BAB II

LANDASAN TEORI

2.1 Penelitian Terdahulu

Penelitian terdahulu menjadi landasan penting dalam memahami konsep, metode, serta pendekatan teknis yang relevan dengan perancangan basis data dan integrasi modul dalam *Project Management Information System* (PMIS). Kajian dilakukan untuk memetakan kontribusi ilmiah pada bidang integritas data, normalisasi relasional, integrasi skema, dan metodologi *Database System Development Life Cycle* (DBSDLC), serta memahami konteks digitalisasi sistem pada industri Maintenance, Repair, and Overhaul (MRO). Tabel 2.1 merangkum penelitian yang relevan dengan fokus penelitian ini.

Tabel 2.1 Ringkasan Penelitian Terdahulu yang Relevan

No.	Peneliti & Tahun	Fokus Penelitian	Metode / Teori	Temuan Utama & Relevansi
1	Yesin et al., 2021 [23]	Integritas data relasional	Data Integrity	Menjelaskan prinsip *entity*, *referential*, dan *domain integrity*. Relevan sebagai dasar memastikan schema *Bizcase* bebas inkonsistensi.
2	Olivier, 2023 [24]	Pelanggaran aturan integritas	Integrity Constraint Analysis	Menunjukkan dampak kegagalan FK/PK terhadap anomali data. Penting untuk analisis akar masalah redudansi *Bizcase*.
3	Hammad et al., 2021 [25]	Integrasi skema	Schema Matching	Menguraikan teknik *linguistic* dan *constraint-based matching*. Relevan untuk integrasi Bizreq–*Bizcase*–Risk.
4	Yousfi et al., 2020 [26]	Integrasi multi-skema	Holistic Schema Matching	Menawarkan pendekatan integrasi multi-modul. Mendukung kebutuhan konsolidasi skema PRISMA.

No.	Peneliti & Tahun	Fokus Penelitian	Metode / Teori	Temuan Utama & Relevansi
5	Sug, 2020 [27]	Normalisasi 1NF–3NF	Relational Normalization	Menjelaskan eliminasi *partial* dan *transitive dependency*. Penting untuk perbaikan struktur bc_*.
6	Li et al., 2024 [28]	DB design berbasis FD	Functional Dependency	Menunjukkan pentingnya FD sebagai dasar normalisasi relasional. Mendukung rekonstruksi tabel *Bizcase*.
7	Setiyadi, 2021 [2]	Implementasi DBSDLC	DBSDLC	Memvalidasi DBSDLC sebagai pendekatan terstruktur untuk pengembangan database. Menjadi metodologi utama penelitian.
8	Aminu & Ogwueleka, 2020 [1]	SDLC vs DBSDLC	Comparative Study	DBSDLC dinilai lebih tepat untuk sistem berbasis data dibanding SDLC umum. Menguatkan pemilihan metode.
9	Żyluk et al., 2025 [7]	Digitalisasi MRO	Case Study	Menunjukkan pentingnya integrasi data real-time dalam industri aviasi. Mendukung konteks PRISMA di GMF.
10	Majerik & Borkovcova, 2023 [9]	Akses data menggunakan ORM	ORM Architecture	ORM seperti Prisma mengurangi kompleksitas query dan meningkatkan efisiensi backend. Relevan untuk implementasi penelitian.
11	Vijayakumar et al., 2024 [29]	Prioritas Kebutuhan	MoSCoW Prioritization	Menunjukkan efektivitas teknik prioritas dalam pengembangan sistem. Relevan sebagai dasar adaptasi teknik M-D-I.

Penelitian yang dilakukan oleh *Yesin et al.* dan *Olivier* memberikan dasar teoretis yang kuat terkait integritas data, khususnya *entity integrity* dan *referential integrity*, yang berperan penting dalam menjaga konsistensi dan keakuratan data pada database relasional. Temuan tersebut relevan dengan permasalahan awal pada

modul *Bizcase* yang menunjukkan potensi duplikasi dan ketidaksesuaian relasi antar-entitas[23], [24].

Penelitian *Hammad et al.* dan *Yousfi et al.* berfokus pada integrasi skema, termasuk *schema matching* dan *holistic schema integration*, yang relevan dalam konteks integrasi antar modul PRISMA seperti *Bizreq, Bizcase,* dan *Risk Management[25], [26]*. Sementara itu, *Sug* dan *Li et al.* menegaskan bahwa normalisasi hingga 3NF dan analisis functional dependency merupakan langkah penting untuk mengurangi anomali struktur dan redudansi data[27], [28]. Hal ini mendukung perancangan ulang struktur tabel bc_* dalam penelitian ini.

Dari sisi metodologis, *Setiyadi* dan *Aminu & Ogwueleka* menunjukkan bahwa DBSDLC memberikan tahapan analitis dan desain yang lebih terstruktur dibanding SDLC umum, sehingga lebih sesuai untuk pengembangan sistem berbasis data[1], [2]. Penelitian *Żyluk et al.* memberikan konteks industri terkait digitalisasi dan kebutuhan integrasi data pada sektor MRO[7]. Selain itu, *Majerik & Borkovcova* menguatkan pentingnya penggunaan ORM untuk mempermudah pengelolaan data pada sistem backend berbasis enterprise seperti PRISMA[9].

Selain aspek teknis basis data, kajian literatur juga menyoroti pentingnya manajemen kebutuhan data yang efektif. Penelitian Vijayakumar et al. menegaskan bahwa dalam pengembangan sistem yang kompleks, penerapan teknik prioritas kebutuhan (*requirements prioritization*) seperti metode MoSCoW sangat krusial untuk mencegah pembengkakan ruang lingkup dan memastikan fitur-fitur kritis dapat diselesaikan tepat waktu [29]. Temuan ini menjadi landasan bagi penelitian ini untuk mengadopsi pendekatan prioritas M-D-I (*Mandatory-Desirable-Inessential*) dalam proses seleksi atribut modul *Bizcase*.

Melengkapi tinjauan metodologis tersebut, kajian ini juga meninjau konsep dasar dari sistem yang dikembangkan. Secara umum, *Project Management Information System* (PMIS) didefinisikan sebagai sistem berbasis perangkat lunak yang dirancang untuk membantu manajer proyek dalam merencanakan, melaksanakan, dan memantau perkembangan proyek. Fungsi utama PMIS secara universal adalah menyediakan informasi yang akurat guna mendukung

pengambilan keputusan manajerial, pengelolaan sumber daya, serta penjadwalan waktu (*scheduling*) yang efisien [1].

Di dalam kerangka manajemen proyek, dokumen *Business Case* memegang peranan vital sebagai instrumen justifikasi investasi. Secara teoritis, *Business Case* adalah dokumen yang menyajikan alasan logis di balik inisiatif proyek, mencakup analisis biaya-manfaat (*cost-benefit analysis*), estimasi risiko, dan proyeksi dampak bisnis yang diharapkan. Dokumen ini menjadi dasar validasi apakah suatu proyek layak untuk disetujui atau ditolak oleh manajemen [16].

Namun, penerapan konsep umum tersebut memiliki tantangan tersendiri dalam industri yang sangat tergulasi seperti *Maintenance, Repair, and Overhaul* (MRO) penerbangan. Penelitian Żyluk et al. dan Alharasees et al. menunjukkan bahwa dalam konteks aviasi, PMIS tidak hanya berfungsi sebagai alat administratif, melainkan sebagai infrastruktur kritis untuk menjamin integritas data operasional dan keselamatan[7], [16]. Oleh karena itu, penelitian ini menggunakan definisi umum tersebut sebagai landasan untuk kemudian menganalisis bagaimana konsep PMIS dan *Business Case* perlu diadaptasi secara teknis melalui pendekatan DBSDLC dan normalisasi basis data untuk memenuhi kebutuhan integrasi data yang kompleks di PT GMF AeroAsia Tbk.

Berdasarkan kajian literatur tersebut, penelitian sebelumnya telah membahas integritas data, integrasi skema, normalisasi relasional, serta penerapan DBSDLC pada berbagai konteks. Namun, belum terdapat penelitian yang secara spesifik menerapkan pendekatan tersebut pada modul *Bizcase* dalam lingkungan PMIS industri aviasi. Selain itu, belum ditemukan kajian yang secara terstruktur mengevaluasi integrasi data antara modul Bizreq, *Bizcase*, dan Risk Management melalui rekonstruksi skema basis data, normalisasi, serta penerapan integritas relasional. Oleh sebab itu, penelitian ini difokuskan untuk merancang ulang struktur basis data modul *Bizcase* menggunakan DBSDLC guna meningkatkan konsistensi, mengurangi redudansi, dan memperkuat integrasi data pada sistem PRISMA di GMF AeroAsia.

2.2 Teori yang berkaitan

Perancangan basis data yang terstruktur dan terintegrasi memerlukan pemahaman mendalam terhadap konsep dasar data relasional. Konsep ini meliputi integritas data, mekanisme pengendalian hubungan antar entitas, teknik integrasi skema, normalisasi untuk menghilangkan anomali, serta metodologi pengembangan basis data yang sistematis. Teori-teori berikut digunakan sebagai dasar akademik bagi proses analisis, perancangan, dan implementasi basis data pada modul *Bizcase* di PRISMA.

2.2.1 *Data Integrity*

Data integrity merupakan konsep inti yang menjamin bahwa data di dalam database selalu konsisten, akurat, dan valid meskipun terjadi operasi pembaruan atau penghapusan. Menurut Yesin et al., integritas data terdiri dari beberapa komponen[23]:

1. **Entity Integrity**

   Menjamin setiap entitas memiliki identitas unik melalui *primary key* yang tidak boleh bernilai *null* maupun duplikat. Aturan ini memastikan bahwa setiap baris data dapat direferensikan dengan benar.

2. **Referential Integrity**

   Menjaga konsistensi referensi antar tabel melalui penggunaan *foreign key*. Setiap nilai FK harus merujuk pada nilai PK yang valid pada tabel induk. Pelanggaran aturan ini umumnya menyebabkan data *orphan*.

3. **Domain Integrity**

   Mengontrol nilai yang dapat dimasukkan ke dalam kolom melalui tipe data, panjang karakter, rentang nilai, dan *constraints* lainnya.

4. **User-Defined Integrity**

   Aturan tambahan yang didefinisikan berdasarkan kebutuhan khusus aplikasi, seperti kombinasi nilai tertentu yang wajib unik.

Olivier menekankan bahwa pelanggaran integritas sering menjadi akar munculnya anomali, duplikasi, dan konflik pembaruan (*update conflict*). Oleh karena itu, teori integritas data memberikan landasan bagi rekonstruksi schema *Bizcase* untuk meminimalkan inkonsistensi[24].

2.2.2 Referential Integrity

Referential integrity menjadi aspek penting dalam menjaga konsistensi relasi master–detail pada sistem PRISMA. Referential integrity mengatur bagaimana perubahan pada tabel induk berdampak pada tabel anak melalui aturan *foreign key actions*, seperti:

1. **CASCADE**: perubahan pada entitas induk diteruskan ke entitas anak.
2. **SET NULL / SET DEFAULT**: nilai pada tabel anak dirubah menjadi *null* atau nilai default.
3. **RESTRICT / NO ACTION**: perubahan tidak diperbolehkan jika masih ada referensi.

Kim memperingatkan bahwa desain relasi yang tidak tepat misalnya referensi siklikal dapat menimbulkan *deadlock* ketika FK menggunakan CASCADE[30]. Sementara He et al. menyatakan bahwa enforcement referential integrity dapat dilakukan secara deklaratif (melalui DDL) atau prosedural (melalui trigger)[31].

Bagi PRISMA, mekanisme ini penting karena modul *Bizcase* memiliki banyak tabel detail seperti bc_activity_detail, bc_cost_item, dan bc_efficiency_detail sehingga setiap perubahan harus konsisten antar-entitas.

2.2.3 Schema Integration

Integrasi skema diperlukan ketika sistem memiliki modul berbeda yang harus saling berkomunikasi. Integrasi skema melibatkan penyatuan entitas, atribut, tipe data, dan relasi dari beberapa skema menjadi struktur terpadu.

Hammad et al. menguraikan tiga pendekatan utama *schema matching[25]*:

1. **Linguistic-Based Matching**

   Cocok berdasarkan kemiripan nama atribut, misalnya *activity_name* dan *name_activity*.

2. **Constraint-Based Matching**

   Memanfaatkan metadata seperti tipe data, batasan nilai, dan definisi kunci.

3. **Instance-Based Matching**

   Berdasarkan kesamaan nilai pada data aktual.

Yousfi et al. mengusulkan *holistic schema matching* untuk kasus multi-modul, di mana lebih dari dua skema harus diintegrasikan sekaligus[26]. Teknik ini relevan untuk integrasi modul Bizreq, *Bizcase*, dan Risk Management pada PRISMA yang masing-masing memiliki struktur dan entitas berbeda.

2.2.4 Normalization Theory

Normalisasi merupakan proses mengorganisasi atribut dalam struktur relasional untuk menghilangkan redudansi dan memastikan konsistensi data. *Sug* dan *Li et al.* menjelaskan beberapa bentuk normalisasi[27], [28]:

1. **First Normal Form (1NF)**

   Atribut harus atomik dan tidak boleh mengandung nilai berulang.

2. **Second Normal Form (2NF)**

   Tidak boleh ada *partial dependency*, yaitu atribut non-kunci tidak boleh hanya bergantung pada sebagian *composite primary key*.

3. **Third Normal Form (3NF)**

   Tidak boleh ada *transitive dependency*, yaitu atribut non-kunci tidak boleh bergantung pada atribut non-kunci lainnya.

Normalisasi memastikan tidak terjadi anomali seperti:

a. *update anomaly*

b. *delete anomaly*

c. *insert anomaly*

Bagi sistem PRISMA, normalisasi diperlukan untuk menata ulang tabel bc_* agar setiap komponen *Bizcase* (cost, activity, efficiency, dll.) dapat diakses dengan konsisten tanpa redudansi.

2.2.5 Teknik *Elicitation Requirements* (M-D-I)

Dalam pengembangan sistem perangkat lunak yang kompleks, penentuan prioritas kebutuhan (*requirements prioritization*) merupakan langkah krusial untuk memastikan bahwa fitur-fitur yang dikembangkan memberikan nilai maksimal bagi pengguna dalam batasan waktu dan sumber daya yang tersedia [32], [33]. Tanpa prioritas yang jelas, proses pengembangan berisiko mengalami pembengkakan ruang lingkup dan kegagalan dalam memenuhi fungsi inti bisnis [34].

Penelitian ini mengadaptasi teknik MoSCoW, yang merupakan salah satu metode prioritas paling efektif dalam pengembangan perangkat lunak modern [29]. Untuk kebutuhan spesifik sistem PRISMA, kategori MoSCoW dipetakan menjadi model M-D-I (*Mandatory, Desirable, Inessential*) dengan definisi sebagai berikut:

1. ***Mandatory* (M) – Setara dengan *Must Have*:** Merupakan kebutuhan kritis yang wajib dipenuhi agar sistem dapat beroperasi. Jika kebutuhan ini diabaikan, sistem dianggap gagal memberikan fungsionalitas utamanya. Dalam konteks basis data, ini mencakup integritas data dasar dan alur proses utama bisnis [29], [35].
2. ***Desirable* (D) – Setara dengan *Should Have*:** Merupakan kebutuhan penting yang memiliki prioritas tinggi namun tidak bersifat kritis (*critical*). Fitur dalam kategori ini memberikan nilai tambah yang signifikan bagi

efisiensi kerja pengguna, namun sistem masih dapat berjalan tanpanya dalam jangka pendek menggunakan prosedur manual sementara [29].

3. ***Inessential* (I) – Setara dengan *Could Have / Won't Have*:** Merupakan kebutuhan tambahan yang bersifat pelengkap atau kosmetik. Kebutuhan ini memiliki prioritas terendah dan hanya akan dikerjakan apabila terdapat sisa waktu dan sumber daya setelah kebutuhan M dan D terpenuhi sepenuhnya [35].

2.2.6 Analisis Komparatif Konsep Teoritis dan Implementasi Studi Kasus

Dalam pengembangan sistem perangkat lunak yang kompleks, penentuan prioritas kebutuhan (*requirements prioritization*) merupakan langkah krusial untuk memastikan bahwa fitur-fitur yang dikembangkan memberikan nilai maksimal bagi pengguna dalam batasan waktu dan sumber daya yang tersedia [32], [33]. Tanpa prioritas yang jelas, proses pengembangan berisiko mengalami pembengkakan ruang lingkup dan kegagalan dalam memenuhi fungsi inti bisnis [34].

Untuk memastikan relevansi antara landasan teori dengan pengembangan sistem yang dilakukan, penelitian ini melakukan analisis komparatif antara definisi umum yang terdapat pada literatur dengan implementasi spesifik di PT GMF AeroAsia Tbk. Analisis ini mencakup dua komponen utama, yaitu *Project Management Information System* (PMIS) dan *Business Case*.

1. ***Project Management Information System* (PMIS)**

Secara umum, PMIS didefinisikan sebagai sistem berbasis perangkat lunak yang digunakan untuk merencanakan, mengorganisir, dan memantau jalannya proyek. Fokus utama PMIS dalam literatur seringkali menitikberatkan pada penjadwalan (*scheduling*), alokasi sumber daya, dan pelaporan status proyek untuk membantu manajer proyek dalam pengambilan keputusan manajerial [1].

Dalam lingkungan GMF, sistem PRISMA tidak hanya berfungsi sebagai alat penjadwalan, tetapi berevolusi menjadi ekosistem **integrasi data terpusat** yang menghubungkan fase perencanaan (*Bizreq*) dan eksekusi (*Project Service*). Karena GMF bergerak di industri *Maintenance, Repair, and Overhaul* (MRO) yang padat regulasi, PMIS di GMF memiliki kebutuhan spesifik pada **integritas data dan sinkronisasi**. Berbeda dengan PMIS standar yang mungkin menoleransi input manual terpisah, PRISMA menuntut mekanisme transaksi atomik di *backend* untuk mencegah perbedaan data biaya dan risiko antara dokumen perencanaan dan realisasi proyek [7].

2. ***Business Case* (Bizcase)**

Business Case secara teoritis adalah dokumen justifikasi yang digunakan untuk menilai kelayakan investasi proyek. Dokumen ini biasanya berisi analisis biaya-manfaat (*cost-benefit analysis*), risiko, dan estimasi dampak bisnis sebelum proyek disetujui [16]. Pada umumnya, *Business Case* diperlakukan sebagai dokumen statis (seperti PDF atau proposal) yang dilampirkan pada awal proyek.

Pada studi kasus ini, *Bizcase* bukan sekadar dokumen statis, melainkan ditransformasikan menjadi Modul Data Terstruktur yang dinormalisasi hingga tingkat 3NF. Modul *Bizcase Form* di GMF berfungsi sebagai *data gatekeeper* yang memecah komponen analisis (biaya, aktivitas, infrastruktur) ke dalam entitas-entitas modular (bc_activity, bc_cost, dll). Hal ini dilakukan untuk mengatasi masalah spesifik perusahaan berupa duplikasi data dan kesulitan penelusuran riwayat perubahan (*audit trail*) yang sering terjadi pada sistem pengajuan manual sebelumnya.

Untuk mempermudah pemahaman mengenai perbedaan mendasar antara konsep teoritis dan implementasi lapangan, ringkasan komparasi disajikan pada Tabel 2.2 berikut:

Tabel 2.2 Konsep Umum vs Implementasi GMF

Aspek Komparasi	Konsep Umum (Literatur)	Implementasi Studi Kasus (GMF)
Fokus Utama PMIS	Penjadwalan & Pelaporan (*Scheduling & Reporting*) [1]	Integritas Data & Sinkronisasi Backend (*Data Consistency*) [7]
Bentuk *Bizcase*	Dokumen Statis (Proposal/PDF) [16]	Modul Data Relasional (Tabel Terstruktur & API)
Penanganan Data	Input manual atau terpisah antar fase	Integrasi otomatis antara fase *Planning* & *Execution* (Atomic)
Tujuan Spesifik	Efisiensi manajemen waktu proyek	Mencegah *conflict update* dan redundansi data finansial

2.3 *Framework* DBSDLC

2.3.1 Gambaran Umum *Framework* DBSDLC

Pengembangan basis data pada sistem berskala enterprise membutuhkan pendekatan metodologis yang terstruktur agar setiap tahap dapat dikendalikan secara konsisten. Salah satu framework yang paling sesuai untuk kebutuhan tersebut adalah *Database System Development Life Cycle* (*DBSDLC*).

Setiyadi menjelaskan bahwa DBSDLC merupakan kerangka kerja sistematis yang berfokus pada siklus hidup pengembangan basis data, mencakup tahapan mulai dari perencanaan, analisis kebutuhan, desain, implementasi, hingga pemeliharaan[2]. Berbeda dari *System Development Life Cycle* (*SDLC*) yang bersifat lebih umum untuk seluruh sistem perangkat lunak, DBSDLC menitikberatkan pada aspek integritas, efisiensi struktur data, dan hubungan antar entitas.

Aminu dan Ogwueleka menambahkan bahwa DBSDLC lebih tepat digunakan untuk pengembangan sistem yang membutuhkan integrasi lintas modul dan konsistensi data yang tinggi, karena kerangka ini menekankan validasi kebutuhan data serta desain konseptual yang matang sebelum implementasi[1]. Dengan karakteristik tersebut, DBSDLC menjadi framework yang sesuai untuk penelitian ini yang berfokus pada perancangan dan

implementasi basis data modul *Bizcase* dalam sistem PRISMA di PT GMF AeroAsia Tbk.

2.3.2 Tahapan Umum dalam DBSDLC

Secara umum, DBSDLC terdiri dari beberapa tahapan yang saling terhubung. Setiyadi mendeskripsikan urutan proses DBSDLC sebagai berikut[2]:

1. ***Database Planning***

   Penentuan ruang lingkup pengembangan basis data, kebutuhan sistem, serta tujuan bisnis dari rancangan database yang akan dibangun.

2. ***System Definition***

   Identifikasi batasan sistem, modul yang terlibat, serta relasi antar komponen yang akan diintegrasikan.

3. ***Requirements Collection and Analysis***

   Pengumpulan kebutuhan data dari pengguna, proses bisnis, serta aturan integritas yang diperlukan. Tahap ini meliputi identifikasi entitas, atribut, *business rules*, dan kebutuhan integrasi lintas modul.

4. ***Database Design***

   Perancangan model konseptual, logikal, dan fisik. Pada tahap ini dilakukan normalisasi, penentuan kunci, serta perancangan struktur tabel agar bebas dari redudansi dan konsisten secara referensial.

5. ***Implementation and Loading***

   Penerapan model data ke dalam DBMS yang dipilih dalam penelitian ini PostgreSQL termasuk *schema migration*, pembuatan indeks, serta *data seeding* jika diperlukan.

6. ***Testing and Evaluation***

Pengujian terhadap struktur data, aturan integritas, serta fungsionalitas API menggunakan *Black Box Testing*, Postman, dan alat lainnya.

7. ***Operation and Maintenance***

   Pemeliharaan sistem secara berkelanjutan, termasuk penyempurnaan struktur data, optimasi kinerja, dan penyesuaian terhadap kebutuhan bisnis baru.

Damera menegaskan bahwa siklus pengembangan basis data bersifat iteratif dan harus mampu beradaptasi terhadap perubahan kebutuhan operasional, sehingga DBSDLC lebih fleksibel dibanding pendekatan linear seperti *SDLC Waterfall*[5].

2.3.3 Perbandingan DBSDLC dengan *Framework* Lain

Pemilihan DBSDLC dalam penelitian ini didasarkan pada analisis perbandingan terhadap framework lain yang umum digunakan dalam pengembangan sistem, seperti SDLC tradisional, Agile, dan *Rapid Application Development* (RAD). Aminu dan Ogwueleka menyatakan bahwa SDLC memiliki pendekatan linear yang stabil namun kurang responsif terhadap perubahan data yang kompleks[1]. Agile dan RAD lebih cepat pada sisi pengembangan antarmuka, namun kurang cocok untuk sistem dengan struktur database yang rumit karena minimnya penekanan terhadap tahap desain konseptual dan integritas data [10], [12].

Tabel 2.2 berikut merangkum perbandingan antara *framework* umum dengan DBSDLC berdasarkan literatur terkini.

Tabel 2.3 Perbandingan Framework Pengembangan Sistem

Aspek Perbandingan	SDLC (Umum)	Agile / RAD	DBSDLC (Khusus Database)
Fokus Utama	Pengembangan perangkat lunak secara menyeluruh [1]	Iterasi cepat dan kolaboratif [5], [12]	Perancangan dan pengelolaan basis data secara spesifik [2]
Pendekatan Pengembangan	Linear dan sekuensial (Waterfall) [1]	Iteratif dan adaptif [12]	Tahapan spesifik berbasis data (planning → design → implementation) [2]
Peran Pengguna	Terlibat pada awal dan akhir fase [1]	Terlibat aktif pada setiap sprint [12]	Terlibat terutama pada tahap analisis kebutuhan data [2]
Kontrol Kualitas Data	Umum dan tidak fokus pada integritas data [1]	Fokus pada fungsionalitas aplikasi, bukan struktur data [12]	Validasi integritas struktural dan referensial [23], [24]
Kelebihan	Stabil, dokumentasi lengkap, mudah diprediksi [1]	Cepat beradaptasi, fleksibel, time-to-market cepat [5], [12]	Integritas data tinggi, modular, efisien untuk desain skema [2], [23]
Keterbatasan	Kurang adaptif terhadap perubahan kebutuhan [1]	Risiko inkonsistensi struktur data [5]	Membutuhkan perencanaan data yang lebih mendalam [2]

2.3.4 Relevansi DBSDLC terhadap Penelitian

Pemilihan DBSDLC sebagai framework utama penelitian ini didasarkan pada karakteristiknya yang secara langsung mendukung kebutuhan teknis dan fungsional PRISMA. Setiyadi menyebutkan bahwa DBSDLC sangat ideal untuk sistem yang memiliki interdependensi antar modul serta tuntutan konsistensi data tingkat tinggi [2].

Dalam penelitian ini, DBSDLC diterapkan untuk:

a. melakukan analisis kebutuhan data pada modul *Bizcase*,

b. merancang model konseptual (ERD) yang terintegrasi dengan Bizreq dan Risk Management,

c. menyusun desain logikal dan fisik melalui struktur tabel PostgreSQL,

d. mengimplementasikan skema menggunakan Prisma ORM,

e. serta melakukan pengujian API untuk memastikan integritas dan konsistensi data.

Damera menegaskan bahwa integrasi DBSDLC dengan arsitektur modular seperti NestJS dan Prisma ORM dapat meningkatkan efisiensi implementasi karena tiap tahapan berjalan sistematis dan tidak saling mengganggu [5]. Secara keseluruhan, penggunaan DBSDLC diproyeksikan mampu menghasilkan basis data yang stabil, terukur, dan mudah dikembangkan untuk mendukung proses digitalisasi GMF AeroAsia melalui sistem PRISMA.

2.4 Tools/software yang digunakan

Pengembangan backend modul *Bizcase* pada sistem *Project Information System Management* (PRISMA) di PT GMF AeroAsia Tbk memerlukan perangkat lunak yang mendukung integrasi data, modularitas arsitektur, serta konsistensi basis data sesuai pendekatan *Database System Development Life Cycle* (DBSDLC). Setiap tools yang digunakan dipilih berdasarkan kebutuhan spesifik modul *Bizcase*, hasil evaluasi literatur, serta standar *enginering backend* yang diterapkan oleh tim PRISMA.

Dengan demikian, bagian ini menguraikan perangkat lunak yang digunakan beserta alasan pemilihannya dan perbandingan dengan alternatif teknologi lain.

2.4.1 PostgreSQL

PostgreSQL dipilih sebagai *Relational Database Management System* (RDBMS) utama karena mendukung *complex transactions*, *adaptive indexing*, *query optimization*, serta kemampuan menjaga konsistensi data pada sistem enterprise. Penelitian Damera menunjukkan bahwa PostgreSQL memiliki stabilitas tinggi pada lingkungan berskala besar dan optimal dalam memproses

permintaan paralel [5]. Keinsinyuran et al. menegaskan bahwa PostgreSQL efektif digunakan dalam sistem *master data management* karena memiliki kemampuan integrasi lintas modul bisnis yang kuat[3].

Dalam PRISMA, PostgreSQL menyimpan entitas seperti Bizreq, *Bizcase*, dan Risk Management, sehingga memastikan bahwa hubungan referensial antar modul tetap konsisten selama implementasi DBSDLC, khususnya pada tahap *Database Design* dan *Implementation*.

2.4.2 NestJs

NestJS merupakan framework backend berbasis Node.js yang mengusung arsitektur modular dan *dependency injection*. Framework ini dipilih karena mendukung struktur backend yang terorganisir, terukur, dan mudah dikelola pada sistem berskala enterprise. Guntakandla menunjukkan bahwa modular architecture meningkatkan skalabilitas dan resiliensi sistem dalam lingkungan backend yang kompleks[12].

Zima dan *Barszcz* menegaskan bahwa NestJS memiliki memiliki performa lebih baik dibanding framework Node.js lain seperti Express.js karena dukungan native terhadap TypeScript serta pola pengembangan yang lebih sistematis[8].

Pada PRISMA, NestJS digunakan untuk:

a. mengelola logika bisnis *Bizcase*,
b. menghubungkan backend dengan database melalui Prisma ORM,
c. mengimplementasikan API yang stabil dan konsisten.

2.4.3 Prisma ORM

Prisma ORM digunakan untuk menjembatani komunikasi antara aplikasi backend NestJS dan database PostgreSQL. Prisma mengadopsi pendekatan *type-safe query*, sehingga risiko kesalahan akses data dapat dikurangi secara signifikan.

Majerik dan Borkovcova menjelaskan bahwa ORM mampu menyederhanakan proses pengembangan backend dengan mengurangi kebutuhan penulisan SQL manual[9]. Selain itu, Riyanto dan Rochimah menemukan bahwa integrasi ORM dengan *query profiling* dan *timestamp optimization* mampu meningkatkan performa sistem hingga 30% [11].

Dalam konteks PRISMA, Prisma ORM digunakan untuk:

a. *auto migration* selama perubahan skema,
b. menjaga sinkronisasi struktur database,
c. memfasilitasi integrasi antar modul melalui relasi yang didefinisikan pada skema Prisma.

2.4.4 Postman

Postman digunakan sebagai alat utama untuk melakukan pengujian fungsional (functional testing) terhadap endpoint API modul *Bizcase*. Pengujian difokuskan pada operasi CRUD dan validasi integrasi data antar modul.

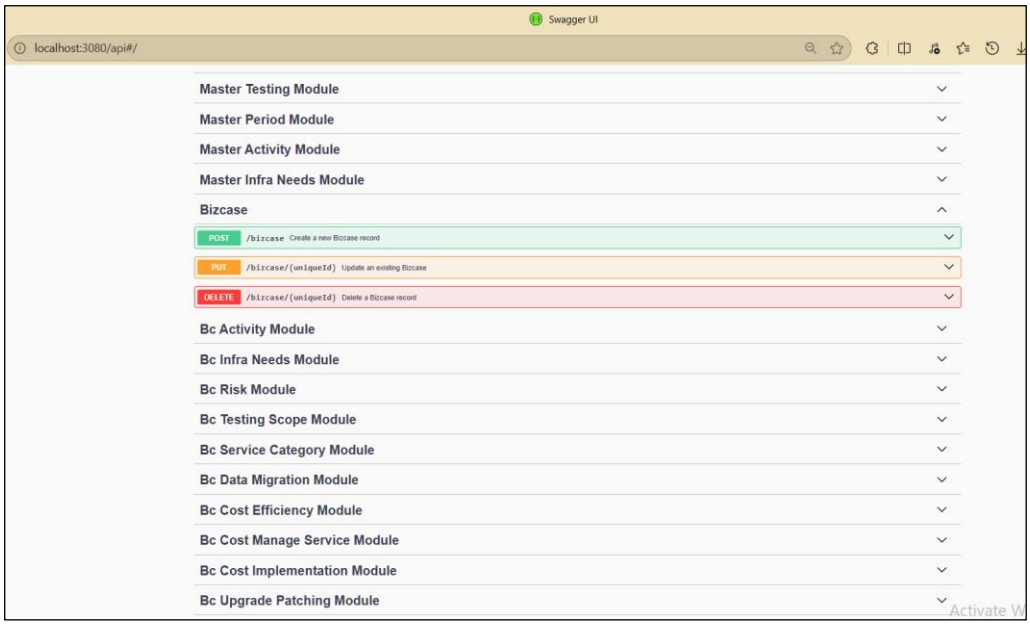
Kore et al. menyatakan bahwa Postman mendukung proses pengujian API secara manual maupun otomatis, sehingga efektif digunakan pada tahap *Testing and Evaluation* dalam DBSDLC [20], [22]. Thooriqoh et al. menambahkan bahwa fitur *automated testing* mampu mempercepat validasi endpoint hingga 80%[22]

Dengan demikian, Postman memastikan bahwa API berjalan konsisten dengan kebutuhan pengguna dan struktur data yang ditetapkan.

2.4.5 SwaggerUI

SwaggerUI merupakan *framework* berbasis antarmuka grafis yang digunakan untuk menampilkan dokumentasi dan melakukan pengujian terhadap API (*Application Programming Interface*) secara interaktif. SwaggerUI terintegrasi secara langsung dengan NestJS melalui pustaka @nestjs/swagger, yang secara otomatis menghasilkan dokumentasi API dari *decorator* dan metadata yang digunakan pada kode *backend*.

Penggunaan *SwaggerUI* mempermudah proses validasi dan pengujian endpoint karena pengembang dapat melihat struktur parameter, tipe data, serta format *response* yang dikembalikan oleh sistem tanpa perlu menggunakan aplikasi pihak ketiga. Selain itu, *SwaggerUI* juga berfungsi sebagai dokumentasi teknis API yang membantu komunikasi antar pengembang dalam proses integrasi modul.



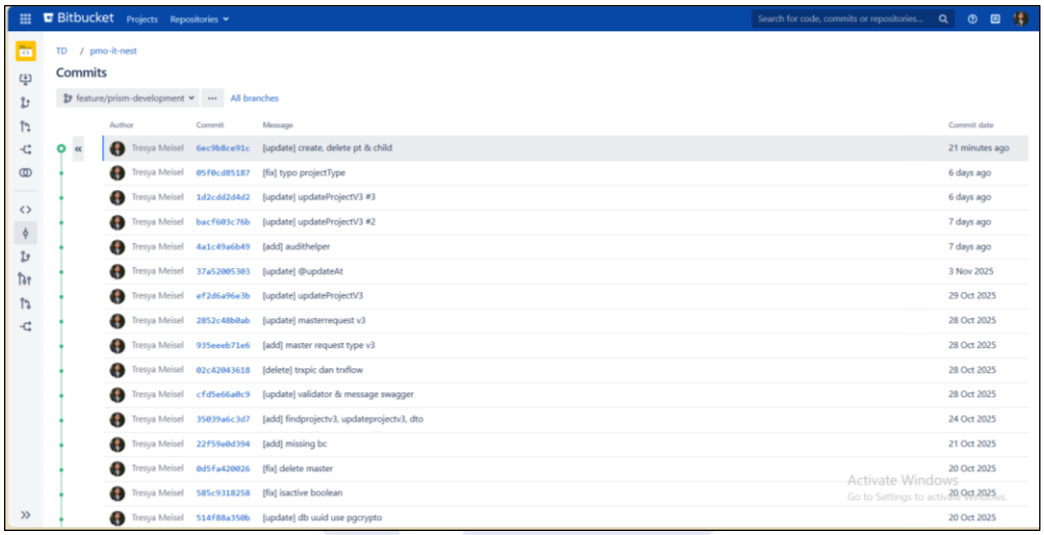
Gambar 2.1 tampilan SwaggerUI

2.4.6 Bitbucket

Bitbucket merupakan platform *version control system* berbasis *Git* yang digunakan untuk mengelola dan menyimpan kode sumber proyek secara terpusat. Dalam pengembangan sistem PRISMA, *Bitbucket* berfungsi sebagai repositori utama yang digunakan oleh tim pengembang untuk melakukan *commit*, *push*, *pull*, dan *merge branch* selama proses kolaborasi.

Penggunaan *Bitbucket* memungkinkan penerapan manajemen versi kode yang lebih terstruktur sehingga setiap perubahan dapat dilacak secara historis. Selain itu, *Bitbucket* juga mendukung integrasi dengan berbagai alat pengembangan seperti Visual Studio Code dan *pipeline* otomatis untuk *continuous integration/continuous deployment (CI/CD)* jika dibutuhkan.

Dengan adanya *Bitbucket*, proses pengembangan sistem menjadi lebih efisien karena setiap anggota tim dapat bekerja pada bagian kode yang berbeda tanpa mengganggu versi utama aplikasi. Hal ini juga membantu dalam proses dokumentasi perubahan (*changelog*) dan pengelolaan *issue tracking* secara kolaboratif. Gambar 2.2 menampilkan tampilan repositori proyek PRISMA pada *Bitbucket*.



Gambar 2.2 Hasil commit message PRISMA pada Bitbucket

2.4.7 Visual Studio Code

Visual Studio Code **(VS Code)** merupakan *Integrated Development Environment (IDE)* yang digunakan dalam proses pengembangan sistem backend. Damera menjelaskan bahwa penggunaan IDE dengan fitur *intelligent suggestion* seperti VS Code mempercepat penulisan kode dan mengurangi kesalahan *sintaks* [5]. Selain itu, integrasi terminal internal dan ekstensi Git memudahkan pengujian langsung dari lingkungan pengembangan tanpa berpindah aplikasi. Dengan demikian, VS Code mendukung kolaborasi pengembang dan menjaga konsistensi hasil implementasi dalam proyek PRISMA.

2.4.8 Draw.io

Draw.io digunakan untuk membuat diagram konseptual seperti *Entity Relationship Diagram (ERD)* dan *flowchart sistem*. Setiyadi menyatakan bahwa

visualisasi diagram mempermudah pemahaman hubungan antar entitas dan mencegah kesalahan desain basis data [2]. Dalam penelitian ini, Draw.io digunakan untuk menggambarkan struktur relasional antar entitas seperti *Bizcase*, *Bizreq*, dan *Risk Management* sebagai bagian dari tahap *Database Design* dalam DBSDLC.

2.4.9 Perbandingan Tools dan Alternatif Teknologi

Pemilihan perangkat lunak dalam pengembangan modul *Bizcase* pada PRISMA dilakukan secara terstruktur berdasarkan kebutuhan basis data, arsitektur backend, serta integrasi antar modul (Bizreq – *Bizcase* – Risk Management). Sejalan dengan rekomendasi reviewer, perbandingan tools tidak hanya mengacu pada hasil *benchmarking*, tetapi dijelaskan berdasarkan relevansi terhadap kebutuhan modul *Bizcase*, kesesuaian dengan stack PRISMA, dan kelayakan implementasi di lingkungan GMF.

Oleh karena itu, setiap kelompok tools dibandingkan dengan kandidat yang secara realistis dapat digunakan dalam konteks PRISMA, bukan seluruh tools yang tersedia secara umum.

A. Database Management System: PostgreSQL vs MySQL vs SQL Server

Pemilihan *Database Management System* (DBMS) dalam penelitian ini didasarkan pada analisis kebutuhan spesifik modul *Bizcase Form* serta standar teknologi yang diterapkan di lingkungan PT GMF AeroAsia Tbk. Berikut adalah justifikasi pemilihan PostgreSQL dibandingkan dengan alternatif lain:

1. **Dukungan Tipe Data JSONB (*Flexible Schema*):** Modul *Bizcase* memiliki karakteristik data yang dinamis, di mana atribut pada formulir pengajuan sering mengalami perubahan (*custom fields*) sesuai kebutuhan bisnis. PostgreSQL memiliki fitur unggulan tipe data **JSONB** yang memungkinkan penyimpanan data semiterstruktur dengan performa *indexing* yang tinggi. Fitur ini tidak dimiliki secara optimal oleh SQL Server atau MySQL versi

lama, sehingga PostgreSQL menjadi pilihan terbaik untuk mengakomodasi fleksibilitas formulir *Bizcase* tanpa harus terus-menerus mengubah skema tabel (*schema migration*).

2. **Integritas Transaksi dan *Transactional DDL*:** Prioritas utama sistem ini adalah integritas data. PostgreSQL mendukung *Transactional DDL* (*Data Definition Language*), yang memungkinkan perubahan struktur *database* (seperti migrasi tabel) dilakukan dalam satu transaksi yang aman (*atomic*). Jika terjadi kegagalan saat migrasi, sistem dapat melakukan *rollback* secara total. Fitur ini sangat krusial dalam lingkungan *enterprise* seperti GMF untuk mencegah kerusakan skema basis data saat pengembangan fitur baru.

3. **Kepatuhan Standar *Enterprise* dan Efisiensi Biaya:** Sebagai perusahaan MRO berskala besar, GMF menuntut perangkat lunak yang *compliant* dengan standar keamanan *enterprise* namun tetap efisien secara biaya. PostgreSQL merupakan solusi *open-source* dengan fitur setara DBMS berbayar (seperti Oracle atau SQL Server Enterprise), mencakup dukungan konkurensi tinggi (*MVCC*) dan keamanan berbasis peran (*Role-Based Access Control*). Hal ini sejalan dengan kebijakan efisiensi IT perusahaan tanpa mengorbankan performa dan keamanan data.

Tabel 2.4 berikut merangkum perbandingan teknis antara ketiga kandidat DBMS tersebut:

Tabel 2.4 P*ostgreSQL vs MySQL vs SQLServer*

Kriteria	PostgreSQL	MySQL	SQL Server
Optimasi Query	Sangat Baik	Baik	Sangat Baik
Dukungan Open Source	Ya	Ya	Tidak
Integrasi ORM	Optimal	Terbatas	Terbatas

Kriteria	PostgreSQL	MySQL	SQL Server
Skalabilitas	Tinggi	Menengah	Tinggi
Dukungan Relasi Kompleks	Sangat Baik	Menengah	Baik

Hasil perbandingan pada tabel 2.3 menunjukkan bahwa PostgreSQL menjadi pilihan paling relevan karena mendukung sistem modular, stabil, serta efisien dalam menangani beban transaksi tinggi yang diperlukan oleh GMF.

B. Framework Backend: NestJS vs Express.js vs Django

NestJS memiliki arsitektur modular dan dukungan penuh terhadap TypeScript, menjadikannya unggul dalam pengembangan sistem enterprise. Express.js lebih ringan namun kurang terstruktur, sedangkan Django unggul dalam rapid prototyping tetapi tidak kompatibel dengan ORM berbasis TypeScript.

Framework yang dibandingkan pada Tabel 2.4 dipilih karena:

a. NestJS dan Express.js adalah framework paling umum dalam ekosistem Node.js, sesuai stack PRISMA.

b. Django dipilih sebagai pembanding lintas bahasa (Python) yang sering digunakan dalam enterprise-scale backend.

Framework lain seperti Laravel (PHP), Spring Boot (Java), Ruby on Rails, dan FastAPI tidak dibandingkan karena tidak kompatibel dengan TypeScript, atau tidak digunakan oleh tim PRISMA, sehingga tidak realistis sebagai alternatif.

Tabel 2.5 NestJS vs Express.js vs Django

Kriteria	NestJS	Express.js	Django
Arsitektur Modular	Ya	Tidak	Ya
Dukungan TypeScript	Native	Parsial	Tidak Ada
Skalabilitas	Tinggi	Menengah	Menengah

C. ORM Tools: Prisma ORM vs Sequelize vs TypeORM

Majerik dan Borkovcova menunjukkan bahwa Prisma unggul dalam efisiensi *schema synchronization*, sedangkan Sequelize memerlukan konfigurasi manual [9]. Riyanto dan Rochimah menambahkan bahwa Prisma mendukung *query profiling* dan *timestamp optimization* yang mempercepat waktu eksekusi [11].

Ketiga ORM pada Tabel 2.5 dipilih karena merupakan ORM paling stabil dan paling umum digunakan pada ekosistem Node.js, serta semuanya kompatibel dengan PostgreSQL. ORM lain seperti MikroORM atau Objection.js tidak dijadikan pembanding karena dokumentasi lebih terbatas dan penggunaannya belum umum di *enterprise.*

Tabel 2.6 *Prisma ORM vs Sequelize vs TypeORM*

Kriteria	***Prisma ORM***	***Sequelize***	***TypeORM***
Type Safety	Tinggi	Rendah	Tinggi
Auto Migration	Ya	Tidak	Ya
Optimasi Query	Tinggi	Menengah	Tinggi

D. API Testing Tools: Postman vs Swagger vs Newman

Kore et al. menyatakan bahwa Postman unggul dalam kemudahan penggunaan dan visualisasi respons, sedangkan Swagger lebih difokuskan pada dokumentasi API[20]. Thooriqoh et al. menambahkan bahwa Newman lebih cocok untuk *continuous integration pipelines*, bukan untuk pengujian manual[22].

Ketiga tools pada Tabel 2.6 dipilih karena merupakan *tools testing REST API* paling relevan dan semuanya mendukung workflow

pengembangan di PRISMA. Tools lain seperti JMeter atau K6 berfokus pada *performance* testing, bukan fungsional API.

Tabel 2.7 *Postman vs Swagger vs Newman*

Kriteria	Postman	Swagger	Newman
Kemudahan Penggunaan	Tinggi	Menengah	Rendah
Automation Support	Ya	Tidak	Ya
Visualisasi Respons	Lengkap	Terbatas	Tidak Ada

Berdasarkan hasil perbandingan yang disajikan pada tabel-tabel sebelumnya, kombinasi *PostgreSQL, NestJS, Prisma ORM,* dan *Postman* merupakan konfigurasi yang paling tepat untuk mendukung pengembangan sistem backend berbasis DBSDLC di GMF AeroAsia. *Damera* menegaskan bahwa penggunaan kombinasi ini terbukti meningkatkan efisiensi pengembangan, menjaga integritas data, serta mempermudah proses integrasi antar modul [5].

Dengan demikian, seluruh perangkat lunak yang digunakan pada penelitian ini telah dipilih secara strategis berdasarkan keunggulan teknis dan kesesuaian terhadap kebutuhan sistem PRISMA, serta siap mendukung tahap implementasi yang akan dibahas pada Bab III.