Pertemuan 2 — Pemodelan Data dan Format *Columnar*

OLTP vs OLAP, *star schema*, *slowly changing dimension*, dan mengapa Parquet cepat

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Dari mana data datang: ragam sumber
2. OLTP vs OLAP: dua dunia, dua bentuk data
3. Model relasional dan *normalization*
4. *Dimensional modeling* ala Kimball
5. Tabel fakta, dimensi, dan *grain*
6. *Slowly changing dimension*
7. *Data Vault* dan *one big table*
8. Format *file*: CSV, JSON, Avro, Parquet, ORC
9. *Schema evolution*
10. Praktikum: ukur sendiri dengan DuckDB

Sasaran pertemuan ini

Anda dapat merancang *star schema* untuk satu proses bisnis, memilih tipe *slowly changing dimension* yang tepat, dan membuktikan dengan pengukuran sendiri mengapa format *columnar* lebih cocok untuk analitik.

Dari mana data datang

Ragam sumber data di bank syariah digital

Sumber	Bentuk khas
Database inti (*core banking*)	Tabel relasional (PostgreSQL, Oracle)
Aplikasi *mobile*	*Event* klik, JSON
Log server dan ATM	Teks semi-terstruktur
API mitra (QRIS, *payment gateway*)	JSON lewat HTTP
File dari regulator atau mitra	CSV, Excel, XML
Sensor dan perangkat (agen, EDC)	*Message* kecil berfrekuensi tinggi

Tiga pertanyaan untuk setiap sumber

1. **Bentuk**: terstruktur, semi-terstruktur, atau tidak terstruktur?
2. **Ritme**: berubah tiap detik, tiap hari, atau sekali sebulan?
3. **Pemilik**: siapa yang boleh mengubah skemanya tanpa memberi tahu kita?

Jebakan klasik

Tim aplikasi mengganti nama kolom, dan *pipeline* analitik diam-diam rusak. Masalah ini dibahas lagi saat *data contract* (P15).

OLTP vs OLAP

Dua dunia, dua bentuk data

OLTP — *Online Transaction Processing*

- Melayani aplikasi: transfer, setor, buka rekening
- Banyak transaksi kecil, tiap transaksi menyentuh sedikit baris
- Butuh *latency* milidetik dan ACID
- *Schema* ternormalisasi agar tidak ada data ganda yang bisa bertentangan
- Contoh *query*: “ambil saldo rekening 7712”

OLAP — *Online Analytical Processing*

- Melayani analis, *dashboard*, laporan, model
- Sedikit *query* besar, tiap *query* memindai jutaan baris tetapi sedikit kolom
- Toleran terhadap *latency* detik sampai menit
- *Schema* didenormalisasi agar mudah dan cepat di-*query*
- Contoh *query*: “total pembiayaan murabahah per cabang per bulan, 3 tahun”

Ringkasan

Menjalankan *query* OLAP berat langsung di *database* OLTP produksi adalah cara cepat memperlambat aplikasi nasabah. Karena itu data disalin ke tempat terpisah untuk analitik.

Pola akses menentukan bentuk *storage*

OLTP: ambil satu baris utuh

id	rekening	tanggal	nominal	kanal
id	rekening	tanggal	nominal	kanal
id	rekening	tanggal	nominal	kanal

Storage ***row-based*** menaruh semua kolom satu baris berdampingan: sekali baca, satu transaksi utuh didapat.

OLAP: ambil satu kolom, semua baris

id	rekening	tanggal	nominal	kanal
id	rekening	tanggal	nominal	kanal
id	rekening	tanggal	nominal	kanal
id	rekening	tanggal	nominal	kanal

Storage ***columnar*** menaruh semua nilai satu kolom berdampingan: `SUM(nominal)` hanya membaca kolom `nominal`.

Model relasional dan *normalization*

Normalization: satu fakta di satu tempat

- **1NF**: tiap sel berisi satu nilai atomik, tidak ada kelompok berulang.
- **2NF**: setiap kolom bukan *key* bergantung pada **seluruh** *key* primer, bukan sebagian.
- **3NF**: tidak ada ketergantungan transitif; kolom bukan *key* tidak bergantung pada kolom bukan *key* lain.

Contoh pelanggaran 3NF

Tabel `transaksi(id, rekening, nama_cabang, alamat_cabang, ...)`: alamat cabang bergantung pada nama cabang, bukan pada transaksi. Bila cabang pindah, ribuan baris harus diubah, dan satu yang terlewat menimbulkan data bertentangan.

Normalization vs *denormalization*

	Normal	**Denormal**
Tulis	cepat, aman	lambat, rawan
Baca analitik	banyak *join*	sedikit *join*
Ruang	hemat	boros
Cocok untuk	OLTP	OLAP

Ringkasan

Normalization melindungi **kebenaran saat menulis**; *denormalization* mempercepat **membaca**. Analitik memilih yang kedua secara sadar.

Dimensional modeling

Empat langkah Kimball

1. **Pilih proses bisnis**: kejadian yang diukur, misalnya “transaksi rekening” atau “pencairan pembiayaan”.
2. **Tetapkan *grain***: apa arti **satu baris** tabel fakta? Misalnya “satu transaksi” atau “saldo satu rekening pada akhir hari”.
3. **Pilih dimensi**: konteks yang menjawab siapa, apa, kapan, di mana, lewat apa.
4. **Pilih fakta**: angka yang diukur pada *grain* itu (nominal, jumlah, margin).

Kesalahan paling mahal

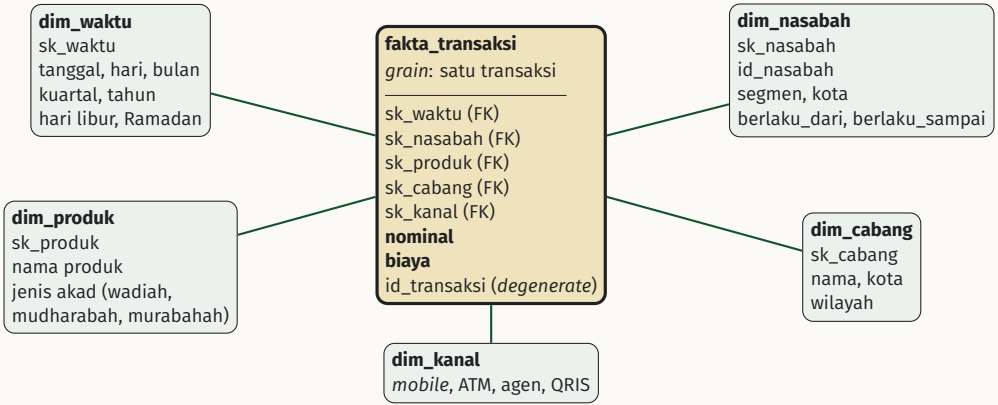
Grain yang tidak jelas. Mencampur baris “per transaksi” dan “per hari” di satu tabel membuat setiap `SUM` salah tanpa peringatan.

Bahasa bisnis

Nama tabel dan kolom memakai istilah yang dipahami pengguna bisnis: `dim_akad`, `fakta_pencairan`, bukan `tbl_x9`.

Sumber: Kimball dan Ross, *The Data Warehouse Toolkit*, ed. 3, Bab 1–3.

Star schema: transaksi rekening



Satu tabel fakta yang panjang dan sempit di tengah, dikelilingi dimensi yang pendek dan lebar. *Query* analitik cukup satu *layer join*.

Tiga jenis tabel fakta

Jenis	Satu baris berarti	Contoh di bank
Transaction	satu kejadian	satu transfer, satu penarikan ATM
Periodic snapshot	keadaan pada akhir periode	saldo tiap rekening pada akhir hari; outstanding pembiayaan akhir bulan
Accumulating snapshot	satu proses dengan beberapa tahap	satu pengajuan pembiayaan: diajukan, dianalisis, disetujui, dicairkan (satu kolom tanggal per tahap)

Fakta aditif dan tidak

Aditif: boleh dijumlah di semua dimensi (nominal transaksi). **Semi-aditif**: tidak boleh dijumlah lintas waktu (saldo: jumlah saldo 30 hari bukan saldo). **Non-aditif**: rasio dan persentase; simpan pembilang dan penyebutnya.

Ringkasan

Saldo adalah *periodic snapshot* yang semi-aditif. Menjumlahkan saldo harian sebulan untuk “total saldo” adalah kesalahan analitik yang sering terjadi.

Dimensi, *surrogate key*, dan *star* vs *snowflake*

Surrogate key

Key buatan *data warehouse* (`sk_nasabah = 1, 2, 3, ...`), terpisah dari *key* sistem sumber (`id_nasabah`). Alasannya:

- satu nasabah bisa punya beberapa versi baris (lihat SCD tipe 2),
- sumber bisa berganti sistem dan *key*-nya berubah,
- menggabungkan beberapa sumber dengan *key* yang bertabrakan.

Star vs *snowflake*

Snowflake schema menormalisasi dimensi lagi (kota → provinsi → wilayah di tabel terpisah). Hemat ruang, tetapi *query* lebih banyak *join* dan lebih sulit dipahami pengguna bisnis.

Praktik industri

Pakai *star schema* sebagai bawaan. Ruang *storage* murah; waktu analis dan *query* yang salah mahal.

Slowly changing dimension

Nasabah pindah kota: apa yang terjadi pada laporan lama?

Nasabah 7712 tinggal di Bogor sampai Juni, lalu pindah ke Bandung. Transaksi Maret dihitung untuk kota mana?

Tipe	Perlakuan	Akibat
0	Tidak pernah diubah	Cocok untuk atribut tetap (tanggal lahir)
1	Timpa nilai lama	Sejarah hilang: transaksi Maret ikut pindah ke Bandung
2	Tambah baris baru dengan rentang berlaku	Sejarah benar: Maret tetap Bogor, Juli ke atas Bandung
3	Simpan nilai sebelumnya di kolom tambahan	Hanya satu langkah sejarah

Ringkasan

Ini keputusan bisnis: apakah laporan harus mencerminkan keadaan **saat kejadian** atau keadaan **sekarang**? Keduanya sah untuk tujuan berbeda.

SCD tipe 2 dalam tabel

sk_nasabah	id_nasabah	kota	berlaku_dari	berlaku_sampai	aktif
101	7712	Bogor	2024-01-15	2026-06-30	0
245	7712	Bandung	2026-07-01	9999-12-31	1

- Fakta transaksi Maret 2026 menyimpan `sk_nasabah = 101` → Bogor.
- Fakta transaksi Agustus 2026 menyimpan `sk_nasabah = 245` → Bandung.
- "Keadaan sekarang" cukup difilter `aktif = 1`.
- Tanggal `9999-12-31` adalah konvensi untuk "belum berakhir".

Di alat modern

dbt menyediakan `snapshot` untuk SCD tipe 2 secara otomatis (P6); tabel *lakehouse* mendukung `MERGE` untuk menutup baris lama dan menambah baris baru dalam satu transaksi (P4).

Data Vault dan *one big table*

Dua pendekatan lain yang akan Anda temui

Data Vault

Memisahkan ***hub*** (*key* bisnis), ***link*** (relasi), dan ***satellite*** (atribut beserta sejarahnya).

- \+ tahan perubahan sumber, mudah diaudit, cocok untuk banyak sumber
- – banyak tabel; tetap butuh *layer star schema* di atasnya untuk pengguna bisnis

One big table (OBT)

Satu tabel sangat lebar berisi fakta dan seluruh atribut dimensinya.

- \+ tanpa *join*, cepat di mesin *columnar*, mudah untuk *dashboard*
- – data ganda besar, perubahan dimensi sulit, rawan definisi ganda

Praktik industri

Banyak tim memakai kombinasi: *star schema* sebagai sumber kebenaran di *layer silver/gold*, lalu OBT turunan untuk *dashboard* tertentu. Kita membahas *layer* ini di P4.

Format *file*

Lima format yang paling sering dipakai

Format	***Layout***	***Schema***	***Compression***	**Kegunaan khas**
CSV	baris, teks	tidak ada	lemah	pertukaran *file* sederhana, ekspor manual
JSON / JSONL	baris, teks	tersirat	lemah	API, *event*, log
Avro	baris, biner	wajib, ikut *file*	baik	*message* Kafka, *ingestion*, *schema evolution*
Parquet	kolom, biner	wajib	sangat baik	analitik, *data lake*, *lakehouse*
ORC	kolom, biner	wajib	sangat baik	ekosistem Hive dan Hadoop

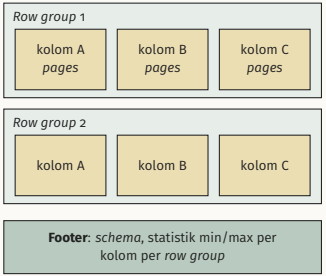
Masalah CSV yang sering terjadi

Tanda pemisah di dalam teks, format tanggal ganda (03/04: Maret atau April?), angka desimal koma vs titik, semuanya bertipe teks.

Aturan praktis

Format baris untuk **menulis dan mengirim** satu catatan (Avro, JSON); format kolom untuk **menyimpan dan menganalisis** banyak catatan (Parquet, ORC).

Di dalam *file* Parquet



Sumber: Spesifikasi Apache Parquet (parquet.apache.org).

- *File* dibagi menjadi ***row group***; di dalamnya tiap kolom disimpan terpisah sebagai ***column chunk***.
- ***Encoding***: *dictionary* (nilai berulang seperti nama kanal), *run-length* (nilai berurutan sama), *delta* (tanggal berurutan).
- ***Compression*** per kolom: Snappy, Zstd, Gzip.
- ***Footer*** menyimpan statistik, sehingga *query engine* bisa melewati *row group* yang pasti tidak relevan.

Tiga trik yang membuat *query* analitik cepat

Column pruning

`SELECT SUM(nominal)` hanya membaca kolom `nominal`. Tabel 40 kolom, yang dibaca 1 kolom.

Predicate pushdown

`WHERE tanggal >= '2026-10-01'`: *row group* yang `max(tanggal)`-nya sebelum Oktober dilewati tanpa dibuka.

Compression efektif

Nilai sejenis bersebelahan mudah dimampatkan, sehingga byte yang dibaca dari disk dan jaringan jauh lebih sedikit.

Hitungan kasar (ilustrasi): tabel 1 miliar baris $\times$ 40 kolom $\times$ rata-rata 8 byte $\approx$ 320 GB mentah. *Query* yang butuh 2 kolom membaca $\approx$ 16 GB sebelum *compression*, atau 5% dari tabel. Dengan *predicate pushdown* pada satu bulan dari tiga tahun data, sisanya bisa turun lagi sekitar 36 kali.

Ringkasan

Hari ini kita **mengukur** bahwa ini cepat. Alasan *hardware*-nya (*cache line*, SIMD) dibedah di Sistem Terdistribusi P10.

Memilih format: *when to use*

Kebutuhan	Pilihan	Alasan
Menerima *file* dari mitra	CSV atau JSON, lalu segera ubah	Mitra jarang mampu menghasilkan Parquet; simpan aslinya di *bronze*
Event message di Kafka	Avro atau Protobuf	Biner kecil, *schema* eksplisit, *schema registry* (P11)
Tabel analitik di *lake*	Parquet (+ Iceberg/Delta)	*Columnar*, statistik, didukung hampir semua mesin
Pertukaran di memori antar-proses	Apache Arrow	Format *columnar* di memori, *zero-copy* (P5)
File kecil sekali pakai	CSV	Tidak perlu rekayasa berlebihan

Best practice industri

Terima apa adanya, simpan aslinya, lalu ubah ke Parquet sedini mungkin. Analitik berat di atas CSV atau JSON mentah adalah pemborosan.

Schema evolution

Schema pasti berubah, jadi rancang agar perubahannya tidak merusak

Perubahan	Umumnya
Tambah kolom opsional	aman
Hapus kolom opsional	aman bagi pembaca baru
Ganti nama kolom	merusak
Ganti tipe (`int` → `string`)	merusak
Perlebar tipe (`int` → `long`)	biasanya aman

Arah kompatibilitas

Backward: pembaca versi baru dapat membaca data lama.
Forward: pembaca versi lama dapat membaca data baru.
Full: keduanya.

Avro dan Protobuf memeriksa aturan ini secara eksplisit; Iceberg dan Delta melacak kolom dengan ID sehingga ganti nama tidak merusak tabel (P4).

Ringkasan

Perlakukan *schema* seperti API publik: setiap perubahan diberi versi dan diumumkan lebih dulu.

Praktikum

Praktikum 1: *star schema* dalam SQL

```sql
CREATE TABLE dim_nasabah (
  sk_nasabah     INTEGER PRIMARY KEY,
  id_nasabah     VARCHAR NOT NULL,     -- kunci sistem sumber
  segmen         VARCHAR, kota VARCHAR,
  berlaku_dari   DATE NOT NULL,
  berlaku_sampai DATE NOT NULL DEFAULT DATE '9999-12-31',
  aktif          BOOLEAN NOT NULL);

CREATE TABLE fakta_transaksi (
  id_transaksi VARCHAR,                -- degenerate dimension
  sk_waktu     INTEGER, sk_nasabah INTEGER, sk_produk INTEGER,
  sk_cabang    INTEGER, sk_kanal  INTEGER,
  nominal      DECIMAL(18,2), biaya DECIMAL(18,2));

-- Total nominal per kota pada saat transaksi terjadi (SCD tipe 2)
SELECT n.kota, SUM(f.nominal) AS total
FROM fakta_transaksi f JOIN dim_nasabah n USING (sk_nasabah)
GROUP BY n.kota ORDER BY total DESC;
```

Praktikum 2: CSV vs JSON vs Parquet dengan DuckDB

```
import duckdb, os, time
con = duckdb.connect()
con.sql("""CREATE TABLE trx AS
  SELECT * FROM 'transaksi_sintetis.csv'""")
con.sql("COPY trx TO 'trx.json' (FORMAT JSON)")
con.sql("COPY trx TO 'trx.parquet' (FORMAT PARQUET)")

q = """SELECT kanal, SUM(nominal)
       FROM '{f}' GROUP BY kanal"""
for f in ["transaksi_sintetis.csv",
          "trx.json", "trx.parquet"]:
    t = time.perf_counter()
    con.sql(q.format(f=f)).fetchall()
    print(f, os.path.getsize(f) // 2**20, "MB",
          round(time.perf_counter() - t, 3), "s")
```

Langkah

1. Bangkitkan data transaksi sintetis $\geq$ 10 juta baris (skrip dibagikan).
2. Jalankan tiap *query* 5 kali; laporkan median dari kelima jalannya.
3. Isi tabel: ukuran *file*, waktu *query*, rasio terhadap CSV.
4. Ulangi dengan `WHERE` pada tanggal; jelaskan bedanya.

Melihat isi Parquet dari dalam

```python
import pyarrow.parquet as pq
md = pq.ParquetFile("trx.parquet").metadata
print(md.num_row_groups, md.num_rows)
rg = md.row_group(0)
for i in range(rg.num_columns):
    c = rg.column(i)
    print(c.path_in_schema, c.compression,
          c.total_compressed_size,
          c.statistics.min, c.statistics.max)
```

Yang perlu Anda amati

- Kolom mana yang paling kecil setelah *compression*, dan mengapa?
- Apakah statistik min/max kolom `tanggal` tumpang tindih antar *row group*? Apa akibatnya bagi *predicate pushdown*?
- Coba urutkan data berdasarkan tanggal sebelum ditulis, lalu bandingkan.

Penutup

Rangkuman

- OLTP dan OLAP punya pola akses berbeda, sehingga bentuk datanya juga berbeda.
- *Star schema*: fakta di tengah, dimensi di sekeliling, *grain* jelas.
- SCD tipe 2 menjaga sejarah dengan baris baru dan rentang berlaku.
- Format baris untuk menulis dan mengirim; format kolom untuk analitik.
- Parquet cepat karena *column pruning*, *predicate pushdown*, dan *compression*.
- *Schema* adalah kontrak; ubah dengan aturan kompatibilitas.

Tugas untuk pertemuan berikutnya

1. Rancang *star schema* untuk proses **pencairan pembiayaan** bank syariah (murabahah, musyarakah, ijarah): tetapkan *grain*, dimensi, fakta, dan tipe SCD tiap atribut nasabah. Sertakan diagram.
2. Laporan praktikum 2: tabel ukuran dan waktu *query* CSV, JSON, Parquet (median 5 kali), plus penjelasan hasil *predicate pushdown*.
3. Baca Kimball dan Ross Bab 1–3, serta Kleppmann Bab 3 bagian *column-oriented storage*.

Pertemuan depan

Di mana semua *file* itu disimpan bila jumlahnya terabyte: HDFS vs *object storage*, *replication* vs *erasure coding*, dan *storage tiering hot*, *warm*, *cold*, *archive* beserta hitungan *cost*-nya.

Terima kasih

Pertanyaan?