PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Tugas Besar
mini Database Management System Development

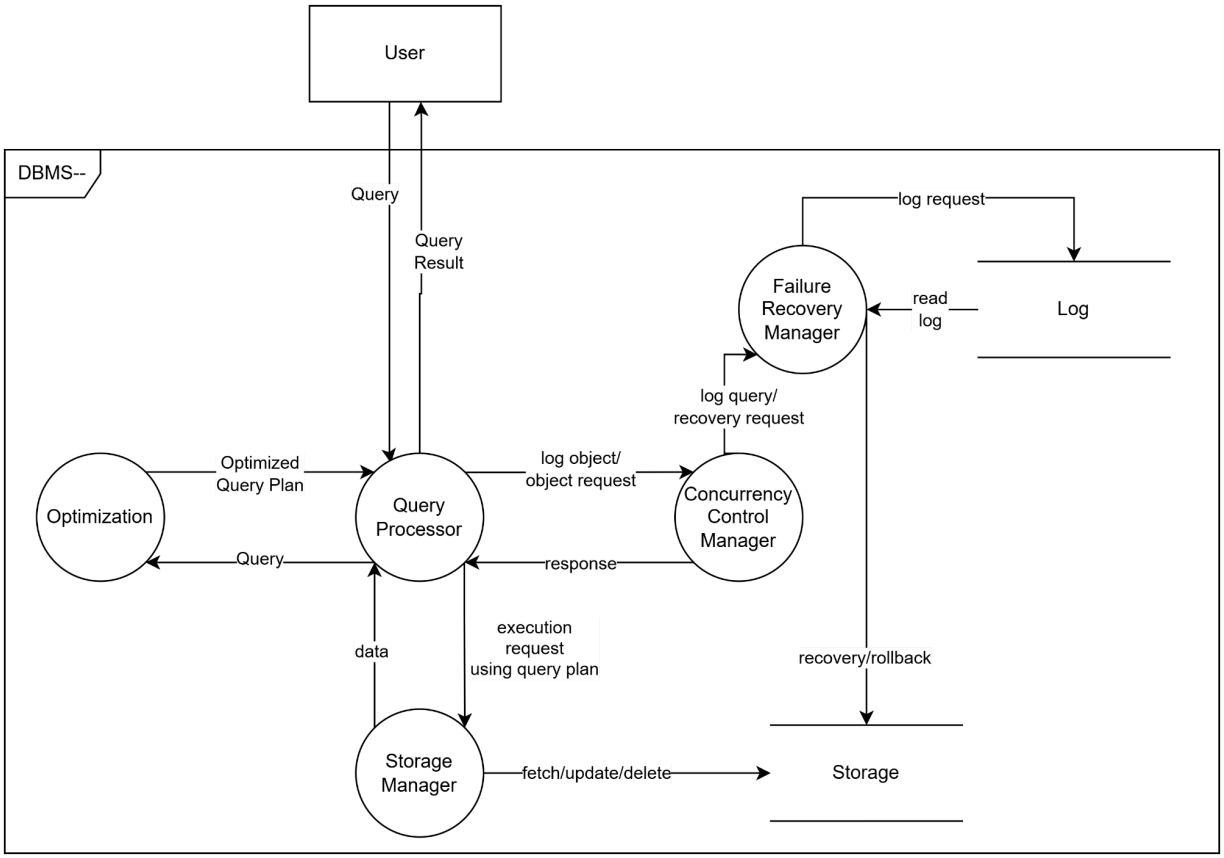
Revisi:
1. Aturan Penamaan File: 11 November 2025

IF3140 - Sistem Basis Data

2025

Part I. Gambaran Umum mDBMS

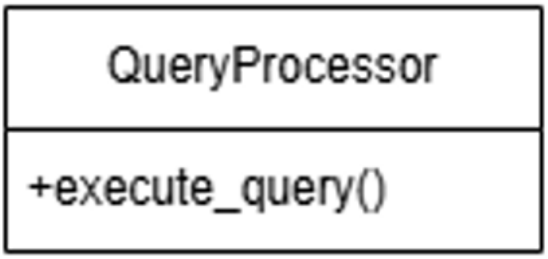
Tujuan dari proyek ini adalah untuk mengembangkan sebuah **mini sistem manajemen basis data (mDBMS)** untuk basis data relasional. Setiap **kelompok** hanya akan mengerjakan **satu (1) komponen** dari mDBMS, yang akan dijelaskan pada bagian berikutnya. Karena setiap komponen mDBMS saling bergantung satu sama lain, asisten akan menyediakan test case untuk setiap komponen serta interaksi antar-komponen guna membantu setiap kelompok memastikan bahwa implementasi mereka sudah benar. Berikut ini adalah **data flow diagram** dari mDBMS yang akan diimplementasikan dalam proyek ini.



mDBMS alan terdiri dari **5** komponen:

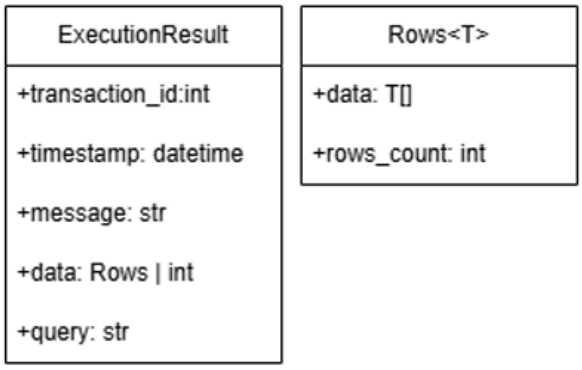
1. Query Processor

Komponen ini menangani **eksekusi dari sebuah query**. Komponen ini memiliki satu public method:



a. **execute_query(query:str)->ExecutionResult**: Method ini menerima sebuah query, yang kemudian akan dikirim ke komponen **query optimizer**. Setelah itu, method ini akan menerima query plan dari query optimizer dan mengambil baris data yang sesuai dari **storage manager**, serta melakukan manipulasi terhadap baris yang diambil, seperti metode JOIN (algoritma yang digunakan bergantung pada query plan yang diberikan). Untuk setiap transaksi, pastikan bahwa objek yang akan dibaca atau ditulis diizinkan sesuai dengan concurrency control schema. Jika diizinkan, maka eksekusi query akan dilanjutkan; jika tidak, maka transaksi tersebut harus **dibatalkan**. Sebuah query hanya dapat dieksekusi hanya jika **concurrency control** manager memberikan response; jika belum ada response, maka eksekusi akan menunggu hingga response diterima. Setiap query yang dieksekusi akan dicatat (logged) ke dalam **failure recovery** module.

Berikut adalah class interface untuk object yang digunakan dalam komponen ini:



2. Concurrency Control Manager

Komponen ini bertugas untuk mengatur scheduling query and menangani transaction concurrency dalam mDBMS. Komponen ini memastikan setiap transaksi dapat berjalan dengan benar dan konsisten saat dijalankan secara bersamaan. Komponen ini berkomunikasi dengan **Query Processor** untuk memastikan keamanan dalam melakukan operasi read dan write terhadap object. Komponen ini memiliki public method sebagai berikut:

Concurrency Control Manager
+ begin_transaction()
+ log_object()
+ validate_object()
+ end_transaction()

a. **begin_transaction() -> transaction_id: int**
   Saat perintah **'BEGIN TRANSACTION'** diberikan atau untuk setiap pernyataan yang dijalankan, berikan transaction id pada transaksi yang baru saja dimulai.
b. **log_object(object: Row, transaction_id: int)**
   Mencatat (log) sebuah objek pada transaksi tertentu. Log ini dapat digunakan untuk menerapkan lock pada suatu objek atau memberikan **timestamp** pada objek tersebut. Apa yang dicatat atau di-log tergantung pada algoritma **concurrency control** yang dipilih. Setiap kelompok wajib memilih satu algoritma concurrency control, namun **jika ingin mengimplementasikan lebih dari satu algoritma, akan mendapatkan nilai bonus.** Mekanisme concurrency control dapat berupa lock-based, timestamp-based, validation-based atau multi version-based. Pastikan

untuk mengimplementasikan perintah untuk dapat mengubah mekanisme tersebut.

c. **validate_object(object: Row, transaction_id: int, action: Action) -> Response**

   Memvalidasi apakah suatu object diizinkan melakukan aksi tertentu atau tidak. Contohnya, jika ada permintaan untuk melakukan write pada objek *x* dari transaksi 1, method ini akan mengembalikan apakah aksi penulisan tersebut **boleh dilakukan atau tidak**. Response yang diberikan berupa izin atau penolakan terhadap transaksi tersebut.

d. **end_transaction(transaction_id: int)**

   Melakukan **flush** terhadap object - object yang dimiliki oleh suatu transaksi setelah transaksi tersebut **berhasil di-commit atau dibatalkan (abort)**. Method yang diimplementasikan tergantung pada algoritma concurrency control yang dipilih. Selain itu, method ini juga akan **mengakhiri** transaksi tersebut (termination).

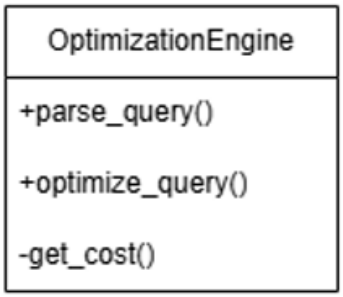
Ide untuk interface dari object yang digunakan diberikan sebagai berikut:

Action
action: Union['write', 'read']

Response
allowed: boolean
transaction_id: int

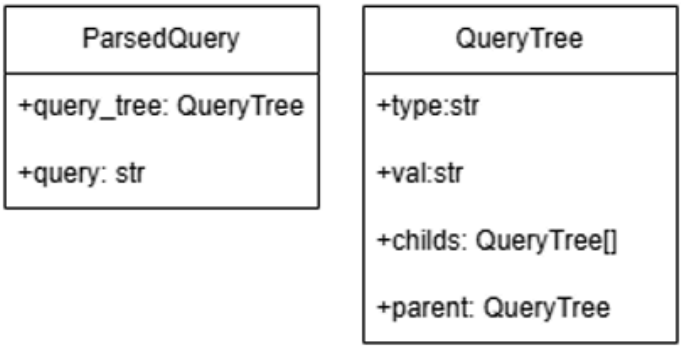
3. Query Optimizer

Komponen ini menerima sebuah **query** sebagai input, kemudian **melakukan parsing dan validasi**, serta mengembalikan query plan yang telah di optimized, dalam bentuk parsed query **object**.



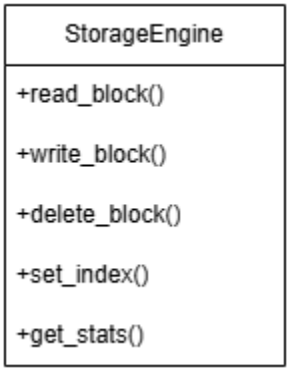
a. **parse_query(query:str)->ParsedQuery**: Method ini menerima query dalam bentuk string dan mengubahnya menjadi object yang merepresentasikan **query yang telah di-parse**. Implementasi internal dari objek *parsed query* sepenuhnya diserahkan kepada masing - masing kelompok.
b. **optimize_query(query:ParsedQuery)->ParsedQuery**: Method ini melakukan optimasi pada parsed query berdasarkan aturan optimisasi, kemudian **mengembalikan query** yang telah dipotimize. Implementasi menggunakan genetic algorithm akan mendapatkan nilai bonus.
c. **get_cost(query:ParsedQuery)->int**: Method ini menghitung biaya eksekusi dari query yang diberikan, dan adalah method pendukung untuk method optimize_query.

Berikut adalah class interface untuk object - object tersebut.



4. Storage Manager

Komponen ini bertugas untuk menangani penyimpanan, modifikasi, dan pengambilan data di harddisk. Selain itu, komponen ini juga mengelola dan menyimpan indeks, serta menyediakan informasi statistik yang digunakan untuk optimisasi query. Data disimpan di harddisk dalam bentuk **binary file(s)**. Implementasi penyimpanan data dalam satu file untuk seluruh tabel atau satu file untuk tiap tabel diserahkan kepada keputusan kelompok masing - masing. Untuk membantu implementasi, komponen ini memiliki method berikut:



a. **read_block(data_retrieval:DataRetrieval)->Rows**: Method ini menerima object data retrieval sebagai input dan mengambil data dari harddisk. Object data retrieval disperse dari query plan. Object ini berisi informasi yang membantu **storage manager** menentukan data mana yang perlu diambil dari disk. Implementasi internal diserahkan pada masing - masing kelompok. Method ini mengembalikan **data** yang berhasil diambil.
b. **write_block(data_write:DataWrite)->int**: Method ini menerima object data write sebagai input dan melakukan penambahan atau modifikasi data di harddisk. Data write object berisi informasi yang dapat membantu storage menager untuk menentukan data yang perlu dimodifikasi, data hasil modifikasi, dan data baru. Method ini mengembalikan jumlah baris affected dari operasi tersebut
c. **delete_block(data_deletion:DataDeletion)->int**: Method ini menerima object data deletion sebagai input dan melakukan penghapusan data dari

harddisk. Object *data deletion* berisi informasi yang membantu *storage manager* menentukan data mana yang harus dihapus. Method ini mengembalikan baris yang affected dari operasi tersebut.

d. **set_index(table:str,column:str,index_type:str)->None**: Method ini bertugas **membuat indeks** untuk kolom pada tabel tertentu. Method ini menerima tiga input, **nama tabel, nama kolom,** dan **tipe index**. Jenis indeks dapat berupa **B+ Tree** atau **Hash**. Untuk implementasi, cukup satu jenis indeks yang wajib dibuat, namun jika kelompok berhasil mengimplementasikan keduanya, maka akan mendapatkan nilai bonus.

e. **get_stats()->Statistic**: Method ini mengambil **data statistik** seperti jumlah tuple dalam relasi, **ukuran** dari tupel, **blocking factors**,dll. Metric ini akan digunakan query optimizer untuk menghitung query plan cost.

   n_r: number of tuples in a relation r.

   b_r: number of blocks containing tuples of r.

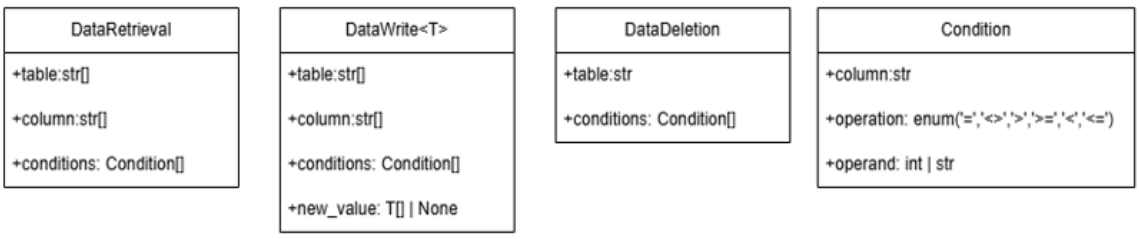
   l_r: size of tuple of r.

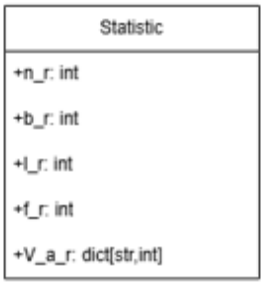
   f_r: blocking factor of r – i.e., the number of tuples of r that fit into one block.

   $V(A,r)$: number of distinct values that appear in r for attribute A; same as the size of $\Pi A(r)$.

   If tuples of r are stored together physically in a file, then: $b_r = \lceil n_r / f_r \rceil$

Berikut adalah class diagram untuk method ini. Untuk *data retrieval*, dapat menambahkan atribut tambahan untuk menentukan jenis pencarian (search type), misalnya apakah akan melakukan pencarian linear atau menggunakan indeks.



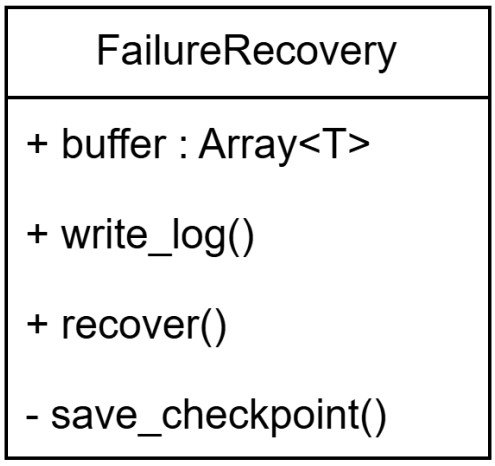


storage manager harus berkomunikasi dengan failure recovery manager untuk menangani **dirty** data yang sedang ditulis, serta untuk meminimalkan akses I/O ke harddisk.

Note: Dalam implementasi DBMS pada umumnya, proyeksi atribut dan seleksi berdasarkan kondisi biasanya dilakukan oleh query processor. Dalam hal ini, *storage manager* hanya mengembalikan blok-blok data. Namun, untuk proyek mDBMS ini, keputusan apakah tugas tersebut akan tetap dilakukan oleh storage manager (seperti pada diagram) atau dialihkan ke query processor, diserahkan kepada kelompok masing-masing. Kedua pendekatan tersebut diperbolehkan.

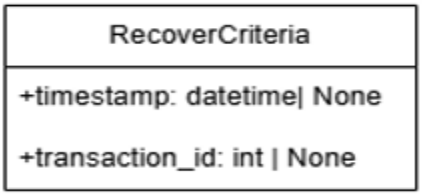
5. Failure Recovery Manager

Komponen ini bertugas untuk menangani pencatatan (logging) dan recovery terhadap eksekusi. Untuk tugas besar ini, jenis recovery yang wajib diimplementasikan adalah recovery akibat transaction aborts. Implementasi recovery akibat system failure akan mendapatkan nilai bonus.



a. **write_log(info:ExecutionResult)->None**: Method ini menerima object execution result sebagai input dan menambahkan entry baru ke dalam write-ahead log berdasarkan informasi dari object execution result tersebut.
b. **save_checkpoint()**: Method ini digunakan untuk menyimpan checkpoint ke dalam log. Dalam method ini, semua entri dalam write-ahead log sejak checkpoint terakhir akan digunakan untuk memperbarui data di physical storage, agar data tetap sinkron. Method ini dapat dipanggil setelah waktu periode tertentu, atau ketika write ahead log hampir penuh.
c. **recover(criteria:RecoverCriteria)->None**: Method ini menerima object checkpoint yang berisi RecoveryCriteria. Kriteria ini dapat berupa timestamp atau transaction ID. Proses recovery dilakukan secara backward dari entri terakhir dalam **write-ahead log**, hingga kriteria tidak lagi terpenuhi. Untuk setiap entri log, method ini akan berinteraksi dengan query processor untuk menjalankan query recovery, yang akan mengembalikan basis data ke kondisi sebelum eksekusi entry log tersebut.

Berikut adalah class diagram untuk object RecoveryCriteria:



Fungsi buffer adalah untuk menyimpan sementara data yang baru saja diakses atau diperbarui dari disk. Buffer ini digunakan untuk menjamin durability dari basis data. Buffer memiliki ukuran yang terbatas dan akan dikosongkan jika proses checkpoint dilakukan, atau buffer hampir penuh atau sudah penuh. *Storage manager* harus berkomunikasi dengan buffer ketika membaca data atau sebelum menulis data baru ke dalam disk.

Write-ahead log bertanggung jawab untuk menuliskan perubahan terlebih dahulu sebelum data benar-benar disimpan ke disk. Log ini menyimpan urutan operasi yang dilakukan pada basis data, yang kemudian dapat digunakan untuk melakukan replay operasi tersebut selama proses pemulihan.

Part II. What To Do

Kembangkan sebuah Sistem Manajemen Basis Data yang terdiri dari lima komponen, yaitu: query processor, concurrency control manager, optimization engine, storage manager, dan failure recovery. Seluruh mahasiswa **diperbolehkan menggunakan bahasa pemrograman apa pun**, namun **Python dianjurkan** karena mudah untuk dipahami, dan telah familiar bagi banyak pengguna sehingga akan mempermudah proses integrasi antar komponen.

1. Query Processor GROUP

Untuk **Query Processor**, jenis query yang harus dapat dilakukan adalah sebagai berikut:

1. SELECT

   Pengguna dapat memilih data dari satu atau lebih tabel berdasarkan atribut yang ditentukan, atau semua atribut dengan menggunakan tanda *.
2. UPDATE

   Pengguna dapat mengubah nilai record dalam sebuah tabel menggunakan perintah **UPDATE** dan **SET**. Untuk fungsionalitas ini, cukup implementasikan **satu kondisi** setelah pernyataan **WHERE**.

   Contoh: UPDATE employee SET salary = 1.05 * salary WHERE salary > 1000;
3. FROM

   Pengguna dapat memilih satu atau lebih tabel sebagai sumber data. Jika data diambil dari lebih dari satu tabel, maka hasilnya merupakan **cartesian product** dari tabel-tabel tersebut.
4. JOIN

   Pengguna dapat melakukan **penggabungan tabel** menggunakan perintah **JOIN ON** atau **NATURAL JOIN**.
5. WHERE

   Setelah perintah **WHERE**, terdapat **kondisi** yang harus dipenuhi dalam pemilihan data. Operator perbandingan yang harus didukung adalah: **=, <>, >, >=, <, <=.** Implementasi subquery akan mendapatkan nilai bonus.

6. ORDER BY

   Pengguna dapat **mengurutkan data hasil seleksi** berdasarkan satu atribut, baik secara **ascending** maupun **descending**. Atribut bertipe numerik diurutkan berdasarkan nilainya dan atribut bertipe string atau karakter khusus juga harus dapat diurutkan.

   Hint: Gunakan fungsi ord() di Python untuk membantu implementasi pengurutan ini.

7. LIMIT **(BONUS)**

   Pengguna dapat membatasi jumlah record yang ingin ditampilkan dengan perintah **LIMIT**.

8. BEGIN TRANSACTION **(BONUS)**

   Memulai sebuah transaksi.

9. COMMIT **(BONUS)**

   Melakukan commit transaksi.

10. DELETE **(BONUS)**

    Pengguna dapat menghapus record dari tabel berdasarkan **satu kondisi** saja.

    Contoh: DELETE FROM employee WHERE department = "RnD";

11. INSERT **(BONUS)**

    Pengguna dapat menambahkan record ke dalam tabel menggunakan perintah **INSERT INTO**. Hanya **satu record** yang dapat dimasukkan dalam satu query. Karena tidak ada penanganan nilai khusus seperti **NULL**, maka semua kolom harus ditentukan dalam query.

12. CREATE TABLE **(BONUS)**

    Pengguna dapat membuat tabel menggunakan perintah **CREATE TABLE**. Tipe atribut yang didukung: **integer**, **float**, **char**, dan **varchar**. Pastikan juga untuk mengimplementasikan **foreign key** dan **primary key constraint**. Jangan lupa untuk berkonsultasi dengan tim storage manager.

13. DROP TABLE **(BONUS)**

    Perintah ini digunakan untuk **menghapus (drop)** tabel. Pastikan untuk menangani **kasus khusus** seperti **cascading delete** dan **restricted delete**.

14. AS **(BONUS)**

Dengan menggunakan perintah **AS**, pengguna dapat memberikan **alias** pada tabel.

Contoh: SELECT * FROM student AS s, lecturer AS l WHERE s.lecturer_id = l.id;

2. Query Optimizer GROUP

Untuk optimisasi, seluruh aturan ekivalensi yang didukung adalah sebagai berikut:

1. **Operasi seleksi konjungtif** dapat diuraikan menjadi urutan seleksi.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Operasi seleksi bersifat komutatif.
3. **Hanya seleksi proyeksi terakhir** dalam urutan operasi proyeksi yang diperlukan; sisanya dapat dihilangkan:

$$\Pi_{L_1}(\Pi_{L_2}(\dots(\Pi_{L_n}(E))\dots)) = \Pi_{L_1}(E)$$

4. **Operasi seleksi dapat digabungkan** dengan hasil *Cartesian product* dan *theta join*:
   a. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$
   b. $\sigma_\theta(E_1 \bowtie_\theta E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
5. **Operasi theta-join (dan natural join)** bersifat komutatif:

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

6. a. Natural join bersifat asosiatif:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

   b. Theta join juga bersifat asosiatif dalam kondisi berikut:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_1 \wedge \theta_2} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_2} (E_2 \bowtie_{\theta_2} E_3)$$

   di mana θ_2 hanya melibatkan atribut dari E_2 dan E_3
7. **Operasi seleksi dapat terdistribusi** terhadap operasi theta join dalam dua kondisi berikut:
   a. Ketika semua atribut dalam kondisi seleksi θ_0 hanya melibatkan atribut dari salah satu ekspresi (E_1)

$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

b. Ketika θ_1 hanya melibatkan atribut dari E_1 dan θ_2 hanya melibatkan atribut dari θ_2

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

8. **Operasi proyeksi dapat terdistribusi** terhadap operasi theta join sebagai berikut:

   a. Jika θ hanya melibatkan atribut dari $L_1 \cup L_2$:

   $$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   b. Misalkan dilakukan join $E_1 \bowtie_\theta E_2$

   L_1dan L_2 adalah himpunan atribut dari E_1 dan E_2 respectively

   L_3 adalah atribut dari E_1 yang terlibat dalam kondisi join θ, tetapi tidak ada di $L_1 \cup L_2$ dan

   L_4 adalah atribut dari E_2 yang terlibat dalam kondisi join θ, tetapi tidak ada di $L_1 \cup L_2$

   $$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

Gunakan **metrik heuristik** untuk menentukan query plan terbaik dari beberapa query plan yang telah dibangun. Tidak seluruh plan perlu diimplementasikan

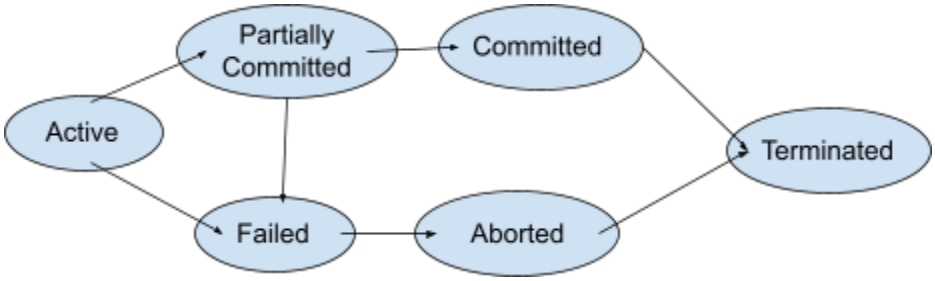
3. Concurrency Control Manager <u>GROUP</u>

Concurrency Control Manager membantu memastikan terpenuhinya properti **ACID** dalam setiap transaksi. Implementasikan sistem sehingga setiap transaksi yang dimulai dapat memenuhi properti ACID dan juga mencegah terjadinya *deadlock*.

Untuk memungkinkan dua atau lebih transaksi berjalan secara bersamaan, perlu membuat sebuah *client object* sederhana yang akan menjadi antarmuka interaksi

pengguna. Dengan demikian, bisa terdapat banyak *client*, namun untuk kelima komponen dalam mDBMS, hanya ada satu instans aktif pada satu waktu tertentu (*singleton*).

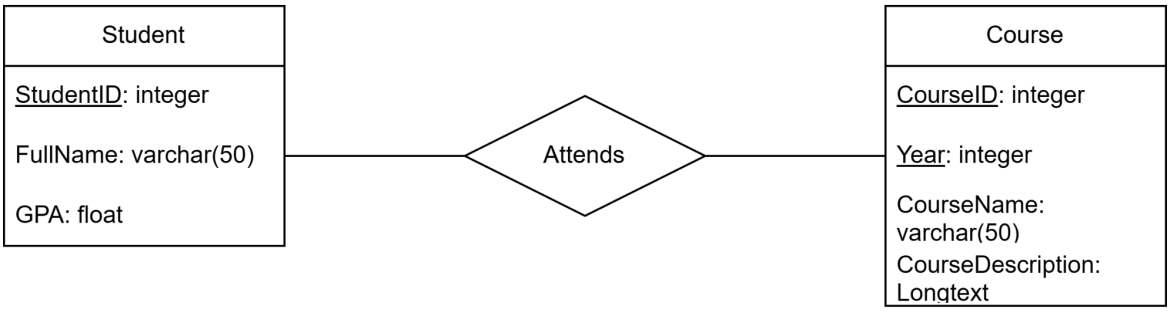
Status-status yang mungkin dimiliki oleh transaksi dapat dilihat pada bagian berikut. Pastikan *concurrency control manager* dapat mencapai setiap status tersebut beserta status lanjutannya sesuai dengan kondisi yang telah dipelajari dalam mata kuliah IF3140.



4. Storage Manager GROUP

Storage manager bertanggung jawab atas pengelolaan penyimpanan pada disk. Komponen ini memiliki akses langsung ke media penyimpanan. Komponen ini menangani proses *read* dan *write*. Jenis atribut yang didukung meliputi **integer**, **float**, **char**, dan **varchar**. Penanganan pembuatan atau penghapusan *schema* serta pembuatan atau penghapusan data merupakan bagian *bonus* yang dapat diimplementasikan bersama dengan tim *query processor*.

Kelompok ini juga harus mengimplementasikan **serializer** untuk tabel dan baris agar membantu proses alokasi blok. Basis data minimum yang harus dibuat setidaknya memiliki **tiga tabel** yang dapat di-*join* dan **minimal 50 baris** pada setiap tabel. *Serializer* harus mampu melakukan serialisasi *schema* ke dalam sebuah berkas dan memprosesnya menggunakan alokasi blok. Berikut adalah contoh **ERD** yang dapat digunakan:



5. Failure Recovery Manager GROUP

Untuk *failure recovery*, terdapat dua fitur utama yang harus dikembangkan sebagai berikut:

1. Penyimpanan log

   Semua transaksi yang telah *committed* harus disimpan dalam penyimpanan terpisah. Kelompok dapat menentukan sendiri bagaimana skema log tersebut akan dirancang.
2. Recovery

   Basis data dapat dipulihkan dengan menjalankan replay semua transaksi yang telah disimpan dalam log. Fitur ini juga dapat digunakan untuk melakukan *rollback* pada transaksi.

Untuk menguji *failure recovery* dengan beberapa transaksi, perlu membuat sebuah objek *client* sederhana yang akan digunakan oleh pengguna untuk berinteraksi. Dengan demikian, dapat terdapat banyak *client*, tetapi untuk kelima komponen *mDBMS*, hanya ada **satu instance aktif pada satu waktu** (*singleton*).

Part III. Project Mechanisms

1. Proyek ini akan dikerjakan dalam satu kesatuan yang disebut **SUPER GROUP**. Satu SUPER GROUP terdiri dari **5 GROUP**, dan setiap GROUP beranggotakan **4–5 orang**. Pembagian anggota untuk setiap GROUP telah ditentukan berdasarkan hasil performa tiap peserta pada *IF3140 Lab Sessions Results*. pembagian GROUP dapat dilihat pada spreadsheet di tab **'Group Member Lists'**.
2. Terdapat **5 komponen** yang akan dikembangkan oleh setiap SUPER GROUP, yaitu:
    a. Query Processor
    b. Query Optimizer
    c. Concurrency Control Manager
    d. Storage Manager
    e. Failure Recovery Manager
3. Setiap komponen dapat diimplementasikan menggunakan **bahasa pemrograman apa pun** dengan pendekatan **Object Oriented Programming (OOP)**, namun **Python dianjurkan** karena familiar sehingga mudah untuk mengintegrasikan antar komponen. Selain itu, **TIDAK diperbolehkan** menggunakan library Python (karena seharusnya tidak diperlukan) selain library dasar seperti datetime, ABC, UUID, re, struct, dan sejenisnya. Jika ragu, dapat menanyakan di *QnA thread* apakah library tersebut diperbolehkan atau tidak. Di luar library yang diizinkan, semua kelas, metode, fungsi, dan prosedur harus **diimplementasikan dari nol (FROM SCRATCH)**.
4. Kelompok diperbolehkan melakukan modifikasi pada *class diagram* dan metode jika diperlukan untuk membantu proses development. Beberapa diagram hanya bersifat ilustratif, sehingga mungkin perlu disesuaikan dengan kebutuhan implementasi tiap kelompok.
5. Setiap GROUP dalam satu SUPER GROUP akan mengerjakan satu komponen berbeda dari mDBMS untuk membentuk satu sistem mDBMS yang lengkap. Penentuan komponen yang akan dikerjakan oleh masing-masing GROUP harus

dilakukan melalui diskusi internal dalam SUPER GROUP. Harap isi pembagian tugas setiap SUPER GROUP **paling lambat tanggal 6 November 2025 pukul 23.59** pada spreadsheet di tab **'Group Task Assignment'**.

6. Setiap GROUP harus menyediakan *driver program* yang dapat digunakan untuk menjalankan dan melakukan pengujian terhadap komponen yang dikembangkan.
7. Anggota dalam satu GROUP **tidak diperbolehkan berkontribusi** dalam proses pengembangan komponen milik GROUP lain, bahkan dalam SUPER GROUP yang sama
8. Sistem penilaian akan terdiri dari **60%** berdasarkan komponen yang dikembangkan oleh GROUP, dan **40%** berdasarkan sistem mDBMS hasil kerja sama SUPER GROUP.
9. Semua SUPER GROUP **wajib memastikan seluruh komponen dapat berfungsi bersama**. Rancanglah sistem yang kompatibel dengan komponen lain. Karena modul *Query Processor* berfungsi menggabungkan seluruh modul lain, kelompok yang mengerjakan modul ini harus mendesain interaksi antar modul agar dapat diintegrasikan dengan baik.
10. Sangat disarankan untuk setiap GROUP memilih **satu orang penanggung jawab integrasi** untuk mengoordinasikan desain komponen dan integrasi antar GROUP dalam SUPER GROUP. Orang ini bertanggung jawab terhadap desain komponen, koordinasi lintas GROUP, serta integrasi akhir sistem. Disarankan agar penanggung jawab ini tidak terlalu banyak terlibat dalam pengembangan teknis komponen agar dapat fokus pada sistem secara keseluruhan. Namun, skema kerja ini **tidak wajib**, bebas menentukan cara kerja yang paling sesuai.
11. Deadline proyek penuh (mDBMS lengkap): **4 Desember 2025**. Namun terdapat beberapa milestone penting.
    a. **13 November 2025**, *Query Processor* harus dapat menerima query dan meneruskan query baca/tulis seperti SELECT, UPDATE, dan lainnya ke GROUP *Query Optimizer*; serta query transaksi seperti BEGIN TRANSACTION, ABORT, dan COMMIT ke *Concurrency Control Manager* dan *Failure Recovery Manager*. GROUP *Storage Manager* harus sudah

membuat basis data sederhana yang dapat diproses. Belum perlu dilakukan integrasi penuh; cukup gunakan *dummy data* untuk memastikan komponen dapat menerima data dengan benar.

b. **20 November 2025**, Semua GROUP harus telah menyelesaikan minimal **50%** dari pekerjaannya. Coba untuk mulai mengintegrasikan *Query Processor*, *Storage Manager*, dan *Query Optimizer*. Mulai buat *client* sederhana (CLI, tidak perlu GUI).
c. **27 Novermber 2025**, Mulai integrasikan *Concurrency Control Manager* dan *Failure Recovery Manager* dengan sistem mDBMS lainnya. Melanjutkan pengujian bug dan test case.
d. **4 December 2025**, Semua komponen sudah terintegrasi menjadi satu sistem mDBMS yang berfungsi penuh. *Client* sederhana sudah dapat digunakan.

12. Pada setiap *milestone*, **setiap GROUP** wajib mengumpulkan laporan dalam bentuk *milestone report* dan GitHub Release, dan progress laporan SUPER GROUP. Laporan tidak perlu panjang; yang penting singkat, padat, dan jelas.
    a. Milestone 1: 13 November 2025 **23.59,**
    b. Milestone 2: 20 November 2025 **23.59,**
    c. Milestone 3: 27 November 2025 **23.59**

    Kumpulkan laporan *milestone* melalui form berikut.

    Penamaan laporan milestone adalah sebagai berikut:

    - MilestoneX_<Nama Super Group>_<Nama Group> untuk laporan group
    - MilestoneX_<Nama Super Group> untuk laporan super group

    Dengan X adalah nomor milestone.
13. Setiap SUPER GROUP **wajib membuat satu laporan akhir (FINAL REPORT).** Berbeda dengan laporan *milestone*, laporan akhir harus ditulis sedetail mungkin, karena penilaian arsitektur sistem dan implementasi komponen akan didasarkan pada laporan ini. Pada batas waktu akhir, setiap SUPER GROUP **harus mengumpulkan laporan akhir** dan **GitHub Release**. Seluruh *deliverables* harus dikumpulkan **paling lambat 4 Desember 2025 pukul 23.59**. Kumpulkan laporan

akhir dan tautan GitHub repository melalui form berikut. Penamaan laporan akhir adalah FinalReport_<Nama Super Group>

14. Untuk *Git Releases*, untuk setiap *repository* (biasanya satu per komponen), buat *release* dengan format **v<x>.<y>**, di mana **x** menunjukkan *checkpoint* dan **y** menunjukkan versi *release* yang dimulai dari 1. Jika hanya memiliki satu *repository*, cukup buat satu *release* saja. Harap cantumkan **tautan *release*** tersebut pada **lampiran laporan**. Untuk **laporan akhir**, sertakan **seluruh repository** yang terlibat dalam *tugas besar* ini.
15. Pengumpulan terlambat untuk setiap *milestone* akan dikenakan **penalti pada kelompok (GROUP) yang bersangkutan** (bukan *SUPER GROUP*). Pengumpulan terlambat untuk **laporan akhir** akan dikenakan **penalti pada SUPER GROUP yang bersangkutan**.
16. Contoh project structure

```
|- Query Processor
        |- classes.py
        |- UnitTest.py
|- Query Optimizer
        |- classes.py
        |- UnitTest.py
|- Concurrency Control Manager
        |- classes.py
        |- UnitTest.py
|- Storage Manager
        |- classes.py
        |- UnitTest.py
|- Failure Recovery
        |- classes.py
        |- UnitTest.py
main.py
data.dat
```