Analisis Soal UTS PDL (IF4040)

Cara membaca ER, tabel konversi ke lima model, tata cara menjawab,
jawaban UTS sample, studi kasus prediksi, dan penerapan ke tugas

Dokumen ini mendampingi catatan *UTS PDL* dan soal *UTS Sample*. Tujuannya satu: **kamu bisa menjawab soal bergaya UTS sample untuk kasus apa pun**.

Susunan isi.

1. Analisis UTS sample: bentuk soal, pola, dan peta CPMK.
2. Cara membaca diagram ER dan **tabel konversi** semua konstruksi ER ke lima model (relasional, dokumen, column-family, graf, XML).
3. Tata cara menjawab tiap jenis soal.
4. Jawaban lengkap UTS sample.
5. Satu studi kasus prediksi lengkap dengan jawaban.
6. Penerapan ke studi kasus tugas: Nimonspedia (Project-1) dan TransitTap (Project-2).
7. Ringkasan inti.

Cakupan. Materi minggu 01–07 (ER dan relasional, pengantar NoSQL, key-value, dokumen, column-family, graf, XML), ditambah materi *Choosing a Database* sebagai bahan alasan memilih model. Spasial dan temporal tidak termasuk. Penjelasan yang tidak ada di catatan kuliah diberi tanda *(tambahan)*.

Konvensi skema relasional. `Tabel(pk, atribut, FK -> Tabel)`: atribut bergaris bawah adalah primary key, sama seperti notasi di slide.

Diagram ER digambar dengan notasi kuliah (varian Chen ala Silberschatz: atribut ditulis di dalam kotak entity).

Daftar Isi

1 Analisis UTS Sample **3**

1.1 Bentuk Soal . . . 3

1.2 Pola yang Terlihat . . . 3

1.3 Prediksi Bentuk UTS . . . 3

2 Membaca Diagram ER dan Tabel Konversi **4**

2.1 Cara Membaca Notasi ER (Notasi Kuliah) . . . 4

2.2 Tabel Konversi: Konstruksi ER ke Lima Model . . . 6

3 Tata Cara Menjawab Tiap Jenis Soal **8**

3.1 Sebelum Mulai (Untuk Semua Soal) . . . 8

3.2 Soal Model Relasional . . . 8

3.3 Soal Desain Dokumen . . . 8

3.4 Soal Desain Column-Family (BigTable-like) . . . 9
3.5 Soal Desain Graf . . . 10
3.6 Soal Use Case Key-Value . . . 10
3.7 Soal XML (Skema/DTD dan Query) . . . 11
3.8 Soal "Pilih Model Mana" (Choosing a Database) . . . 11

4 Jawaban Lengkap UTS Sample **11**
4.1 Analisis Kasus (Langkah 3.1) . . . 11
4.2 Soal 1: Model Relasional . . . 13
4.3 Soal 2a: Desain Dokumen . . . 14
4.4 Soal 2b: Desain Column-Family (BigTable-like) . . . 16
4.5 Soal 2c: Desain Graf . . . 17
4.6 Soal 3: Use Case Key-Value . . . 18
4.7 Soal 4a: XML Schema atau DTD untuk Hasil Q3 . . . 19
4.8 Soal 4b: XQuery . . . 20

5 Studi Kasus Prediksi: KursusKita **21**
5.1 Soal . . . 21
5.2 Analisis Kasus . . . 23
5.3 Jawaban Soal 1: Model Relasional . . . 23
5.4 Jawaban Soal 2a: Desain Dokumen . . . 24
5.5 Jawaban Soal 2b: Desain Column-Family . . . 26
5.6 Jawaban Soal 2c: Desain Graf . . . 26
5.7 Jawaban Soal 3: Use Case Key-Value . . . 28
5.8 Jawaban Soal 4: DTD, XQuery, XPath . . . 28
5.9 Jawaban Soal 5: Model Paling Cocok per Query . . . 29

6 Penerapan ke Studi Kasus Tugas **29**
6.1 Nimonspedia (Project-0 dan Project-1) . . . 29
6.2 TransitTap (Project-2, ClickHouse columnar) . . . 30

7 Ringkasan Inti **31**

1 Analisis UTS Sample

1.1 Bentuk Soal

Satu studi kasus dipakai untuk **semua** soal: aplikasi media sosial, berisi diagram ER, informasi tambahan (aturan data), dan **tiga query yang sering terjadi**.

Soal	Yang diminta	CPMK
1	Model relasional murni + *reasoning*; boleh dioptimasi asal dengan asumsi	3
2a	Desain logis dokumen + penjelasan embedding/referencing	2, 3
2b	Desain logis column-family (BigTable-like, kecuali dijelaskan lain)	2, 3
2c	Desain logis graf	2, 3
3	Satu use case key-value di kasus itu	1
4a	XML Schema **atau** DTD untuk hasil query 3	2, 3
4b	XQuery berdasarkan desain 4a	2, 3

CPMK: (1) menjelaskan kapan basis data non-relasional/semi-terstruktur lebih disukai; (2) mendeskripsikan karakteristik dasar model non-relasional/semi-terstruktur; (3) mengeksplorasi implementasinya.

1.2 Pola yang Terlihat

1. **Semua soal adalah soal desain.** Tidak ada soal hafalan teori (CAP, BASE, sejarah SQL), tidak ada query SQL, dan tidak ada sintaks MongoDB/Cypher yang diwajibkan.
2. **Desain digerakkan oleh query.** Tiga "query yang sering terjadi" adalah petunjuk tersembunyi yang menentukan:
   - apa yang di-*embed* di dokumen,
   - kolom apa yang dikelompokkan dalam satu column family,
   - bentuk XML (soal 4 memakai hasil query 3 secara langsung),
   - kapan model relasional layak dioptimasi.
3. **Yang dinilai adalah *reasoning* dan asumsi.** Setiap soal meminta penjelasan. Desain yang benar tanpa alasan kemungkinan kehilangan nilai.
4. **Diagram ER sengaja memuat konstruksi yang menguji aturan pemetaan:** ISA disjoint yang tidak lengkap ("dll."), atribut multivalued yang boleh kosong, atribut komposit, relasi rekursif beratribut, relasi rekursif dengan panah, partisipasi total, dan many-to-many.
5. **Soal saling menyambung.** Soal 4 memakai query 3, dan soal 4b memfilter berdasarkan **email** pengguna. Jadi desain XML di 4a **harus memuat email** walaupun query 3 tidak menyebutnya. *Tips: baca semua sub-soal dulu sebelum mendesain.*

1.3 Prediksi Bentuk UTS

Kemungkinan besar formatnya sama:

- Satu kasus baru dengan diagram ER, aturan tambahan, dan beberapa query sering.
- Soal yang meminta model relasional, desain dokumen, column-family, dan graf, satu use case key-value, serta XML (skema atau DTD) beserta query-nya.

Yang berubah adalah **domain** dan **konstruksi ER**-nya. Konstruksi yang belum muncul di sample dan wajar muncul: **weak entity, relasi ternary, ISA overlapping, ISA total, relasi 1:1, atribut turunan (derived), dan aggregation**. Semuanya ada di materi minggu 01. Karena itu bagian 2 membahas **semua** konstruksi ER, bukan hanya yang ada di sample.

Variasi lain yang mungkin, dan sudah disiapkan jawabannya:

- XML dengan **DTD** (sample dijawab dengan XML Schema dan DTD).
- Soal **XPath**.
- Pendekatan **columnar** untuk soal 2b.
- Soal "pilih model mana untuk kebutuhan ini" (materi Choosing a Database).

2 Membaca Diagram ER dan Tabel Konversi

2.1 Cara Membaca Notasi ER (Notasi Kuliah)

Simbol	Arti	Contoh
Rectangle	Entity set	User
Double rectangle	**Weak entity set**	section
Atribut digarisbawahi	Primary key	email
Garis bawah **putus-putus**	**Discriminator** weak entity	sec_id
Diamond	Relationship set	post
Double diamond	**Identifying relationship** (penghubung weak entity)	sec_course
Atribut menjorok di bawah atribut lain	**Composite attribute**	ts_posting: date, time
`{ atribut }`	**Multivalued attribute**	{ hobbies }
`atribut()`	**Derived attribute**	age()
Garis berpanah (→) ke entity	Sisi "**one**" (paling banyak satu)	post → User
Garis tanpa panah	Sisi "**many**"	
Garis ganda antara entity dan relationship	**Total participation** (setiap entity wajib ikut)	Posting = post
Label pada garis (followee/follower)	**Role** pada relasi rekursif	follow
Kotak kecil tersambung ke diamond	**Atribut relationship**	ts_follow
Segitiga **ISA**	Specialization/generalization	Media ISA Image
Tulisan **disjoint** di segitiga	Satu entity hanya di satu subclass	
Tanpa tulisan disjoint	**Overlapping** (boleh di beberapa subclass)	
Garis ganda dari superclass ke segitiga	**Total** (wajib masuk salah satu subclass)	
Kotak besar mengelilingi relationship	**Aggregation** (relationship diperlakukan sebagai entity)	Kotak mengelilingi proj_guide; eval_for menghubungkan kotak itu ke evaluation

Membaca kardinalitas dari panah. Pada relasi biner, panah di sisi entity E berarti "setiap entity di sisi lain berhubungan dengan **paling banyak satu** E". Contoh sample: panah dari `post` ke User berarti setiap Posting dimiliki paling banyak satu User. Ditambah garis ganda di sisi Posting (total), setiap Posting dimiliki **tepat satu** User. Itu relasi many-to-one dari Posting ke User.

Relasi ternary: paling banyak **satu panah** (materi). Panah ke E berarti setiap **kombinasi** entity lain berhubungan dengan paling banyak satu E, sehingga primary key relasinya adalah gabungan key entity **tanpa** E (aturan PK ini *(tambahan)* dari buku Silberschatz; catatan kuliah hanya memberi contoh proj_guide).

2.2 Tabel Konversi: Konstruksi ER ke Lima Model

Ini alat utama untuk soal variasi apa pun. Cari konstruksinya di kolom kiri, lalu baca cara memetakannya ke model yang ditanya.

Konstruksi ER	Relasional	Dokumen	Column-family (BigTable-like)	Graf	XML
Strong entity	Satu tabel, PK = key entity	Satu collection, `_id` = key	Row key = key entity bila query mencari entity lewat key-nya; tabel tetap dibuat per kebutuhan baca (3.4)	Vertex berlabel nama entity, properti = atribut	Satu elemen; key sebagai **atribut** elemen
Composite attribute	Diratakan: satu kolom per komponen (`ts_date`, `ts_time`)	Sub-dokumen `{date, time}`	Kolom per komponen dalam satu CF	Properti per komponen (atau satu properti gabungan)	Elemen bersarang `<ts><date/><time/></ts>`
Multivalued attribute	Tabel terpisah `E_M(PK E, nilai)`, PK gabungan	Array `[ ]` di dokumen	**Valueless columns**: nilai jadi nama kolom	Properti list, atau vertex sendiri bila perlu dicari/dibandingkan	Elemen berulang (`*` di DTD, `maxOccurs="unbounded"`)
Derived attribute	**Tidak disimpan**; dihitung lewat function/view	Dihitung aplikasi, atau disimpan bila sering dibaca (denormalisasi)	Bisa disimpan sebagai nilai turunan yang dihitung saat tulis ("derived values" adalah bagian desain CF)	Dihitung saat query	Dihitung saat membuat XML
Atribut opsional	Kolom boleh NULL	Field boleh tidak ada (schemaless)	Kolom boleh tidak ada di row (sparse)	Properti boleh tidak ada	`minOccurs="0"` / `?` di DTD / `#IMPLIED`
1:N (many-to-one)	FK di sisi "many" (NOT NULL bila total)	Embed array di sisi "one" bila anak kecil dan selalu dibaca bersama; selain itu reference di sisi "many"	Denormalisasi: anak sebagai kolom di row induk, atau FK sebagai kolom di row anak	Satu edge berarah	Anak bersarang di induk, atau IDREF
1:1	FK di sisi **total participation** (hindari NULL)	**Embed** (ideal untuk 1:1)	Kolom dalam row yang sama	Satu edge (atau gabung jadi satu vertex)	Elemen bersarang
M:N	Tabel relasi: PK = gabungan PK kedua entity; atribut relasi ikut sebagai kolom biasa	**Reference**: array ID di salah satu/kedua sisi, atau collection relasi sendiri bila punya atribut/besar	**Denormalize instead of join**: kolom bernama ID entity lain di row (valueless column atau ID → info), dua arah bila perlu (`Cust_by_Prod`/`Prod_by_Cust`)	**Edge** (kekuatan utama graf); atribut relasi = properti edge	IDREFS, atau elemen relasi berisi dua referensi
Atribut relationship	Kolom di tabel relasi (atau di sisi "many" bila 1:N)	Field di sub-dokumen/dokumen relasi	Nilai kolom (nama kolom = ID entity lain, value = atribut)	**Properti edge**	Subelemen/atribut dari elemen relasi
Rekursif (role)	Tabel relasi dengan dua FK ke tabel yang sama (nama sesuai role); bila 1:N cukup FK ke tabel sendiri	Array ID (children) dan/atau ID parent; untuk hierarki boleh **array ancestors**	Kolom berisi ID row lain di tabel yang sama	Edge dari vertex ke vertex label yang sama; arah sesuai role	IDREF ke elemen sejenis

Konstruksi ER	Relasional	Dokumen	Column-family (BigTable-like)	Graf	XML
Weak entity	Tabel dengan PK = PK identifying entity + discriminator; FK ke identifying	**Embed** di dokumen identifying entity (eksistensinya bergantung)	Kolom di row identifying entity (mis. `m01_title`) atau row key gabungan `induk#discriminator`	Vertex sendiri + edge identifying (atau properti bila kecil)	Elemen bersarang di dalam elemen induk
Ternary (n-ary)	Tabel relasi berisi ketiga FK; PK = gabungan, **tanpa** key entity yang ditunjuk panah (aturan PK: *(tambahan)*)	Collection relasi sendiri berisi tiga referensi	Row key gabungan dua key; kolom berisi key ketiga	Edge graf hanya biner, jadi **reifikasi**: vertex perantara (entity artifisial) dengan tiga edge	Elemen relasi berisi tiga referensi
ISA disjoint	(a) Tabel superclass + tabel per subclass (PK = PK superclass, FK) + kolom tipe (cara di materi); (b) optimasi satu tabel dengan kolom NULL *(tambahan)*; (c) tabel per subclass saja, hanya bila **total** *(tambahan)*	Satu collection **polymorphic** (field tipe + field khusus), cocok bila subclass sering dibaca bersama	Satu tabel; atribut subclass sebagai kolom dinamis di CF khusus (sparse)	Satu label umum + label/properti tipe (atau label per subclass)	`xs:choice` / `(A|B|C)` di DTD, atau elemen dengan atribut tipe
ISA overlapping	Opsi (a) tabel per subclass; entity boleh punya baris di beberapa tabel, keanggotaan dilihat dari ada/tidaknya baris. Satu kolom tipe tunggal **tidak cukup** (flag per subclass opsional)	Sub-dokumen per peran + array `roles`	Satu CF per subclass di row yang sama	Vertex boleh punya **beberapa label** sekaligus *(tambahan)*	Elemen peran opsional (`?`) yang boleh muncul bersamaan
ISA total vs partial	Total tidak bisa ditegakkan FK biasa (perlu trigger/aplikasi); partial memungkinkan baris superclass tanpa subclass	Ditegakkan kode aplikasi (schemaless)	Ditegakkan aplikasi	Ditegakkan aplikasi	Skema: subelemen wajib vs opsional
Aggregation	Tabel berisi PK relationship teragregasi + PK entity terkait + atribut	Embed di dokumen relationship-nya (jika relationship-nya sudah jadi dokumen)	Kolom di row relationship teragregasi	Edge tidak bisa ke edge, jadi relationship teragregasi dijadikan **vertex**	Elemen bersarang di elemen relationship
Total participation	NOT NULL pada FK (sisi many), atau trigger untuk M:N	Ditegakkan aplikasi	Ditegakkan aplikasi	Ditegakkan aplikasi	Elemen wajib (`minOccurs="1"`, tanpa `?`)

Ingat tiga perbedaan sifat model:

- **Relasional:** desain mengikuti **struktur** dan meminimalkan redundansi; join dipakai bebas; ACID dan constraint ditegakkan database.
- **NoSQL (dokumen, column-family, key-value):** desain mengikuti **query**; redundansi (denormalisasi) disengaja demi performa; integritas dijaga **aplikasi**.
- **Graf:** desain mengikuti **relasi**; relasi adalah data utama dan ditelusuri tanpa join.

3 Tata Cara Menjawab Tiap Jenis Soal

3.1 Sebelum Mulai (Untuk Semua Soal)

1. **Baca semua sub-soal dulu.** Catat atribut yang dibutuhkan soal belakang (seperti email untuk XQuery di sample).
2. **Tandai konstruksi ER** dengan tabel di 2.1: panah, garis ganda, ISA (disjoint/overlapping, total/partial), weak, multivalued, komposit, derived, rekursif, ternary, aggregation.
3. **Tulis ulang query sering** dalam bentuk "entity apa + atribut apa + filter apa + diurutkan apa". Ini bahan semua desain NoSQL.
4. **Tulis asumsi** untuk apa pun yang tidak disebut kasus (ukuran data, frekuensi tulis, apakah data sering berubah). Soal secara eksplisit meminta asumsi.

3.2 Soal Model Relasional

1. Petakan **setiap** konstruksi dengan kolom "Relasional" di tabel 2.2, berurutan: strong entity → weak entity → atribut (komposit, multivalued, derived) → relasi 1:N/1:1 → M:N → rekursif → ternary → ISA → aggregation.
2. Tentukan **PK** tiap tabel: entity memakai key-nya; relasi memakai aturan kardinalitas (M:N = gabungan; 1:N = PK sisi many; 1:1 = salah satu).
3. Tulis **FK** dan **constraint**: NOT NULL untuk atribut wajib dan FK partisipasi total; CHECK untuk domain terbatas (mis. `posting_type IN ('normal','reply')`); UNIQUE untuk candidate key lain.
4. **Optimasi (opsional, wajib dengan alasan):** periksa query sering. Jika satu query butuh banyak join atau perhitungan berat, boleh menggabungkan tabel ISA, menyimpan atribut turunan, menduplikasi atribut (denormalisasi), atau memecah tabel secara vertikal (vertical partitioning). Selalu sebutkan **trade-off**: redundansi, risiko anomali update, atau NULL.
5. **Reasoning yang ditulis:** sebutkan aturan pemetaan yang dipakai untuk konstruksi yang tidak trivial (ISA, multivalued, rekursif, ternary) dan alasan setiap optimasi.

3.3 Soal Desain Dokumen

Fase desain dari materi: konseptual (ER + **query pattern**) → logis (tipe dokumen dan relasinya, mis. Document Interaction Diagram dan JSON Schema) → fisik (index, sharding). Soal UTS meminta level **logis**.

1. **Tentukan dokumen utama** dari query: entity yang paling sering diambil sebagai "satu kesatuan" menjadi collection.
2. **Untuk tiap relasi, putuskan embed atau reference** dengan empat faktor dari materi:

Faktor	Embed bila…	Reference bila…
Pola akses	Sering diakses bersama induknya	Diakses mandiri
Ukuran dan pertumbuhan	Kecil atau tetap	Besar atau terus bertambah
Frekuensi update	Perlu update atomik bersama induk	Sering diupdate sendiri
Kompleksitas	Hierarki sederhana	Kompleks atau berevolusi

Aturan cepat: **1:1 → embed**; **1:N dengan anak kecil → embed array**; **M:N atau anak besar → reference** (array ID).

3. **Pertimbangkan jalan tengah:** reference ditambah **salinan sebagian atribut** (snapshot) yang dibutuhkan query, mis. `store: {_id, name}` di Nimonspedia. Pilih atribut yang jarang berubah agar salinan tidak cepat basi.
4. **ISA:** satu collection polymorphic bila subclass sering dibaca bersama; pisah collection bila jarang bersama.
5. **Hierarki/rekursif:** parent reference, children references, atau array ancestors.
6. **Waspadai** dokumen yang tumbuh tanpa batas (follower jutaan, event log). Pecah, atau pakai pola "satu dokumen per periode" seperti contoh truk di materi.
7. **Tulis:** contoh struktur JSON tiap collection, keterangan embed/reference beserta alasannya, dan index utama.

3.4 Soal Desain Column-Family (BigTable-like)

Desain dimulai dari **query**: entity, atribut, kriteria query, dan nilai turunan.

1. **Satu tabel per kebutuhan baca utama** (bukan per entity). Tabel analog dengan tabel relasional, tetapi data terurut menurut row key dan kolomnya dinamis.
2. **Pilih row key** dari kriteria pencarian query. Karena row tersimpan **terurut** (HBase: lexicographic), row key gabungan seperti `user#waktu` memungkinkan **range scan** (mis. riwayat dalam rentang waktu). Hindari **hotspotting**: jangan memakai nilai sekuensial murni (waktu, auto-increment) sebagai awal row key. Materi menyebut dua cara: **hashing** nilai sekuensial atau **prefix string acak**. Memakai ID entity (mis. ID user) sebagai awal row key juga menyebarkan tulisan *(tambahan).*
3. **Kelompokkan kolom yang dibaca bersama dalam satu column family.** Pisahkan yang jarang dibaca ke CF lain.
4. **Terapkan 7 guideline:** (1) denormalize instead of join; (2) valueless columns (ID sebagai nama kolom); (3) pakai nama kolom **dan** nilainya untuk menyimpan data (mis. nama kolom = ID follower, value = nama follower); (4) satu entity dalam satu row (tulis atomik); (5) hindari hotspotting; (6) simpan versi secukupnya; (7) hindari struktur kompleks di value.
5. **Index:** row key otomatis jadi primary index. Pencarian lewat atribut lain memakai secondary index terkelola (hindari untuk nilai distinct sedikit, terlalu unik, atau sparse) atau **tabel index buatan sendiri** (seperti `Cust_by_Prod`).
6. **Tulis:** tiap tabel dengan row key, nama CF, kolom (tandai kolom dinamis/valueless), contoh satu row, dan alasan.

Format penulisan yang disarankan:

```
Tabel User
  Row key     : email
  CF profile  : phone_number, name, gender, date_of_birth
  CF follower : <email_follower> = <nama_follower>   (kolom dinamis)
```

Bila ingin memakai pendekatan columnar (seperti ClickHouse di Project-2)

Soal mengizinkan pendekatan lain asal dijelaskan. Columnar murni menyimpan tiap **kolom** tabel secara terpisah (bukan row berisi column family). Tabel biasanya dibuat **lebar dan terdenormalisasi**, urutan fisik ditentukan `ORDER BY`, dan cocok untuk **analitik/agregasi** atas banyak baris, kurang cocok untuk lookup atau update satu baris. Untuk soal bergaya sample (lookup profil, posting tertentu), BigTable-like lebih tepat. Pakai columnar hanya bila query-nya analitik.

3.5 Soal Desain Graf

Empat langkah dari materi:

1. **Identifikasi query.**
2. **Entitas menjadi vertex**, dengan label dan properti.
3. **Relasi menjadi edge**: tentukan label, **arah** (directed untuk relasi tidak simetris seperti follow; undirected untuk yang simetris), dan properti (atribut relasi).
4. **Petakan query domain ke query graf**, mis. "siapa follower X" menjadi "vertex di ujung edge FOLLOWS yang masuk ke X".

Aturan tambahan:

- Ternary dan aggregation → buat **vertex perantara**.
- Atribut multivalued → properti list, atau vertex sendiri bila nilainya perlu ditelusuri (mis. "user yang hobinya sama").
- ISA → label tambahan (vertex boleh punya beberapa label, cocok untuk overlapping). Multi-label dan properti berupa list adalah fitur property graph seperti Neo4j *(tambahan)*; alternatifnya properti `type`.
- Relasi "A dibuat oleh B" dan "B membuat A" cukup **satu edge**.
- Waspadai **cycle** saat menelusuri (mis. rantai prasyarat).
- **Tulis:** daftar vertex (label, properti), daftar edge (label, arah, properti), gambar kecil, dan pemetaan tiap query. Contoh Cypher boleh ditambahkan.

3.6 Soal Use Case Key-Value

1. **Cari kebutuhan yang cocok:** read/write kecil dan sering, model data sederhana, dan diakses **hanya lewat key** (point lookup). Contoh dari materi: cache, session, shopping cart, konfigurasi user, objek besar, penahanan sementara dengan **TTL**.
2. **Rancang key** dengan pola `<entity>:<id>:<atribut>` dan delimiter konsisten. Pertimbangkan namespace.
3. **Rancang value:** gabungkan atribut yang selalu dibaca bersama dalam satu value terstruktur (bagian "designing structured values" di materi, contoh `cust:198277:nameAddr`). Hindari value terlalu besar. Jangan tertukar dengan nama pola: **aggregates** = satu entity type untuk beberapa subtipe beratribut berbeda + indikator tipe (cocok untuk ISA); **atomic aggregates** = nilai yang harus berubah bersamaan ditulis dalam satu assignment.
4. **Sebutkan operasi** (get/set/expire) dan **alasan** memakai key-value, bukan model lain.
5. Sebut pola dari materi bila cocok: TTL keys, emulating tables, aggregates, atomic aggregates, enumerable keys, inverted index.

3.7 Soal XML (Skema/DTD dan Query)

1. **Tentukan elemen root** (satu elemen top-level), lalu struktur bersarang mengikuti **bentuk hasil query** yang diminta.
2. **Atribut vs subelemen:** atribut untuk **identifier** (ID), subelemen untuk **isi** (saran materi).
3. **Kemunculan:** wajib (default), opsional (`?` / `minOccurs="0"`), banyak (`*` atau `+` / `maxOccurs="unbounded"`).
4. **Pastikan memuat semua data yang dibutuhkan query di soal berikutnya** (filter, urutan).
5. **DTD vs XML Schema:** DTD lebih singkat tetapi tanpa tipe data (semua teks), sehingga tanggal ditulis `YYYY-MM-DD` agar urutan teksnya benar. XML Schema punya tipe (`xs:date`, `xs:integer`) dan key/keyref, tetapi lebih panjang. Jika boleh memilih dan waktunya sempit, **DTD lebih cepat ditulis**.
6. **Query: XPath** untuk seleksi (`/root/elemen[kondisi]/@atribut`); **XQuery FLWOR** untuk seleksi + bentuk hasil: `for` (iterasi), `let` (variabel), `where` (filter), `order by` (urutan), `return` (bentuk hasil; isi dalam `{ }` dievaluasi).

3.8 Soal "Pilih Model Mana" (Choosing a Database)

Query mendorong desain. Faktor lain: volume read/write, toleransi data tidak konsisten di replika, sifat relasi antar entity, availability dan disaster recovery, kebutuhan fleksibilitas, dan latency.

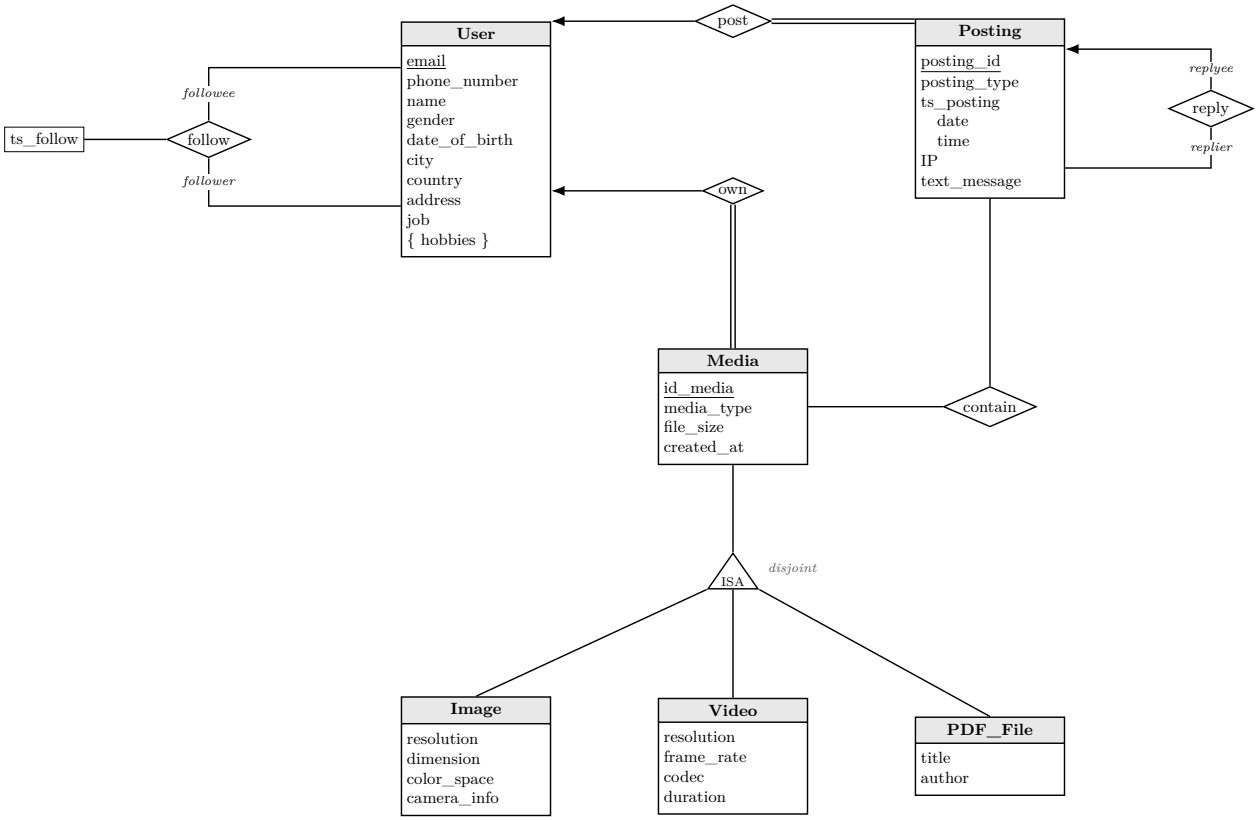
Kebutuhan	Model
Integritas kuat, ACID, relasi terstruktur, transaksi	Relasional
Cache, session, cart, konfigurasi, lookup lewat key	Key-value
Atribut bervariasi, JSON, data sering dibaca sebagai satu kesatuan (embedding)	Dokumen
Data sangat besar, tulis sangat tinggi, multi data center, toleran inkonsistensi singkat	Column-family
Banyak relasi antar instance (jaringan, rekomendasi, sosial)	Graf
Semi-terstruktur bersarang, pertukaran data lintas sistem, dokumen terregulasi	XML
Butuh skala NoSQL tetapi tetap ACID untuk OLTP	NewSQL

Satu aplikasi **boleh memakai beberapa model sekaligus** (memilih alat yang tepat untuk tiap pekerjaan; tidak ada DBMS terbaik).

4 Jawaban Lengkap UTS Sample

4.1 Analisis Kasus (Langkah 3.1)

Diagram ER sample, digambar ulang dengan notasi yang sama seperti slide:



Informasi tambahan dari soal: address, hobi, dan job boleh tidak diisi; atribut user lain wajib. `media_type` bernilai image, video, pdf file, dll. (dimungkinkan ada jenis lain). `posting_type` bernilai normal atau reply.

Konstruksi ER yang ditemukan:

Elemen	Konstruksi	Konsekuensi
User.{hobbies}	Multivalued, opsional	Tabel/array terpisah; boleh kosong
User.address, job	Opsional	Boleh NULL/tidak ada
Posting.ts_posting (date, time)	Composite	Diratakan menjadi dua kolom
post: panah ke User, Posting garis ganda	Many-to-one, Posting total	FK `email` NOT NULL di Posting
own: panah ke User, Media garis ganda	Many-to-one, Media total	FK pemilik NOT NULL di Media
follow: tanpa panah, atribut ts_follow, role followee/follower	M:N rekursif beratribut	Tabel relasi dua FK ke User
reply: panah ke sisi replyee, role replyee/replier	Many-to-one rekursif (satu reply menanggapi paling banyak satu posting)	FK `replyee_id` di Posting (atau tabel terpisah)
contain: tanpa panah	M:N Posting–Media	Tabel relasi
Media ISA Image/Video/PDF_File, disjoint, garis tunggal	Disjoint, **partial** (ada jenis lain: “dll.”)	Superclass harus tetap punya tabel sendiri
posting_type: normal/reply	Domain terbatas, berkaitan dengan reply	CHECK + konsistensi dengan replyee

Query sering ditulis ulang:

- **Q1:** ambil User berdasarkan email → {email, name, phone_number, gender, date_of_birth} + daftar follower {email, name}.
- **Q2:** ambil **semua** Media beserta atribut khusus tiap jenisnya (dibaca bersama).
- **Q3:** ambil satu Posting milik seorang User → data posting + media terkait (id_media, media_type, file_size, created_at) + daftar posting_id reply-nya.

Asumsi umum: email unik dan tidak berubah; data media standar (ukuran, waktu dibuat) tidak berubah setelah dibuat; nama user jarang berubah; satu reply menanggapi tepat satu posting.

4.2 Soal 1: Model Relasional

Langkah berpikir

Petakan tiap baris tabel analisis kasus dengan kolom "Relasional" di tabel konversi (bagian 2.2).

Model dasar (normal):

```
User(email, phone_number, name, gender, date_of_birth, city, country,
     address NULL, job NULL)
User_Hobby(email, hobby)                          FK email -> User
Follow(followee_email, follower_email, ts_follow)
                                                    FK followee_email -> User
                                                    FK follower_email -> User
Posting(posting_id, posting_type, ts_date, ts_time, IP, text_message,
        email NOT NULL, replyee_id NULL)
                                                    FK email -> User
                                                    FK replyee_id -> Posting
Media(id_media, media_type, file_size, created_at, owner_email NOT NULL)
                                                    FK owner_email -> User
Image(id_media, resolution, dimension, color_space, camera_info)
                                                    FK id_media -> Media
Video(id_media, resolution, frame_rate, codec, duration)
                                                    FK id_media -> Media
PDF_File(id_media, title, author)              FK id_media -> Media
Contain(posting_id, id_media)
                                                    FK posting_id -> Posting
                                                    FK id_media -> Media
```

Constraint tambahan:

- `CHECK (posting_type IN ('normal','reply'))`.
- Konsistensi: `posting_type = 'reply'` jika dan hanya jika `replyee_id` tidak NULL (CHECK).
- Atribut user selain address, job, dan hobi: NOT NULL.

Reasoning yang ditulis

1. **Hobi multivalued** dipisah ke `User_Hobby` (nilai atribut relasional harus atomik); karena opsional, user tanpa hobi cukup tidak punya baris.
2. **ts_posting komposit** diratakan menjadi `ts_date` dan `ts_time`.
3. **post** dan **own** adalah many-to-one dengan partisipasi total di sisi Posting dan Media, sehingga cukup FK NOT NULL di sisi "many" tanpa tabel relasi.
4. **follow** adalah M:N rekursif dengan atribut, sehingga menjadi tabel relasi; PK gabungan dua email dengan nama sesuai role.
5. **reply** adalah many-to-one rekursif (panah ke replyee), sehingga cukup FK `replyee_id` ke Pos-

ting sendiri. NULL untuk posting normal. Alternatif tanpa NULL: tabel `Reply(replier_id, replyee_id)` dengan PK `replier_id`.

6. **ISA disjoint dan partial**: dipakai pola superclass + tabel per subclass (aturan specialization di materi). Tabel superclass `Media` wajib ada karena jenis media lain (selain image, video, PDF) tidak punya tabel subclass. `media_type` berfungsi sebagai penanda subclass (disjoint: satu media satu jenis).
7. **contain** adalah M:N, sehingga menjadi tabel relasi.

Model teroptimasi (opsional, sertakan bila waktu cukup):

Optimasi	Alasan (query)	Trade-off
Gabung `Media`, `Image`, `Video`, `PDF_File` menjadi **satu tabel Media** dengan semua atribut subclass (NULL bila tidak relevan)	Q2 menampilkan semua jenis media bersama, jadi tanpa join	Banyak NULL; jenis media baru dengan atribut baru butuh `ALTER TABLE`; constraint per jenis lebih sulit
Tambah `follower_name` di `Follow`	Q1 mengambil follower (email, nama) tanpa join ke User	Redundansi: nama berubah harus diupdate di semua baris Follow (anomali update). Masuk akal bila nama jarang berubah
Vertical partitioning User: `User_Main(email, name, phone_number, gender, date_of_birth)` dan `User_Detail(email, city, country, address, job)`	Q1 hanya butuh atribut utama; baris lebih kecil, lebih banyak baris per pembacaan block	Data user lengkap butuh join 1:1
Index pada `Posting(replyee_id)` dan `Contain(posting_id)`	Q3 mencari reply dan media sebuah posting	Ruang dan biaya tulis index

4.3 Soal 2a: Desain Dokumen

Langkah berpikir: embed atau reference per relasi

Relasi	Keputusan	Alasan (empat faktor)
User–hobi (multivalued)	**Embed** array	Kecil, bagian dari profil
User–follower (follow)	**Embed** array follower {email, name, ts_follow} di dokumen user, dengan asumsi jumlah follower wajar	Q1 membaca profil + follower bersama. Bila follower bisa jutaan, dokumen tumbuh tanpa batas, jadi pindahkan ke collection `follows` (reference)
User–Posting (post)	**Reference**: posting adalah dokumen sendiri dengan `author_email`	Posting banyak dan terus bertambah, diakses mandiri (Q3)
User–Media (own)	**Reference**: `owner_email` di media	Media diakses mandiri (Q2) dan berjumlah banyak
Posting–Media (contain)	**Reference + salinan atribut standar** media di posting	M:N, jadi media tetap collection sendiri. Q3 butuh atribut standar bersama posting, dan atribut itu tidak berubah setelah dibuat, sehingga aman disalin
Posting–reply (rekursif)	**Parent reference** (`reply_to`) di posting reply, dan **children references** (`replies`: array ID) di posting induk	Q3 butuh daftar ID reply. Kedua arah memudahkan navigasi
Media ISA	**Satu collection polymorphic** `media`	Q2 menampilkan semua jenis bersama (materi: subtipe dalam satu collection bila sering diagregasi bersama); jenis baru tidak butuh perubahan skema

Desain (komentar `//` hanya penjelasan):

```
// collection: users
{ "_id": "fazat@informatika.org",
  "name": "Fazat", "phone_number": "0812...", "gender": "F",
  "date_of_birth": "1990-01-01", "city": "Bandung", "country": "Indonesia",
  "address": "...", "job": "Dosen",                  // opsional
  "hobbies": ["membaca", "berenang"],                // opsional (multivalued)
  "followers": [                                     // embed, untuk Q1
    { "email": "a@x.org", "name": "Andi", "ts_follow": "2026-09-01T10:00" } ] }

// collection: media (polymorphic)
{ "_id": "M1", "media_type": "image", "file_size": 2048,
  "created_at": "2026-09-02T08:00", "owner_email": "fazat@informatika.org",
  "resolution": "1920x1080", "dimension": "...", "color_space": "sRGB", "camera_info":
    "..." }
{ "_id": "M2", "media_type": "pdf file", "file_size": 512,
  "created_at": "2026-09-03T09:00", "owner_email": "fazat@informatika.org",
  "title": "Silabus", "author": "Fazat" }

// collection: postings
{ "_id": "P10", "posting_type": "normal",
  "ts_posting": { "date": "2026-09-05", "time": "10:00" },
  "IP": "10.0.0.1", "text_message": "Halo",
  "author_email": "fazat@informatika.org",
  "media": [                                         // salinan atribut standar, untuk Q3
    { "id_media": "M1", "media_type": "image", "file_size": 2048,
      "created_at": "2026-09-02T08:00" } ],
  "reply_to": null,                                  // diisi ID induk bila reply
  "replies": ["P11", "P12"] }                        // children references, untuk Q3
```

Index: `postings.author_email` (posting milik user), `media.media_type` (Q2 per jenis), `postings.reply_to`.

Reasoning yang ditulis

- Q1 cukup satu baca dokumen user karena follower di-embed.
- Q2 cukup membaca satu collection polymorphic.
- Q3 cukup satu baca dokumen posting karena atribut standar media dan ID reply sudah ada di dalamnya.
- Redundansi (nama follower, atribut standar media) disengaja demi performa dan dipilih dari atribut yang jarang berubah.
- Integritas (follower masih ada, ID media valid) dijaga aplikasi karena schemaless.

4.4 Soal 2b: Desain Column-Family (BigTable-like)

Langkah berpikir

Satu tabel per kebutuhan baca utama; row key dari kriteria pencarian; kolom yang dibaca bersama dalam satu CF.

```
Keyspace: socmed

Tabel User                          (untuk Q1)
  Row key      : email
  CF profile   : phone_number, name, gender, date_of_birth
  CF detail    : city, country, address, job          (jarang dibaca bersama Q1)
  CF hobby     : <nama_hobi> = ""                      (valueless columns)
  CF follower  : <email_follower> = <nama_follower>    (nama kolom + nilai berisi data)
  CF follow_ts : <email_follower> = <ts_follow>

Tabel Media                         (untuk Q2)
  Row key      : id_media
  CF basic     : media_type, file_size, created_at, owner_email
  CF ext       : kolom dinamis sesuai jenis:
                 image -> resolution, dimension, color_space, camera_info
                 video -> resolution, frame_rate, codec, duration
                 pdf   -> title, author

Tabel Posting                       (untuk Q3)
  Row key      : posting_id   (di-hash/beri prefix bila sekuensial, cegah hotspotting)
  CF post      : posting_type, ts_date, ts_time, IP, text_message, author_email, reply_to
  CF media     : <id_media>:type, <id_media>:size, <id_media>:created  (kolom dinamis)
  CF reply     : <posting_id_reply> = ""                               (valueless)

Tabel Posting_by_User               (index buatan, mencari posting milik user)
  Row key      : email
  CF posting   : <posting_id> = ""
```

Contoh satu row tabel User:

Row key	profile:name	profile:gender	follower:a@x.org	follower:b@y.org	hobby:membaca
fazat@informatika.org	Fazat	F	Andi	Budi	(kosong)

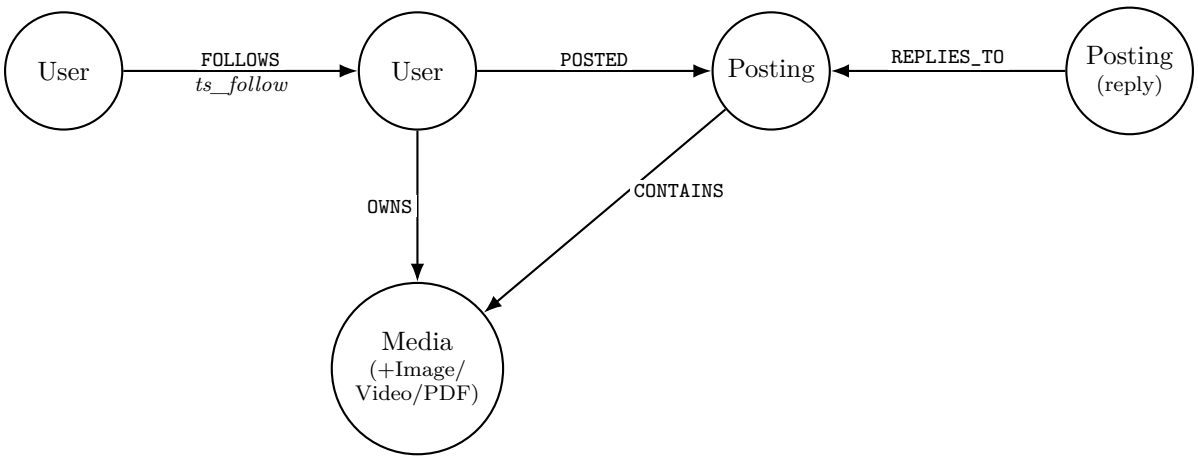
Reasoning yang ditulis

- **Q1 = satu row**: CF `profile` dan `follower` dibaca sekaligus. Atribut yang jarang dibaca dipisah ke CF `detail`, sehingga tidak ikut terbaca.
- **Follow (M:N)** ditangani dengan **denormalize instead of join**: email follower menjadi nama kolom dan nama follower menjadi nilainya (guideline 2 dan 3).
- **ISA**: satu tabel Media, dengan atribut khusus jenis sebagai **kolom dinamis** di CF `ext`. Row yang berbeda boleh punya kolom berbeda (sparse), dan jenis baru tidak butuh perubahan skema.
- **Q3 = satu row**: posting, atribut standar media (salinan), dan ID reply ada di row yang sama. Satu entity dalam satu row juga membuat tulis bersifat atomik (guideline 4).
- **Value tetap sederhana** (guideline 7): atribut media disimpan sebagai kolom terpisah, bukan objek bersarang.
- **Mencegah hotspotting**: row key Posting di-hash bila posting_id sekuensial.
- **Pencarian posting per user** memakai tabel index buatan sendiri, seperti `Cust_by_Prod` di materi.

4.5 Soal 2c: Desain Graf

Vertex (label)	Properti
User	email (ID), name, phone_number, gender, date_of_birth, city, country, address?, job?, hobbies? (list)
Posting	posting_id (ID), posting_type, ts_date, ts_time, IP, text_message
Media (+ label Image / Video / PDF_File)	id_media (ID), media_type, file_size, created_at, plus properti khusus jenis

Edge	Dari → Ke	Properti	Asal ER
FOLLOWS	User (follower) → User (followee)	ts_follow	follow
POSTED	User → Posting		post
OWNS	User → Media		own
CONTAINS	Posting → Media		contain
REPLIES_TO	Posting (replier) → Posting (replyee)		reply



Pemetaan query:

- **Q1:** vertex User dengan email tertentu, lalu vertex di ujung edge **FOLLOWS yang masuk** ke vertex itu (follower).
- **Q2:** semua vertex berlabel Media (properti khusus jenis ikut terambil).
- **Q3:** vertex Posting dengan edge POSTED dari user tersebut, lalu tetangga lewat CONTAINS (media), lalu posting_id dari vertex yang punya edge REPLIES_TO masuk ke posting itu.

Contoh Cypher Q3 *(tambahan)*, tidak wajib:

```
MATCH (u:User {email: $email})-[:POSTED]->(p:Posting {posting_id: $id})
OPTIONAL MATCH (p)-[:CONTAINS]->(m:Media)
OPTIONAL MATCH (r:Posting)-[:REPLIES_TO]->(p)
RETURN p, collect(DISTINCT m) AS media, collect(DISTINCT r.posting_id) AS replies
```

Reasoning yang ditulis

- Semua relasi di kasus (follow, post, own, contain, reply) menjadi edge, sehingga query **mengikuti edge** tanpa join.
- FOLLOWS dan REPLIES_TO **berarah** karena tidak simetris (A mengikuti B tidak berarti B mengikuti A).
- Atribut relasi ts_follow menjadi **properti edge**.
- ISA ditangani dengan label tambahan per jenis.
- Hobi disimpan sebagai properti list karena tidak ada query yang menelusurinya. Bila ada kebutuhan "user dengan hobi sama", jadikan Hobby vertex.

4.6 Soal 3: Use Case Key-Value

Jawaban utama: cache "kartu profil" untuk Q1.

- **Key:** `user:<email>:profileCard`, mis. `user:fazat@informatika.org:profileCard`.
- **Value:** satu value terstruktur (structured value) berisi `{name, phone_number, gender, date_of_birth, followers: [{email, name}]}`.
- **Operasi:** saat profil dibuka: `get(key)`; bila tidak ada: ambil dari basis data utama lalu `set(key, value)` dengan **TTL**, mis. 10 menit; saat profil atau follower berubah: hapus key-nya.

Reasoning yang ditulis

- Q1 sangat sering (read-heavy) dan selalu mencari berdasarkan satu key (email), sehingga cocok untuk point lookup key-value di memori.
- Atribut yang selalu dibaca bersama digabung dalam satu value, sehingga cukup satu akses.
- Key-value tidak perlu query kompleks di kasus ini.

Alternatif lain yang juga sah: session login `session:<token>` dengan TTL; counter jumlah follower `user:<email>:followerCount` (increment saat follow); draft posting yang belum dikirim `draft:<email>`.

4.7 Soal 4a: XML Schema atau DTD untuk Hasil Q3

Langkah berpikir

Hasil Q3 = posting + media standar + ID reply. Soal 4b memfilter berdasarkan email, sehingga **email penulis harus ada**. Identifier dijadikan atribut, isi dijadikan subelemen.

Versi DTD:

```
<!ELEMENT postings (posting*)>
<!ELEMENT posting (posting_type, ts_posting, IP, text_message, media*, reply*)>
<!ATTLIST posting
    posting_id   ID     #REQUIRED
    author_email CDATA  #REQUIRED>
<!ELEMENT posting_type (#PCDATA)>
<!ELEMENT ts_posting (date, time)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT IP (#PCDATA)>
<!ELEMENT text_message (#PCDATA)>
<!ELEMENT media (media_type, file_size, created_at)>
<!ATTLIST media id_media CDATA #REQUIRED>
<!ELEMENT media_type (#PCDATA)>
<!ELEMENT file_size (#PCDATA)>
<!ELEMENT created_at (#PCDATA)>
<!ELEMENT reply EMPTY>
<!ATTLIST reply posting_id CDATA #REQUIRED>
```

Catatan DTD

- Nilai bertipe `ID` harus diawali huruf (mis. `P10`, bukan `10`).
- `reply` memakai `CDATA`, bukan `IDREF`. Hasil Q3 hanya memuat **ID** reply, bukan posting reply-nya, sedangkan `IDREF` wajib menunjuk `ID` yang ada di dokumen yang sama. Pakai `IDREF` hanya bila posting reply juga ikut di dokumen.

Versi XML Schema (pilih salah satu di ujian):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="postings">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="posting" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="posting_type" type="xs:string"/>
              <xs:element name="ts_posting">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="date" type="xs:date"/>
                    <xs:element name="time" type="xs:time"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="IP" type="xs:string"/>
              <xs:element name="text_message" type="xs:string"/>
              <xs:element name="media" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="media_type" type="xs:string"/>
                    <xs:element name="file_size" type="xs:integer"/>
```

```
                    <xs:element name="created_at" type="xs:dateTime"/>
                  </xs:sequence>
                  <xs:attribute name="id_media" type="xs:string" use="required"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="reply" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="posting_id" type="xs:string" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="posting_id" type="xs:string" use="required"/>
            <xs:attribute name="author_email" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:key name="postingKey">
      <xs:selector xpath="posting"/>
      <xs:field xpath="@posting_id"/>
    </xs:key>
  </xs:element>
</xs:schema>
```

Catatan tipe: nilai `xs:time` dan `xs:dateTime` wajib memakai detik (`10:00:00`, `2026-09-02T08:00:00`).

Reasoning yang ditulis

- Satu root `postings`.
- Identifier (posting_id, id_media, author_email) sebagai **atribut**; isi sebagai **subelemen**.
- `ts_posting` komposit menjadi elemen bersarang.
- Media dan reply **berulang 0..n**.
- `author_email` disertakan agar data bisa difilter per pengguna (soal 4b).

4.8 Soal 4b: XQuery

```
for $p in doc("postings.xml")/postings/posting
where $p/@author_email = "fazat@informatika.org"
order by $p/ts_posting/date descending, $p/ts_posting/time descending
return $p
```

Versi yang membentuk hasil sendiri (bila ingin hanya data tertentu):

```
for $p in doc("postings.xml")/postings/posting
where $p/@author_email = "fazat@informatika.org"
return <posting id="{ $p/@posting_id }">
         { $p/text_message }
         <media_count>{ count($p/media) }</media_count>
         { $p/reply }
       </posting>
```

Padanan XPath (tanpa membentuk ulang hasil):

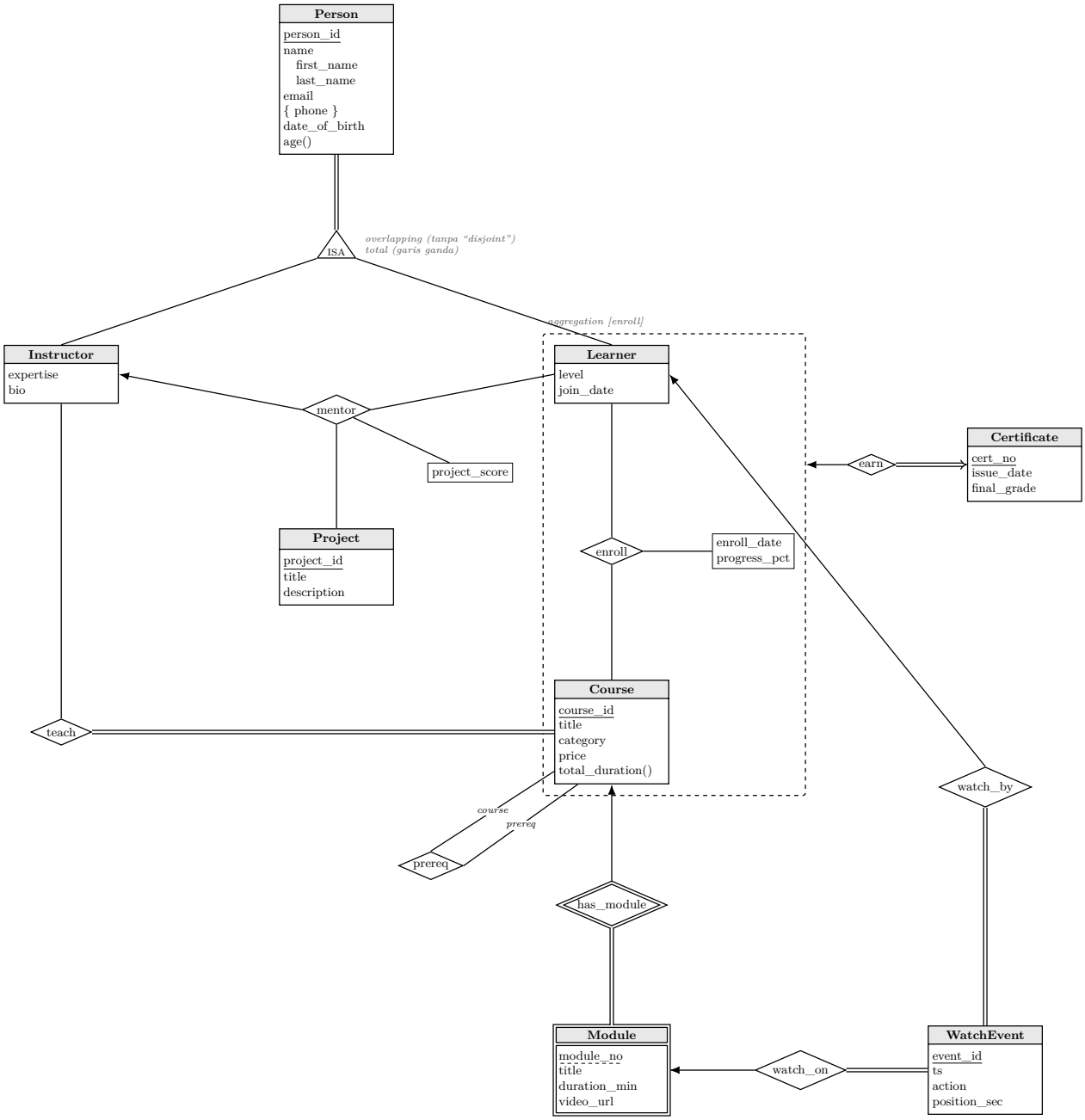
```
/postings/posting[@author_email = "fazat@informatika.org"]
```

5 Studi Kasus Prediksi: KursusKita

Kasus ini sengaja memuat konstruksi ER yang **tidak ada** di sample: weak entity, ISA **overlapping** dan **total**, atribut derived, relasi **ternary**, **aggregation**, dan relasi **1:1**. Query-nya juga berbeda karena menguji data deret waktu (column-family) dan penelusuran graf. Polanya mengikuti contoh di materi: diagram universitas (section, prereq, takes, proj_guide, eval_for), contoh truk untuk dokumen yang tumbuh, dan jejaring developer untuk graf. Kasus ini karangan untuk latihan, bukan soal dosen.

5.1 Soal

Deskripsi. KursusKita adalah platform kursus online. Bagian model ER-nya:



Entity set	Atribut	Catatan
Person	person_id (kunci), name (komposit: first_name, last_name), email, { phone } (multivalued), date_of_birth, age() (derived)	
Instructor (ISA)	expertise, bio	
Learner (ISA)	level, join_date	
Course	course_id (kunci), title, category, price, total_duration() (derived)	
Module (weak)	module_no (discriminator), title, duration_min, video_url	Double rectangle
WatchEvent	event_id (kunci), ts, action, position_sec	
Project	project_id (kunci), title, description	
Certificate	cert_no (kunci), issue_date, final_grade	

Relationship	Menghubungkan	Catatan
ISA	Person → Instructor, Learner	**Overlapping** (tanpa tulisan disjoint), **total** (garis ganda)
has_module	Course – Module	**Identifying relationship** (double diamond); Module total
teach	Instructor – Course	M:N; Course total (setiap course minimal satu instructor)
enroll	Learner – Course	M:N; atribut enroll_date, progress_pct
prereq	Course (course) – Course (prereq)	M:N rekursif
watch_by	WatchEvent → Learner	Many-to-one, WatchEvent total
watch_on	WatchEvent → Module	Many-to-one, WatchEvent total
mentor	Instructor, Learner, Project (**ternary**)	Panah ke Instructor (tiap pasangan learner–project paling banyak satu mentor); atribut project_score
earn	**Aggregation** [enroll] – Certificate	1:1 (panah di kedua sisi); Certificate total

Informasi tambahan: phone dan bio boleh kosong; atribut lain wajib. level bernilai beginner, intermediate, atau advanced. Seseorang bisa menjadi instructor sekaligus learner. WatchEvent sangat banyak (jutaan per hari), hanya ditambahkan, tidak pernah diubah.

Query yang sering terjadi:

1. Halaman course: data course + daftar module terurut (module_no, title, duration_min) + nama instructor + judul course prasyarat langsung.
2. Dashboard learner: daftar course yang diikuti beserta progress_pct, dan posisi terakhir menonton (module dan position_sec) per course untuk fitur "lanjutkan menonton".
3. (a) Riwayat menonton seorang learner dalam rentang waktu, terurut waktu; (b) jumlah event `play` per module per hari.
4. (a) Semua prasyarat langsung maupun tidak langsung sebuah course; (b) rekomendasi: course lain yang diambil learner yang mengambil course yang sama denganku.
5. Kirim data sertifikat ke mitra dalam XML: cert_no, issue_date, final_grade, nama learner, email learner, judul course, dan daftar judul module.

Soal

1. Buat model relasional murni beserta *reasoning*. Boleh dioptimasi dengan asumsi.
2. Buat desain logis (a) dokumen, (b) column-family BigTable-like, (c) graf, beserta *reasoning*.

3. Berikan 1 use case key-value.
4. (a) Buat **DTD** untuk keperluan query 5. (b) Tulis XQuery untuk sertifikat milik learner dengan email `ana@kursuskita.id`, terurut dari yang terbaru. (c) Tulis XPath untuk mengambil cert_no semua sertifikat dengan final_grade "A".
5. Untuk tiap query 1–5, model basis data mana yang paling cocok? Jelaskan.

5.2 Analisis Kasus

Elemen	Konstruksi	Konsekuensi
Person.name	Composite	first_name, last_name
Person.{phone}	Multivalued, opsional	Tabel/array terpisah
age(), total_duration()	Derived	Tidak disimpan (relasional); boleh disimpan di NoSQL bila sering dibaca
ISA overlapping, total	Satu orang bisa di dua subclass; tiap orang wajib di minimal satu	Tabel per subclass; satu kolom "type" tidak cukup; total ditegakkan aplikasi/trigger
Module	Weak entity	PK = (course_id, module_no)
WatchEvent	Many-to-one total ke Learner dan Module	Dua FK NOT NULL; FK ke Module berbentuk gabungan (course_id, module_no)
teach, enroll, prereq	M:N (prereq rekursif)	Tabel relasi
mentor	Ternary, panah ke Instructor	PK = (learner_id, project_id)
earn	Aggregation dari enroll, 1:1, Certificate total	FK (learner_id, course_id) di Certificate, UNIQUE

5.3 Jawaban Soal 1: Model Relasional

```
Person(person_id, first_name, last_name, email UNIQUE NOT NULL, date_of_birth)
Person_Phone(person_id, phone)                      FK person_id -> Person
Instructor(person_id, expertise, bio NULL)              FK person_id -> Person
Learner(person_id, level, join_date)                    FK person_id -> Person
Course(course_id, title, category, price)
Module(course_id, module_no, title, duration_min, video_url)
                                                      FK course_id -> Course
Teach(instructor_id, course_id)
                                FK instructor_id -> Instructor, FK course_id -> Course
Enroll(learner_id, course_id, enroll_date, progress_pct)
                                FK learner_id -> Learner, FK course_id -> Course
Prereq(course_id, prereq_id)   FK course_id -> Course, FK prereq_id -> Course
WatchEvent(event_id, learner_id NOT NULL, course_id NOT NULL, module_no NOT NULL,
           ts, action, position_sec)
                                FK learner_id -> Learner
                                FK (course_id, module_no) -> Module
Project(project_id, title, description)
Mentor(learner_id, project_id, instructor_id NOT NULL, project_score)
                                FK learner_id -> Learner, FK project_id -> Project
                                FK instructor_id -> Instructor
Certificate(cert_no, issue_date, final_grade, learner_id NOT NULL, course_id NOT NULL)
                                FK (learner_id, course_id) -> Enroll
                                UNIQUE (learner_id, course_id)
```

Constraint: `CHECK (level IN ('beginner','intermediate','advanced'))`; atribut wajib NOT NULL.

Reasoning yang ditulis

1. **name** komposit diratakan; **phone** multivalued dipisah ke `Person_Phone`.
2. **age() dan total_duration()** adalah atribut derived, jadi tidak disimpan. Nilainya dihitung lewat function/view: age dari date_of_birth, total_duration = SUM duration_min module.
3. **ISA overlapping** dipetakan dengan tabel superclass + tabel per subclass. Seseorang yang menjadi instructor dan learner cukup punya baris di kedua tabel. Satu kolom `type` tidak bisa menyatakan "keduanya". **Total** (setiap person wajib di minimal satu subclass) tidak bisa ditegakkan FK biasa, sehingga dijaga lewat trigger atau aplikasi.
4. **Module weak entity:** PK gabungan dengan PK Course (identifying) + discriminator module_no.
5. **teach, enroll, prereq (M:N)** menjadi tabel relasi. **Course total di teach** (minimal satu instructor) tidak bisa dijamin FK, sehingga dijaga lewat trigger atau aplikasi.
6. **watch_by dan watch_on (many-to-one total)** cukup FK NOT NULL di WatchEvent; FK ke Module berbentuk gabungan.
7. **mentor ternary** dengan panah ke Instructor: tiap pasangan (learner, project) paling banyak satu instructor, sehingga PK = (learner_id, project_id) dan instructor_id menjadi kolom biasa NOT NULL.
8. **earn** adalah aggregation 1:1 dengan Certificate total. FK diletakkan di sisi yang total (Certificate) agar tidak ada NULL. UNIQUE menjamin satu enrollment paling banyak satu sertifikat. Bentuk umum aggregation di materi adalah tabel relasi terpisah (seperti `eval_for`); di sini tabel Earn dilipat ke Certificate karena relasinya 1:1, dengan aturan 1:1 yang sama seperti relasi biasa.

Optimasi (opsional):

- **Simpan posisi terakhir menonton di Enroll** (`last_module_no`, `last_position_sec`). Q2 tidak perlu mencari event terakhir di antara jutaan WatchEvent. Trade-off: redundansi, dan setiap event (atau secara berkala) harus memperbarui Enroll.
- **Index** `WatchEvent(learner_id, ts)` untuk Q3a. (Urutan module di Q1 sudah terlayani oleh PK `Module(course_id, module_no)`, yang otomatis terindeks.)
- Q3b (agregasi per hari) bisa memakai **materialized view** yang di-refresh berkala.

5.4 Jawaban Soal 2a: Desain Dokumen

Langkah berpikir: keputusan per relasi

Relasi/konstruksi	Keputusan	Alasan
ISA overlapping	Satu collection `persons` dengan sub-dokumen `instructor` dan/atau `learner` + array `roles`	1:1 ke tiap peran (embed); overlapping cukup dengan dua sub-dokumen
Module (weak)	**Embed** array di `courses`	Eksistensinya bergantung pada course, jumlahnya kecil dan terbatas, selalu dibaca bersama di Q1
teach, prereq (M:N)	Reference + **salinan nama/judul** di `courses`	Q1 butuh nama instructor dan judul prasyarat; nama dan judul jarang berubah
enroll (M:N beratribut, sering diupdate)	Collection sendiri `enrollments`	Diakses per learner (Q2), progress sering berubah, jumlahnya terus bertambah
earn (1:1 dengan enroll)	**Embed** `certificate` di `enrollments`	1:1 berarti embed
WatchEvent	Collection `watch_events` dengan pola **satu dokumen per learner per hari** berisi array event	Volume sangat besar; mengikuti contoh truk di materi (ruang array dialokasikan cukup sejak dokumen dibuat agar dokumen tidak berpindah)
mentor (ternary)	Collection sendiri `mentorships` berisi tiga referensi	Ternary tidak bisa di-embed ke satu sisi saja tanpa duplikasi

```
// persons
{ "_id": "P1", "name": {"first_name": "Ana", "last_name": "Putri"},
  "email": "ana@kursuskita.id", "phones": ["0812..."], "date_of_birth": "2003-05-01",
  "roles": ["learner", "instructor"],
  "learner": {"level": "intermediate", "join_date": "2025-01-10"},
  "instructor": {"expertise": "SQL", "bio": "..."} }

// courses
{ "_id": "C10", "title": "Basis Data Lanjut", "category": "Data", "price": 150000,
  "modules": [ {"module_no": 1, "title": "ER", "duration_min": 30, "video_url": "..."},
               {"module_no": 2, "title": "NoSQL", "duration_min": 45, "video_url": "..."}
    ],
  "instructors": [ {"person_id": "P7", "name": "Budi Santoso"} ],
  "prerequisites": [ {"course_id": "C3", "title": "Basis Data"} ] }

// enrollments
{ "_id": "E55", "learner_id": "P1",
  "course": {"course_id": "C10", "title": "Basis Data Lanjut"},
  "enroll_date": "2026-08-01", "progress_pct": 40,
  "last_watch": {"module_no": 2, "position_sec": 615, "ts": "2026-10-12T20:15"},
  "certificate": {"cert_no": "K001", "issue_date": "2026-10-01", "final_grade": "A"} }

// watch_events (satu dokumen per learner per hari)
{ "_id": "P1#2026-10-12", "learner_id": "P1", "date": "2026-10-12",
  "events": [ {"ts": "20:10:00", "course_id": "C10", "module_no": 2,
               "action": "play", "position_sec": 0} ] }

// mentorships
{ "_id": "MT1", "learner_id": "P1", "project": {"project_id": "PR4", "title": "Capstone"},
  "instructor": {"person_id": "P7", "name": "Budi Santoso"}, "project_score": 88 }
```

Index: `enrollments.learner_id` (Q2), `watch_events.learner_id + date` (Q3a), `persons.email`.

Reasoning yang ditulis

Q1 = satu dokumen course. Q2 = dokumen enrollments milik learner, sudah berisi posisi terakhir. Q3a = dokumen per hari dalam rentang tanggal. Q3b dan Q4 kurang cocok di dokumen (agregasi besar dan penelusuran rekursif); lihat soal 5.

5.5 Jawaban Soal 2b: Desain Column-Family

```
Keyspace: kursuskita

Tabel Course                          (Q1)
  Row key       : course_id
  CF info       : title, category, price, total_duration  (derived, disimpan saat module
    berubah)
  CF module     : m001_title, m001_dur, m002_title, m002_dur, ...
                  (module_no diberi nol di depan agar terurut)
  CF instructor : <person_id> = <nama>
  CF prereq     : <course_id_prasyarat> = <judul>

Tabel Learner_Course                  (Q2)
  Row key       : learner_id
  CF enroll     : <course_id>:title, <course_id>:progress, <course_id>:enroll_date
  CF resume     : <course_id>:module, <course_id>:pos

Tabel WatchEvent                      (Q3a, deret waktu)
  Row key       : learner_id#ts#event_id    (mis. P1#20261012T201000#E9812)
  CF ev         : course_id, module_no, action, position_sec

Tabel PlayCount                       (Q3b, nilai turunan yang dihitung saat tulis)
  Row key       : course_id#module_no
  CF day        : <yyyymmdd> = <jumlah play>   (diincrement setiap event play)
```

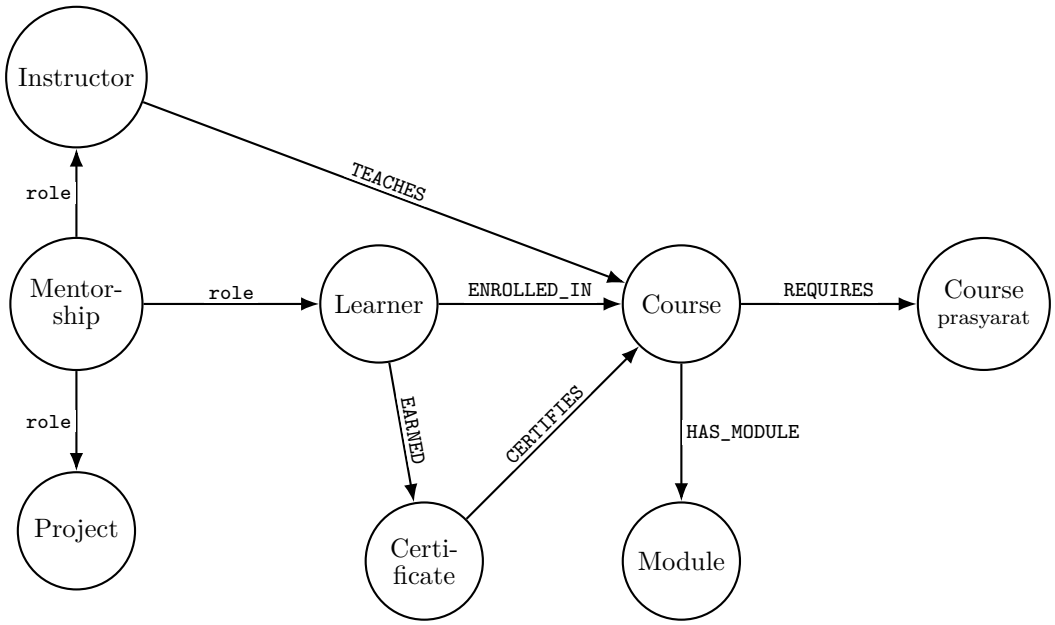
Reasoning yang ditulis

- **Q3a:** row tersimpan terurut menurut row key, sehingga `learner_id#ts` membuat riwayat seorang learner berdekatan dan terurut waktu. Query 1–13 Oktober menjadi **range scan** dari `P1#20261001` sampai **sebelum** `P1#20261014` (batas akhir scan eksklusif). `event_id` di akhir row key mencegah dua event pada detik yang sama saling menimpa. Prefix `learner_id` juga menyebarkan tulisan sehingga tidak terjadi hotspotting; row key yang diawali waktu saja akan menumpuk semua tulisan baru di satu server.
- **Q3b:** jumlah play disimpan sebagai **nilai turunan** di tabel tersendiri (derived values termasuk bahan desain CF). Kolom per hari adalah kolom dinamis. Module yang sangat populer membuat semua increment menumpuk di satu row (sejenis hotspot); bila itu terjadi, counter bisa dipecah ke beberapa row (mis. `course#module#0` sampai `#7`) lalu dijumlahkan saat dibaca *(tambahan)*.
- **Q1 dan Q2:** satu row. M:N (instructor, prerequisite, enrollment) di-denormalisasi menjadi kolom bernama ID entity lain beserta nilainya.
- **Volume:** WatchEvent sangat besar dan write-intensive; inilah alasan utama column-family dipilih (performa tulis tinggi, banyak server).

5.6 Jawaban Soal 2c: Desain Graf

Vertex: Person (label `Instructor` dan/atau `Learner`; overlapping cukup dengan **dua label** pada vertex yang sama *(tambahan)*), Course, Module, Project, Certificate, Mentorship.

Edge	Dari → Ke	Properti
TEACHES	Instructor → Course	
ENROLLED_IN	Learner → Course	enroll_date, progress_pct
REQUIRES	Course → Course (prasyarat)	
HAS_MODULE	Course → Module	
EARNED	Learner → Certificate	
CERTIFIES	Certificate → Course	
IN_MENTORSHIP (×3)	Mentorship → Instructor / Learner / Project	role



- **Ternary mentor** direifikasi menjadi vertex `Mentorship` (berisi project_score), karena edge graf hanya menghubungkan dua vertex (sama dengan konversi non-binary ke binary di materi: entity artifisial).
- **Aggregation earn:** edge tidak bisa terhubung ke edge ENROLLED_IN. Aturan penuhnya (tabel 2.2) adalah menjadikan enrollment **vertex** `Enrollment` (Learner → Enrollment → Course, lalu Enrollment → Certificate), sehingga sertifikat terikat pada enrollment tertentu. Di desain ini dipakai **penyederhanaan**: ENROLLED_IN tetap edge (agar query rekomendasi Q4b sederhana), dan Certificate dihubungkan ke Learner dan Course. Penyederhanaan ini sah karena earn 1:1, tetapi aplikasi harus memastikan pasangan learner–course di sertifikat memang punya enrollment. Sebutkan pilihan dan alasannya di jawaban.
- **WatchEvent tidak dimasukkan ke graf.** Jumlahnya jutaan dan tidak dipakai untuk penelusuran relasi, sehingga lebih tepat di column-family.

Pemetaan query (contoh Cypher; path dengan panjang variabel `*1..` adalah sintaks *(tambahan)* di luar slide):

Q4a, prasyarat tidak langsung: jalur REQUIRES dengan panjang berapa pun dari course itu. Waspadai cycle bila data prasyarat salah.

```
MATCH (c:Course {course_id: $id})-[:REQUIRES*1..]->(p:Course) RETURN DISTINCT p.title
```

Q4b, rekomendasi: dari learner → course yang diikuti → learner lain yang mengikuti course itu → course lain yang mereka ikuti.

```
MATCH (me:Learner {person_id: $id})-[:ENROLLED_IN]->(c)
      <-[:ENROLLED_IN]-(o)-[:ENROLLED_IN]->(rec)
```

```
WHERE NOT (me)-[:ENROLLED_IN]->(rec)
RETURN rec.title, count) AS skor ORDER BY skor DESC
```

Reasoning yang ditulis

Q4 adalah penelusuran relasi berantai. Di relasional ini butuh join rekursif berulang; di graf cukup **mengikuti edge**.

5.7 Jawaban Soal 3: Use Case Key-Value

Posisi "lanjutkan menonton".

- **Key:** `resume:<learner_id>:<course_id>`.
- **Value:** `{module_no, position_sec, updated_at}`.
- Ditulis setiap beberapa detik selama video diputar, dan dibaca saat learner membuka course atau dashboard.

Reasoning yang ditulis

Read/write **kecil dan sangat sering**, model data sederhana, dan selalu diakses lewat key yang sama. Ini persis kriteria key-value di materi. Menyimpannya di relasional berarti update baris yang sangat sering. Alternatif: session login dengan pola TTL.

5.8 Jawaban Soal 4: DTD, XQuery, XPath

(a) DTD

```
<!ELEMENT certificates (certificate*)>
<!ELEMENT certificate (learner, course, issue_date, final_grade)>
<!ATTLIST certificate cert_no ID #REQUIRED>
<!ELEMENT learner (first_name, last_name, email)>
<!ATTLIST learner learner_id CDATA #REQUIRED>
<!ELEMENT course (title, module*)>
<!ATTLIST course course_id CDATA #REQUIRED>
<!ELEMENT module (title)>
<!ATTLIST module module_no CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT issue_date (#PCDATA)>
<!ELEMENT final_grade (#PCDATA)>
```

Catatan DTD

- `module*` (nol atau lebih), karena ER hanya menyatakan Module total terhadap has_module, bukan Course. Bila diasumsikan setiap course pasti punya module, boleh `module+`, asal asumsinya ditulis.
- name komposit menjadi `first_name` dan `last_name`.
- Elemen `title` dipakai untuk course dan module. DTD hanya mengizinkan satu deklarasi per nama elemen, dan keduanya sama-sama `#PCDATA`, jadi aman.
- Karena DTD tidak punya tipe, tanggal ditulis `YYYY-MM-DD` supaya pengurutan teks sama dengan pengurutan tanggal.

- Nilai `cert_no` bertipe `ID` harus diawali huruf (mis. `K001`).

(b) XQuery

```
for $c in doc("certificates.xml")/certificates/certificate
where $c/learner/email = "ana@kursuskita.id"
order by $c/issue_date descending
return <sertifikat no="{ $c/@cert_no }">
         { $c/course/title, $c/issue_date, $c/final_grade }
       </sertifikat>
```

(c) XPath

```
/certificates/certificate[final_grade = "A"]/@cert_no
```

5.9 Jawaban Soal 5: Model Paling Cocok per Query

Query	Model	Alasan
Q1 halaman course	**Dokumen**	Satu "kesatuan" yang dibaca utuh; module di-embed sehingga cukup satu baca
Q2 dashboard dan posisi menonton	**Key-value** (posisi) + **dokumen** (daftar enrollment)	Posisi: tulis/baca kecil sangat sering lewat key; daftar enrollment: dokumen per learner
Q3 event menonton	**Column-family**	Volume sangat besar, write-intensive, range scan menurut waktu
Q4 prasyarat dan rekomendasi	**Graf**	Penelusuran relasi berantai tanpa join
Q5 kirim ke mitra	**XML**	Format pertukaran data lintas sistem dengan skema yang bisa divalidasi
Pembayaran dan pendaftaran (di luar query)	**Relasional**	Butuh ACID dan integritas kuat

Satu aplikasi memakai beberapa model sekaligus: memilih alat yang tepat untuk tiap pekerjaan. Tidak ada DBMS terbaik untuk semua kebutuhan.

6 Penerapan ke Studi Kasus Tugas

Slide dosen meminta "pelajari case study tugas masing-masing". Bagian ini menghubungkan keputusan di tugasmu dengan tata cara di atas, sehingga kamu bisa menjelaskannya bila ditanya.

6.1 Nimonspedia (Project-0 dan Project-1)

Sumber: catatan UTS PDL bagian 2.10 dan catatan tugas Project-1. Beberapa keputusan di catatan tugas masih ditandai **belum final** (lihat poin key-value).

Keputusan di tugas	Aturan yang dipakai	Cara menjelaskannya
Project-0: 15 tabel relasional dari ERD	3.2	Pemetaan ER ke relasional standar
MongoDB 12 collection (users, stores, products, carts, orders, auctions, auction_bids, chat_rooms, chat_messages, …)	3.3	Collection mengikuti query; entity yang dibaca sebagai satu kesatuan menjadi collection
Snapshot `store: {_id, name}` di product dan order	3.3 langkah 3 (reference + salinan sebagian)	Nama toko selalu ditampilkan bersama produk/order dan jarang berubah, sehingga disalin demi menghindari join; `_id` tetap ada sebagai referensi
`chat_messages` collection sendiri, bukan embed di `chat_rooms`	3.3 faktor ukuran/pertumbuhan	Pesan terus bertambah tanpa batas, sehingga di-embed akan membuat dokumen membengkak
`auction_bids` collection sendiri	3.3 faktor update	Bid ditulis sangat sering dan mandiri
Update bersyarat (`stock >= qty`, `$inc` dengan guard) dan Mongo transaction	Sifat NoSQL: integritas dijaga aplikasi	Tanpa constraint relasional, konsistensi ditegakkan lewat operasi atomik dan transaksi
Redis untuk `feature_access` (hash `feature:{feature}:{user_id}`), `user:{uid}:unread`, cache, session	3.6	Point lookup per user + fitur; counter kecil yang sering diubah; TTL untuk data sementara

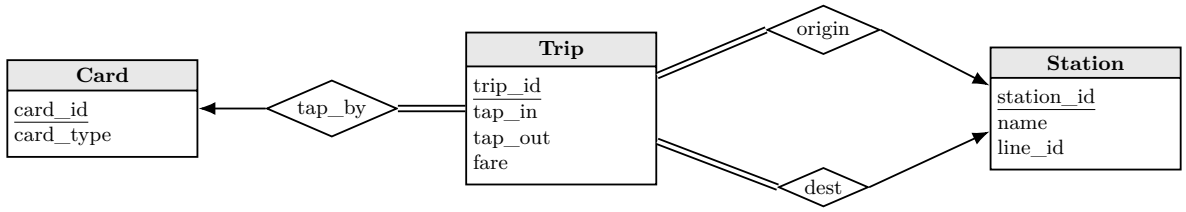
Catatan jujur dari catatan tugas

`feature_access` saat itu **masih duplikat** di Mongo dan Redis, dan belum diputuskan final apakah dicabut dari Mongo. Kalau ditanya, jawaban yang konsisten dengan materi adalah: bagian yang hanya diakses lewat key (feature flag per user, cart) layak **authoritative** di key-value, sedangkan data yang butuh query berdasarkan atribut tetap di dokumen.

Latihan mandiri (pakai tata cara bagian 3): desain ulang `orders` + `order_items` + `products` sebagai column-family untuk query "riwayat order seorang user terurut waktu" (petunjuk: row key `user_id#waktu`, seperti WatchEvent di 5.5), lalu sebagai graf untuk "rekomendasi produk dari pembeli lain" (seperti Q4b di 5.6).

6.2 TransitTap (Project-2, ClickHouse columnar)

Kasus: riwayat tap-in/tap-out kartu transportasi (Station, Card, Trip; Trip punya dua relasi ke Station: asal dan tujuan), 5 juta perjalanan. Kebutuhan utamanya analitik: perjalanan dan pendapatan per hari, stasiun tersibuk, tap per jam per koridor, pendapatan per tipe kartu, kartu aktif per minggu, ditambah satu point lookup (10 perjalanan terakhir satu kartu).



Trip total di ketiga relasi (garis ganda) dan berelasi many-to-one ke Card dan Station (panah). `origin` dan `dest` adalah dua relasi berbeda ke entity yang sama.

Model yang kamu buat:

DBMS	Model
PostgreSQL (relasional)	`stations(station_id, name, line_id)`, `cards(card_id, card_type)`, `trips(trip_id, card_id FK, origin_id FK, dest_id FK, tap_in, tap_out, fare)`
ClickHouse (columnar)	Satu **wide table** `trips` terdenormalisasi (card_type, origin_name, line_id ikut disalin), `ORDER BY (origin_id, tap_in)`, `PARTITION BY` bulan
MongoDB (dokumen)	Collection `trips` dengan card, origin, dan dest di-**embed** di tiap dokumen

Hubungannya dengan soal UTS:

- **Dua relasi ke entity yang sama** (origin dan dest ke Station) dipetakan dengan dua FK bernama sesuai role. Aturannya sama dengan relasi rekursif atau role di tabel 2.2.
- **Columnar vs column-family (penting untuk soal 2b):**

	Column-family (BigTable-like, materi)	Columnar murni (ClickHouse)
Unit penyimpanan	Row berisi column family; kolom dalam satu CF disimpan berdekatan	Tiap **kolom** tabel disimpan terpisah
Kolom	Dinamis, berbeda antar row (sparse)	Tetap dan bertipe kuat
Urutan	Terurut menurut row key	Terurut menurut `ORDER BY`
Kuat di	Lookup/tulis per row key skala besar, tulis tinggi	**Agregasi** atas banyak baris tetapi sedikit kolom
Lemah di	Query analitik lintas row tanpa desain index	Lookup satu baris di luar `ORDER BY` (di tugasmu Q6 lebih lambat dari PostgreSQL), update satu baris

- **Bila kamu memilih menjawab soal 2b dengan pendekatan columnar**, sebutkan eksplisit, lalu jelaskan: tabel lebar terdenormalisasi (menghindari join), pilihan kolom `ORDER BY` dari filter yang paling sering, dan partisi per waktu. Jelaskan juga bahwa ini cocok untuk query analitik. Untuk query lookup seperti sample (profil, satu posting), BigTable-like lebih tepat.
- **Versi BigTable-like TransitTap** (latihan): `Trip_by_Card` dengan row key `card_id#(waktu maksimum − tap_in)` untuk "10 perjalanan terakhir satu kartu". Waktu dibalik (*reverse timestamp, (tambahan)*) supaya perjalanan terbaru berada paling depan dan cukup diambil 10 row pertama; dengan `card_id#tap_in` biasa urutannya naik sehingga butuh reverse scan. Ditambah tabel turunan `Daily_Revenue` dengan row key tanggal untuk Q1. Polanya sama dengan WatchEvent dan PlayCount di 5.5.

7 Ringkasan Inti

- **Format UTS:** satu kasus (ER + aturan + query sering), lalu relasional, dokumen, column-family, graf, satu use case key-value, dan XML + query. Semuanya soal desain dengan *reasoning* dan asumsi.
- **Sebelum mendesain:** baca semua sub-soal, tandai konstruksi ER, tulis ulang tiap query (entity, atribut, filter, urutan), dan tulis asumsi.
- **Relasional:** petakan tiap konstruksi dengan tabel 2.2. Multivalued → tabel; komposit → diratakan; derived → tidak disimpan; 1:N → FK di sisi many; M:N/ternary → tabel relasi; weak → PK gabungan; ISA → superclass + subclass. Optimasi boleh, selalu dengan trade-off.
- **Dokumen:** embed bila dibaca bersama, kecil, dan 1:1/1:N; reference bila M:N, besar, atau mandiri. Jalan tengahnya reference + salinan atribut yang jarang berubah. ISA yang sering dibaca bersama → satu collection polymorphic.

- **Column-family:** satu tabel per kebutuhan baca; row key dari kriteria pencarian (gabungan `id#waktu` untuk rentang waktu, hindari hotspotting); kolom yang dibaca bersama dalam satu CF; M:N → kolom bernama ID; nilai turunan boleh disimpan.
- **Graf:** entity → vertex, relasi → edge berarah dengan properti; ternary dan aggregation → vertex perantara; ISA → label; query = mengikuti edge.
- **Key-value:** cari akses kecil, sering, lewat satu key; rancang key `<entity>:<id>:<atribut>`, value gabungan, TTL bila sementara.
- **XML:** root tunggal, atribut untuk ID, subelemen untuk isi, kemunculan `? * +`; pastikan memuat data yang difilter soal berikutnya. DTD tanpa tipe, XML Schema bertipe. XPath untuk seleksi, FLWOR untuk seleksi + bentuk hasil.
- **Pilih model:** relasional untuk ACID, key-value untuk lookup, dokumen untuk agregat fleksibel, column-family untuk volume/tulis raksasa, graf untuk relasi berantai, XML untuk pertukaran data. Satu aplikasi boleh memakai beberapa model.