation ("what-if" scenario testing) without risking real capital, enforce explicit assumptions about revenue drivers and cost behaviour, and compress complex multi-period cash flows into comparable metrics such as Net Present Value (NPV). However, models carry strict limitations: *model risk* (erroneous structural assumptions, such as assuming linear variable costs across capacity thresholds), *data-quality dependence* ("garbage in, garbage out"), and *false precision* when managers treat simulated decimal outputs as guaranteed outcomes.

Principles of Effective Spreadsheet Architecture

Adopting structured spreadsheet engineering standards (such as the FAST or ICAEW Twenty Principles) mitigates operational error:

1. **Physical Separation of Inputs, Calculations, and Outputs:** Dedicate distinct worksheets (or clearly demarcated vertical blocks) to *Assumptions/Inputs*, *Processing/Engine*, and *Summary Outputs*. Inputs must never be embedded inside calculation formulas.
2. **Left-to-Right, Top-to-Bottom Flow:** Formulas should reference cells only above or to the left, ensuring deterministic auditability and preventing circular reference traps.
3. **Consistent Formula Structure:** Across time-series columns (e.g., Q1–Q4), use one uniform formula copied horizontally rather than bespoke cell-by-cell overrides.
4. **Built-in Integrity & Error Checks:** Include automated cross-footing checks (e.g., `=IF(ABS(Assets - (Liabilities + Equity)) < 0.01, "OK", "ERROR")`) rolled up to a master control flag on the cover sheet.
5. **Documentation and Version Control:** Maintain an assumption register citing sources, units (e.g., AUD thousands), colour conventions (blue font for hardcoded inputs, black for same-sheet formulas, green for cross-sheet links), and change logs.

DegreeLive

Worked Example 1.3: Corporate Break-Even and Operating Leverage Model Audit

Melbourne MedTech Pty Ltd builds a spreadsheet model to evaluate launching a diagnostic device. Fixed manufacturing and compliance overheads are $F = 420{,}000$ AUD per year ($420,000), unit selling price is $P = 350$ AUD ($350), and unit variable cost is $V = 200$ AUD ($200). Management plans baseline sales of $Q = 3,600$ units. We verify the break-even volume, baseline operating profit (EBIT), and Degree of Operating Leverage (DOL):

$$Q_{BE} = \frac{F}{P - V}$$

$$\text{DOL} = \frac{Q(P - V)}{Q(P - V) - F}$$

Where: Q_{BE} is the break-even sales volume in units; F is total annual fixed cost (AUD); P is unit selling price excluding GST (AUD); V is unit variable cost (AUD); Q is planned annual sales volume in units; and *DOL* is the Degree of Operating Leverage measuring the percentage change in EBIT for a 1% change in unit volume.

1. **Unit contribution margin:** $P - V = 350 - 200 = 150$ AUD per unit ($150).
2. **Break-even volume:** $Q_{BE} = 420{,}000/150 = 2,800$ units (representing $980,000 in GST-exclusive revenue).
3. **Baseline EBIT at 3,600 units:** $\text{EBIT} = 3,600(150) - 420{,}000 = 540{,}000 - 420{,}000 = 120{,}000$ AUD ($120,000).
4. **Degree of Operating Leverage:** $\text{DOL} = 540{,}000/120{,}000 = 4.50$.
5. **Sensitivity verification:** If unit demand falls by 10% (to 3,240 units), EBIT falls by $10\% \times 4.50 = 45\%$, dropping from $120,000 to $66,000 ($3,240 \times 150 - 420{,}000 = 66{,}000$ AUD). By separating F, P, V, and Q into dedicated input cells, the model recalculates exact profit sensitivity cleanly without manual formula rewrites.

1.7 Translating REA Models into Relational Schemas & Australian Statutory Record-Keeping

Although the REA diagram captures the semantic architecture of business events, database engineers must translate that diagram into a set of normalised relational tables before deploying an operational AIS. Applying deterministic REA-to-relational mapping rules guarantees that the physical database remains in Third Normal Form while preserving complete audit traceability across resources, events, and agents.

Deterministic REA-to-Relational Mapping Rules

Converting an REA conceptual model into a relational schema follows three mechanical rules:

1. **Rule 1 — Create One Table per Distinct Entity:** Every resource, event, and agent rectangle in the REA diagram becomes its own relational table, assigned an immutable primary key (such as `SKU_ID`,

`Sale_ID`, `Receipt_ID`, `Cust_ID`, or `Emp_ID`) and descriptive non-key attributes.

2. **Rule 2 — Resolve M:N Relationships Using Composite Bridge Tables:** Any relationship whose maximum cardinality is N on both sides (for instance, the stockflow relationship between `Inventory` and `Sale`, or the duality relationship between `Sale` and `Receive_Cash` when customers pay multiple invoices in one lump sum or settle one invoice in instalments) requires a separate associative table. The primary key of the bridge table is the concatenation of both participating entities' primary keys, supplemented by relationship-specific attributes such as `Qty_Sold`, `Unit_Price_Sold`, or `Amount_Applied`.
3. **Rule 3 — Implement 1:1 and 1:N Relationships via Foreign Keys:** For every 1:N association, place the primary key of the parent entity (the $(0, N)$ or $(1, N)$ side) as a foreign key inside the table on the child $(1, 1)$ side. For example, because each `Sale` involves $(1, 1)$ `Customer` and $(1, 1)$ `Salesperson`, `Cust_ID` and `Emp_ID` are embedded as foreign keys inside the `Sales` table rather than creating extra junction tables.

REA Relationship Pattern	Cardinalities	Physical Relational Implementation	Accounting Rationale
Inventory — Sale (Stockflow)	$(0, N)$ to $(1, N)$	Bridge table `Sale_Inventory(Sale_ID, SKU_ID, Qty_Sold, Unit_Price)`	Captures multi-line invoices and preserves historical transaction prices.
Customer — Sale (Participation)	$(0, N)$ to $(1, 1)$	Post `Cust_ID` as FK inside `Sales`	Links each revenue event to a unique legal debtor for AR aging.
Sale — Receive Cash (Duality)	$(0, N)$ to $(1, N)$	Bridge table `Cash_Application(Sale_ID, Receipt_ID, Amount_Applied)`	Supports part-payments, progress claims, and batch remittance matching.
Cash — Receive Cash (Stockflow)	$(0, N)$ to $(1, 1)$	Post `Bank_Acct_ID` as FK inside `Receive_Cash`	Routes each receipt directly to the nominated Australian ADI bank account.

Relational Algebra, Indexing, and Query Optimisation

In high-volume transaction databases, accountants must understand how the DBMS executes queries over normalised tables. Relational queries rely on three core relational algebra operations: **Selection** (σ, filtering rows via `WHERE`), **Projection** (π, choosing columns via `SELECT`), and **Join** ($\bowtie$, combining related tables across primary-foreign key pairs). Because joining millions of rows across `Invoices` and `Invoice_Lines` via sequential table scans would degrade reporting latency, database administrators build **B-tree indexes** on primary keys, foreign keys, and high-frequency date columns (`Inv_Date`). An index functions like a textbook index, reducing lookup complexity from linear $O(N)$ scanning to logarithmic $O(\log N)$ traversal.

However, indexing involves a deliberate architectural trade-off: while indexes accelerate `SELECT` queries for period-end reporting and auditor data extraction, every `INSERT`, `UPDATE`, or `DELETE` operation must also update associated index trees. Consequently, high-throughput OLTP (Online Transaction Processing) accounting systems maintain lean, targeted indexes on transaction tables and offload heavy multi-year analytical joins to read-replicas or data warehouses.

Statutory Record-Keeping under Australian Law

Database design in Australia is directly constrained by statutory retention and auditability mandates. Under **Section 286 of the *Corporations Act 2001 (Cth)***, every company, registered scheme, and disclosing entity must keep written financial records that correctly record and explain its transactions and financial position and performance, and would enable true and fair financial statements to be prepared and audited. These records—including the underlying relational database logs, tax invoices, and chart-of-accounts mappings—must be retained for **seven years** after the transactions covered by the records are completed.

Similarly, under **Section 382-5 of Schedule 1 to the *Taxation Administration Act 1953 (Cth)*** and Australian Taxation Office (ATO) ruling *TR 2018/2*, taxpayers maintaining electronic accounting systems must preserve digital transaction records for at least **five years** in a form that is readily accessible, intelligible, and convertible into readable English. Crucially, database administrators must ensure that archiving or purging old fiscal periods does not break referential integrity or strip historical tax codes; instead, closed financial periods should be locked with immutable audit flags (`Is_Posted = TRUE`) so corrections occur strictly through traceable reversing or adjusting journal events rather than silent `UPDATE` or `DELETE` statements.

DegreeLive