ACBP: A SQL-Native Categorical-Boolean DSL for Deterministic Decision Spaces

From Event Feeds to Sub-Second Analytics via Bitmasks, Categories, and Materialized Spaces

Muteb Hail S. Al Anazi
Independent Researcher (DotK), Riyadh, Saudi Arabia
dotkboy@outlook.com
ORCID: 0009-0004-2668-3564

2025-08-18

Abstract

We present ACBP (Al Anazi Categorical-Boolean Paradigm), a minimal DSL that turns domain rules into SQL-native artifacts: views, functions, and materialized decision spaces. The core idea is compact: represent system state as an ordered bitmask over boolean flags, represent slicers as finite categories, and compile constraints that prune invalid (flag, category) combinations. The result is a join-friendly "valid masks" view and a pruned decision space that can be indexed, materialized, and refreshed. We provide two public v0 models (clinic_visit, inpatient_admission), a SQL emitter, and database utilities for materialization, refresh, and benchmarks. We outline a push-driven path (LISTEN/NOTIFY) for near-real-time dashboards and evaluate build/refresh costs and query latency on synthetic data. We contrast ACBP with decision tables (DMN), policy-as-code (OPA/Rego), and SAT-style configuration.

Contents

1 Introduction

Operational rules are often buried in application code or external engines. Analytics and planning systems, however, benefit from SQL-first artifacts that can be indexed, materialized, and joined. **ACBP** is a small DSL that compiles rules to PostgreSQL: validating functions, `<model>_valid_masks(_mat)` for fast joins, and `<model>_decision_space(_mat)` for exploration.

What is new here.

(1) A bits+categories DSL with a clear semantic predicate and a compiler that emits pure SQL for Postgres.
(2) A practical toolbox: materialization, present-only decision space, and benchmark helpers in SQL/PL/pgSQL.
(3) Two end-to-end models (clinic, inpatient) that readers can compile, apply, and explore.

2 The ACBP Equation

Let $F \in \{0,1\}^B$ be an ordered bitmask and $c \in C = \prod_i C_i$ a tuple of finite categories. Given rule set R, define:

$$\mathrm{ACBP}(F,c) := \bigwedge_{r\in R} r(F,c)$$

Valid masks and decision space:

$$M = \{\, F \mid \exists c \in C : \mathrm{ACBP}(F,c)\,\}, \quad D = \{\,(F,c) \in \{0,1\}^B \times C \mid \mathrm{ACBP}(F,c)\,\}.$$

Present-only restricts categories to those observed in source data:

$$D_{\text{present}} = D \cap (\{0,1\}^B \times C').$$

Compiler contract.
The generated SQL satisfies:
- `acbp_is_valid__m(mask)` $\Leftrightarrow \mathrm{ACBP}(F,\cdot)$ for bit-only rules.
- `acbp_is_valid__m_cats(mask,<cats...>)` $\Leftrightarrow \mathrm{ACBP}(F,c)$.
- `<model>_decision_space` rows satisfy ACBP.
- `<model>_valid_masks` enumerates M.
- `acbp_explain_rules__m(...)` returns failing bit-rule checks (optional cat-rule explainer available).

Guardrail: if effective bit width B_{eff} exceeds `enumeration_limit_bits`, the compiler skips pre-enumeration and emits validators only.

3 DSL v0 Overview (shipped)

- **Flags**: ordered booleans, packed into bigint masks.
- **Categories**: finite string-valued dimensions.
- **Constraints** (subset): IMPLIES, MUTEX, EQUIV, ONEOF, FORBID_WHEN, FORBID_IF_SQL.
- **Outputs**: SQL functions (acbp_is_valid__*), views (*_valid_masks, *_decision_space), and optional explainers.

The compiler preserves all raw facts; pruning occurs in the decision-space layer (and its materialized variants).

4 Implementation (PostgreSQL)

4.1 SQL Emitter

acbp_tester reads a v0 JSON model and emits PostgreSQL SQL. It constructs:

- "<model>_valid_masks" via generate_series(0, 2^B-1) with a **bit-only** predicate.
- "<model>_categories" from unnest(ARRAY[...]) per category.
- "<model>_decision_space" as valid_masks CROSS JOIN categories with a **bit+category** predicate.
- Validators: acbp_is_valid__<model>(mask) and (when cat rules exist) acbp_is_valid__<model>_cats(mas <cats...>).
- Explainers: acbp_explain_rules__<model>(mask) (bit-only) and optional acbp_explain__<model>(mask, <cats...>) (cat-aware).
- Helper: acbp_popcount(bigint) for ONEOF.

The compiler emits pure SQL (views + functions) and does not require stored state beyond standard catalogs.

4.2 Database utilities

acbp.sh installs idempotent helpers:

- acbp_materialize(model [, force]) makes <model>_valid_masks_mat and <model>_decision_space_m and ensures a composite unique index (mask, <categories...>) matching the decision-space order.
- acbp_refresh(model) refreshes both mats concurrently.
- Present-only: acbp_materialize_present(model, data_table), acbp_refresh_present(model), and acbp_bench_full_join_present(...).
- Benchmarks: acbp_bench_valid_join, acbp_bench_valid_func, acbp_bench_full_join.

Ports. The container maps host 5434 -> 5432 inside Postgres.

5 Case Study A: Clinic Visits (v0)

Flags (5): booked, checked_in, seen_by_doctor, canceled, rescheduled.

Categories (9):
appt_type (NewPatient, FollowUp, Urgent, Procedure, Teleconsult)
site (Main, Annex, Downtown)

`age_group` (Peds, Adult, Geriatric)
`department` (General, Cardiology, Orthopedics, Imaging, Pediatrics)
`provider_role` (Attending, Resident, NP/PA)
`modality` (InPerson, Virtual)
`visit_hour` (08:00, 09:00, 10:00, 11:00, 14:00)
`weekday` (Mon–Fri)
`insurance` (SelfPay, Private, Government)

Constraints (subset).
- `IMPLIES(checked_in -> booked)`; `IMPLIES(seen_by_doctor -> checked_in)`; `IMPLIES(rescheduled -> booked)`.
- `MUTEX(canceled, checked_in)`, `MUTEX(canceled, rescheduled)`, `MUTEX(rescheduled, checked_in)`, `MUTEX(rescheduled, seen_by_doctor)`.
- `FORBID_IF_SQL(booked, modality='Virtual' AND department IN ('Imaging','Orthopedics'))`.
- `FORBID_IF_SQL(booked, appt_type='Teleconsult' AND modality <> 'Virtual')`.
- `FORBID_IF_SQL(booked, modality='Virtual' AND appt_type NOT IN ('FollowUp','Teleconsult'))`.
- `FORBID_IF_SQL(seen_by_doctor, visit_hour NOT IN ('09:00','10:00','11:00','14:00'))`.
- `FORBID_IF_SQL(booked, site='Annex' AND department IN ('Cardiology'[, 'Neurology']))`.
- `FORBID_IF_SQL(booked, department='Pediatrics' AND age_group <> 'Peds')`.

Bit enumeration. `enumeration_limit_bits = 22` (5 bits, so pre-enumeration proceeds).

6 Case Study B: Inpatient Admission (v0)

Flags (6): `booked`, `checked_in`, `in_icu`, `discharged`, `expired`, `transferred`.
Categories (8):
`admission_type` (Elective, Emergency, Transfer)
`site` (Main, Annex)
`age_group` (Adult, Peds)
`ward` (Medical, Surgical, ICU, StepDown)
`payer` (SelfPay, Private, Public)
`arrival_source` (ED, Clinic, Transfer, Direct)
`admit_hour` (00:00, 04:00, 08:00, 12:00, 16:00, 20:00)
`weekday` (MonSun)

Constraints (subset).
- `IMPLIES(checked_in -> booked)`, `IMPLIES(discharged -> checked_in)`.
- `MUTEX(discharged, expired)`, `MUTEX(discharged, transferred)`, `MUTEX(expired, transferred)`.
- `FORBID_WHEN(booked, admission_type='Emergency')`.
- `FORBID_WHEN(in_icu, ward IN {'Medical','Surgical','StepDown'})`.
- `FORBID_WHEN(discharged, arrival_source='Transfer')`.

Bit enumeration. `enumeration_limit_bits = 22` (6 bits, so pre-enumeration proceeds).

7 Reproducibility (artifact)

The artifact is self-contained: Postgres in Docker, compiler, SQL helpers, and a web UI.

7.1 Bring up Postgres

```
# from repo root
./acbp.sh up
# Postgres listens on host port 5434 (container 5432)
```

7.2 Compile and apply the models

```
./acbp.sh compile-apply models/clinic_visit.v0.json
./acbp.sh compile-apply models/inpatient_admission.v0.json
```

7.3 Install DB utilities and materialize

```
./acbp.sh reinstall-db-utils
./acbp.sh materialize clinic_visit
./acbp.sh materialize inpatient_admission
```

(Optional) present-only decision space once you have data in `<model>_data`:

```
./acbp.sh materialize-present clinic_visit clinic_visit_data
```

Note. acbp_materialize_present(model, data_table) creates _present_mat by intersecting the decision space with distinct (mask, categories) actually present in your data and then runs ANALYZE so plans are calibrated.

7.4 Seed synthetic data (one-table sketch)

```
# minimal table with mask + a few category columns shared with decision_space
./acbp.sh psql-c "CREATE TABLE IF NOT EXISTS clinic_visit_data(
    mask bigint NOT NULL,
    site text, department text, modality text, weekday text, visit_hour text
);"

# insert synthetic rows (tweak as needed)
./acbp.sh psql-c "INSERT INTO
↪  clinic_visit_data(mask,site,department,modality,weekday,visit_hour)
  SELECT (random()*31)::bigint, s, d, m, w, h
  FROM unnest(array['Main','Annex']) s,

↪  unnest(array['General','Cardiology','Imaging','Orthopedics','Pediatrics']) d,
       unnest(array['InPerson','Virtual']) m,
       unnest(array['Mon','Tue','Wed','Thu','Fri']) w,
       unnest(array['09:00','10:00','11:00','14:00']) h
  LIMIT 5000;"
```

```
./acbp.sh psql-c "SELECT
↪  acbp_create_matching_index('clinic_visit','clinic_visit_data');"
./acbp.sh refresh clinic_visit
```

7.5 Synthetic dataset (provenance)

We release synthetic CSVs for both models (50,000 rows each) derived from the generator in "make_data.py". Rows are generated with a fixed seed and include demographics ("sex", "language", "city") alongside the ACBP mask and categories.

Generation (example):

```
# Clinic (50k rows, seed=42)
python make_data.py clinic_visit --rows 50000 --seed 42
# Inpatient (50k rows, seed=43)
python make_data.py inpatient_admission --rows 50000 --seed 43
```

This produces two-part CSVs per model:

```
dataset/clinic_visit_data_part1.csv
dataset/clinic_visit_data_part2.csv
dataset/inpatient_admission_data_part1.csv
dataset/inpatient_admission_data_part2.csv
```

Import into Postgres:

```
# Tables (schema matches <model>_decision_space shape + demographics)
./acbp.sh psql-c "
CREATE TABLE IF NOT EXISTS clinic_visit_data(
  mask bigint NOT NULL,
  patient_mrn text, sex text, language text, city text,
  appt_type text, site text, age_group text, department text, provider_role text,
  modality text, visit_hour text, weekday text, insurance text
);
CREATE TABLE IF NOT EXISTS inpatient_admission_data(
  mask bigint NOT NULL,
  patient_mrn text, sex text, language text, city text,
  admission_type text, site text, age_group text, ward text, payer text,
  arrival_source text, admit_hour text, weekday text
);
"

# COPY (two parts each)
docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY clinic_visit_data
↪  FROM STDIN WITH CSV HEADER" < dataset/clinic_visit_data_part1.csv
docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY clinic_visit_data
↪  FROM STDIN WITH CSV HEADER" < dataset/clinic_visit_data_part2.csv

docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY
↪  inpatient_admission_data FROM STDIN WITH CSV HEADER" <
↪  dataset/inpatient_admission_data_part1.csv
```

```
docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY
↪  inpatient_admission_data FROM STDIN WITH CSV HEADER" <
↪  dataset/inpatient_admission_data_part2.csv

# Stats + indexing + present-only
./acbp.sh vacuum clinic_visit_data
./acbp.sh vacuum inpatient_admission_data
./acbp.sh psql-c "SELECT
↪  acbp_create_matching_index('clinic_visit','clinic_visit_data'); SELECT
↪  acbp_materialize_present('clinic_visit','clinic_visit_data');"
./acbp.sh psql-c "SELECT
↪  acbp_create_matching_index('inpatient_admission','inpatient_admission_data');
↪  SELECT
↪  acbp_materialize_present('inpatient_admission','inpatient_admission_data');"
```

Present-only sizes observed in our run: clinic=30,416, inpatient=24,346 rows.

8 Evaluation & Results (outline)

We sweep bits and category breadth on synthetic data. Metrics:

- **Build** time and size for `*_valid_masks(_mat)` and `*_decision_space(_mat)`.
- **Refresh** time for concurrent matviews.
- **Query latency** for top-N groupings (full vs present-only).
- **Validator parity**: counts via `JOIN` vs `acbp_is_valid__*`.

We also measure the impact of `acbp_create_matching_index` on the data table.

8.1 8.1 Results (synthetic; 50k rows per model)

Run timestamps (UTC): clinic_visit=20250817T175223Z; inpatient_admission=20250817T175243Z

8.1.1 Clinic Visit

Complexity & sanity (compiler)

```
Model: clinic_visit
  B (flags):       5
  B_eff (reduced): 5
  n_eff (cats):    101250
  Complexity:      2^5 * 101250
  Valid masks enumerated (bit-only): 7 / 32
  First few: [0, 1, 3, 7, 8, 9, 17]
=== Sanity estimates (uniform, independent categories; FORBID_WHEN only) ===
  Flag prevalence among valid masks: booked=83.3%, checked_in=33.3%,
↪  seen_by_doctor=16.7%, canceled=16.7%, rescheduled=16.7%
  Theoretical max rows (bit-only):   607,500
  Est. remaining rows (cat rules):   478,125  (~78.7% of max)
  Est. pruned rows (cat rules):      129,375
```

```
  note: Applied FORBID_WHEN estimates: booked@50.00%; booked@33.33%. Excluded 4
↪  FORBID_IF_SQL rule(s) from estimate.
=== Actuals (latest summary) ===
  Decision rows: 295,650  (~48.7% of theoretical; pruned 311,850)
  Present-only rows: 30,416
  Data rows: 50,000
```

Simulated dashboard performance

scenario	queries	total ms	avg per query
cold	9	879.837	97.760
warm	9	825.539	91.727

Artifacts:
- papers/results/20250817T175223Z/clinic_visit/summary.csv
- papers/results/20250817T175223Z/clinic_visit/valid_counts.csv
- papers/results/20250817T175223Z/clinic_visit/top_groups_full.csv, plan: papers/results/20250817T175223Z/clinic_visit/plan_top_groups_full.txt
- papers/results/20250817T175223Z/clinic_visit/top_groups_present.csv, plan: papers/results/20250817T175223Z/clinic_visit/plan_top_groups_present.txt
- papers/results/20250817T175223Z/clinic_visit/dashboard_perf.csv (cold/warm timings)
- papers/results/20250817T175223Z/clinic_visit/compiler_sanity.txt (complexity & sanity output)
- papers/results/20250817T175223Z/clinic_visit/kpi_by_age_group.csv
- papers/results/20250817T175223Z/clinic_visit/kpi_by_appt_type.csv
- papers/results/20250817T175223Z/clinic_visit/kpi_by_appt_type_site.csv
- papers/results/20250817T175223Z/clinic_visit/kpi_by_department.csv
- papers/results/20250817T175223Z/clinic_visit/kpi_by_provider_role.csv
- papers/results/20250817T175223Z/clinic_visit/kpi_by_site.csv

8.1.2 Inpatient Admission

Complexity & sanity (compiler)

```
Model: inpatient_admission
  B (flags):       6
  B_eff (reduced): 6
  n_eff (cats):    24192
  Complexity:      2^6 * 24192
  Valid masks enumerated (bit-only): 20 / 64
  First few: [0, 1, 3, 4, 5, 7, 11, 15, 16, 19]
=== Sanity estimates (uniform, independent categories; FORBID_WHEN only) ===
  Flag prevalence among valid masks: booked=90.0%, checked_in=80.0%,
↪  in_icu=40.0%, discharged=20.0%, expired=20.0%, transferred=20.0%
  Theoretical max rows (bit-only):   241,920
  Est. remaining rows (cat rules):   117,279  (~48.5% of max)
  Est. pruned rows (cat rules):      124,641
```

```
  note: Applied FORBID_WHEN estimates: booked@33.33%; in_icu@25.00%;
↪  in_icu@25.00%; in_icu@25.00%; discharged@25.00%.
=== Actuals (latest summary) ===
  Decision rows: 126,000  (~52.1% of theoretical; pruned 115,920)
  Present-only rows: 24,346
  Data rows: 50,000
```

Simulated dashboard performance

scenario	queries	total ms	avg per query
cold	9	609.826	67.758
warm	9	622.174	69.130

Artifacts: - `papers/results/20250817T175243Z/inpatient_admission/summary.csv`
- `papers/results/20250817T175243Z/inpatient_admission/valid_counts.csv`
- `papers/results/20250817T175243Z/inpatient_admission/top_groups_full.csv`, plan: `papers/results/20250817T175243Z/inpatient_admission/plan_top_groups_full.txt`
- `papers/results/20250817T175243Z/inpatient_admission/top_groups_present.csv`, plan: `papers/results/20250817T175243Z/inpatient_admission/plan_top_groups_present.txt`
- `papers/results/20250817T175243Z/inpatient_admission/dashboard_perf.csv` (cold/warm timings)
- `papers/results/20250817T175243Z/inpatient_admission/compiler_sanity.txt` (complexity & sanity output)
- `papers/results/20250817T175243Z/inpatient_admission/kpi_by_admission_type.csv`
- `papers/results/20250817T175243Z/inpatient_admission/kpi_by_admission_type_site.csv`
- `papers/results/20250817T175243Z/inpatient_admission/kpi_by_age_group.csv`
- `papers/results/20250817T175243Z/inpatient_admission/kpi_by_payer.csv`
- `papers/results/20250817T175243Z/inpatient_admission/kpi_by_site.csv`
- `papers/results/20250817T175243Z/inpatient_admission/kpi_by_ward.csv`

8.1.3 8.1.1 Compact summary (space pruning)

model	naïve bit×cats	decision space	share of naïve	present-only	share of naïve	bit-valid masks
Clinic	607,500	295,650	48.7%	30,416	~5.0%	7 / 32
Inpatient	241,920	126,000	52.1%	24,346	~10.1%	20 / 64

8.1.4 8.1.2 Latency dispersion — Clinic (9-query bundle; 30 cold samples)

query	median (ms)	p95 (ms)	mean (ms)	notes
count_all	1.897	2.749	4.631	`clinic_visit_data`
top_groups_full	426.257	536.968	446.148	`*_decision_space_mat`
top_groups_present	340.484	461.184	361.629	`*_present_mat`
kpi_by_site	7.860	10.131	8.163	
kpi_by_department	8.190	10.987	8.432	

query	median (ms)	p95 (ms)	mean (ms)	notes
kpi_by_age	7.565	10.182	8.115	
kpi_by_appt_type	7.371	11.231	8.100	
kpi_by_role	8.007	14.565	8.780	
kpi_by_site2	7.554	10.460	7.930	
bundle total	**820.329**	**921.278**	**826.442**	sum across queries

8.1.5 8.1.3 Latency dispersion — Inpatient (9-query bundle; 30 cold samples)

query	median (ms)	p95 (ms)	mean (ms)	notes
count_all	1.783	4.523	3.341	`inpatient_admission_data`
top_groups_full	288.001	412.979	306.874	`*_decision_space_mat`
top_groups_present	256.711	345.071	262.424	`*_present_mat`
kpi_by_site	7.572	14.879	8.355	
kpi_by_ward	7.605	10.089	8.060	
kpi_by_age_group	7.110	11.196	7.874	
kpi_by_admission_type	6.521	14.342	7.513	
kpi_by_payer	7.394	12.863	8.129	
kpi_by_arrival_source	7.889	11.145	8.486	
bundle total	**606.973**	**702.404**	**619.035**	sum across queries

8.1.6 8.1.4 Ablations (clinic)

scenario	top_groups_full (ms)	top_groups_present (ms)
matching index **OFF**	433.259	189.876
matching index **ON**	429.538	348.270

Delta (present-only vs full, medians): 85.773 ms faster (-20.12%).
Delta (index ON vs OFF, full): -3.721 ms (-0.86%).
Delta (index ON vs OFF, present): +158.394 ms (+83.4%) — cold/noisy single-shot; present-only often prefers a sequential scan on a small matview over using the index.

(Inpatient present-only vs full, medians: 256.711 ms vs 288.001 ms → 31.290 ms faster, -10.9%.)

8.2 8.2 Interpretation

8.2.1 Summary at a glance

model	theoretical max (bit-only)	pruned decision space (rows)	share of theo-retical	bit-valid masks	present-only size	dashboard total (cold)	avg/query (cold)	dashboard total (warm)	avg/query (warm)
Clinic Visit	607,500	295,650	48.7%	**7**	30,416	879.837 ms	97.760 ms	825.539 ms	91.727 ms
Inpatient Admission	241,920	126,000	52.1%	**20**	24,346	609.826 ms	67.758 ms	622.174 ms	69.130 ms

Clinic Visit

- **Space & pruning.** Naïve space $2^5 \times |C|$ is **607,500**; rule pruning yields **295,650** (**48.7%** of theoretical).
- **Mask entropy. 7 / 32** masks are bit-valid; the small M keeps join keys compact and cache-friendly.
- **Present-only. 30,416** actually observed (mask, categories) tuples are captured in `clinic_visit_present_mat`, powering `top_groups_present`.
- **Latency.** "Dashboard" bundle (9 queries) totals **879.837 ms** cold vs **825.539 ms** warm (~**6.2%** faster) after buffer priming. Over **30 cold samples**, the **bundle median** is **820.329 ms** (p95 **921.278 ms**).

Inpatient Admission

- **Space & pruning.** Naïve space $2^6 \times |C|$ is **241,920**; pruning yields **126,000** (**52.1%**).
- **Mask entropy. 20 / 64** masks are bit-valid.
- **Present-only. 24,346** tuples in `inpatient_admission_present_mat`.
- **Latency. 609.826 ms** cold vs **622.174 ms** warm; slight warm $\geq$ cold is expected jitter for sub-second queries even with `pg_prewarm`. Over **30 cold samples**, the **bundle median** is **606.973 ms** (p95 **702.404 ms**).

Method

- Timings are taken **inside PostgreSQL** with
  `acbp_time_ms_many(labels[], queries[], iters=3, warm={false|true})`
  from a single backend (planner + executor overhead included).
- For "warm", we **prime shared buffers** for data tables, decision/present mats, and primary indexes using `pg_prewarm` (if available).

- Raw per-query latencies (including "full" vs "present-only") are recorded in each model's dashboard_perf.csv.

Dashboard query bundle (labels → SQL)

- count_all → SELECT COUNT(*) FROM "<model>_data".
- top_groups_full → SELECT * FROM acbp_bench_full_join('<model>','<model>_data', true, <topN>).
- top_groups_present → SELECT * FROM acbp_bench_full_join_present('<model>','<model>_data', <topN>) (if present mat exists).
- kpi_by_<col> ($\leq$ 5 columns) — SELECT <col> AS key, COUNT(*) FROM "<model>_data" GROUP BY 1 ORDER BY 2 DESC LIMIT <topN>.
- kpi_by_<col1>_<col2> — same, grouped by two keys.

Interpretation highlights

- **Structural pruning, not luck.** ACBP reduces the search space *before* query time: only **7/32** (Clinic) and **20/64** (Inpatient) masks are bit-valid; category rules cut the naïve bit×cats product to **48.7% / 52.1%**; intersecting with observed tuples shrinks it further to ~**5% /** ~**10%**. These artifacts exist regardless of the data and explain why group-bys touch far fewer rows.

- **Latency tracks the space.** With identical 9-query bundles and cold caches, medians are ~**0.82 s** (Clinic) and ~**0.61 s** (Inpatient) end-to-end, with the heavy queries (top_groups_*) ~**20%** (Clinic) and ~**11%** (Inpatient) faster on *_present_mat than on the full decision space.

- **Indexes help when they should.** On Clinic, the "matching index" barely changes the full-space query (-0.86%) and can be counter-productive on a tiny present-only matview (planner may pick an index path slower than a seq scan). The big win comes from **ACBP's compiled artifacts**—valid-mask pre-enumeration and pruned/present decision spaces—rather than hand-tuned query tricks.

- **Takeaway.** ACBP turns rules into SQL-native structures that (1) pre-filter impossible states, (2) prune the category cross-product, and (3) optionally restrict to observed tuples. Those three levers explain the consistent sub-second per-query medians and the robust cold-cache bundle times you measured.

Sanity checks

- **Validator parity.** Joining via <model>_valid_masks_mat and evaluating acbp_is_valid__<model>(mask) produce **identical counts of valid rows** for each model (see Appendix G for the exact SQL).

- **Rule effect.** Category rules reduce the naïve bit×cats product to **48.7–52.1%**, cutting aggregation work roughly in half.

- **Present-only advantage.** Intersecting D with observed tuples (**30,416 / 24,346**) yields additional aggregation wins without changing semantics.

Threats to validity

- **Synthetic data & independence** assumptions may differ from production; CSVs + SQL are provided to enable re-runs.

- **Single backend; default config** (no parallelism, conservative memory) — results are a baseline.

- **Cache effects.** `pg_prewarm` helps but OS cache and sub-second jitter remain.

9 Related Work

Decision tables (DMN) [dmn], policy-as-code (Rego/OPA) [opa], and SAT-based configuration [sat] solve adjacent problems. ACBP differs in compiling to pure SQL artifacts that can be indexed and materialized directly inside PostgreSQL.

10 Discussion & Limits

- **Bit ceiling** ($\leq$ 61 bits with `bigint`); shard masks or use multiple words when needed.
- **Category explosion**; prefer present-only and avoid high-cardinality core dimensions.
- **Event feeds**; LISTEN/NOTIFY is a pragmatic option, but protocol adapters (e.g., HL7) are future work.
- **Portability**; emitted SQL targets Postgres. Other engines need light adaptation.

11 Availability & Reproducibility

Code & DSL: https://github.com/DotKBoy-web/acbp Docs & schema: https://dotkboy-web.github.io/acbp/ Software DOI (v0.3.2): https://doi.org/10.5281/zenodo.16888028

A reproducible script and synthetic data sketch are included above; no external systems required.

12 Ethics

This paper uses synthetic data only; no personal or protected data are involved.

13 References

Bayer, Mike, and contributors. 2024. "SQLAlchemy 2.x Documentation." https://docs.sqlalchemy.org/.

Benavides, David, Sergio Segura, and Antonio Ruiz-Cortés. 2010. "Automated Analysis of Feature Models 20 Years Later: A Literature Review." *Information and Software Technology*.

Codd, E. F. 1970. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* 13 (6): 377–87.

"Decision Model and Notation (DMN)." 2019. https://www.omg.org/dmn/.

"Docker Engine Overview." 2024. https://docs.docker.com/engine/.

LISTEN and NOTIFY. 2023. PostgreSQL Global Development Group. https://www.postgresql.org/docs/16/sql-notify.html.

MacFarlane, John. 2024. "Pandoc User's Guide." https://pandoc.org/MANUAL.html.

Materialized Views. 2023. PostgreSQL Global Development Group. https://www.postgresql.org/docs/16/rules-materializedviews.html.

"Open Policy Agent (OPA) and Rego." 2024. https://www.openpolicyagent.org/.

Pg_prewarm. 2023. PostgreSQL Global Development Group. https://www.postgresql.org/docs/16/pgprewarm.html.

PostgreSQL 16 Documentation. 2023. PostgreSQL Global Development Group. https://www.postgresql.org/docs/16/.

Sluijs, Gijs van der. 2023. "Fvextra: Extensions for Fancyvrb." https://ctan.org/pkg/fvextra.

Society, American Mathematical. 2023. "User's Guide for the Amsmath Package." https://ctan.org/pkg/amsmath.

"Streamlit Documentation." 2024. https://docs.streamlit.io/.

Using EXPLAIN. 2023. PostgreSQL Global Development Group. https://www.postgresql.org/docs/16/using-explain.html.

Warren Jr., Henry S. 2013. *Hacker's Delight*. 2nd ed. Addison-Wesley.

14 Appendix

14.1 Appendix A Hardware, OS & DB Settings

- **Machine:** `<CPU model, cores/threads>`, RAM `<N GiB>`, storage `<NVMe/SATA, size>`.
- **OS:** `<Windows 11 + WSL2 / Linux distro & version>`, Docker Engine `<version>`.
- **Database:** PostgreSQL **16-alpine** (`acbp-pg`), default config; `pg_prewarm` available.
- **Ports:** host **5434** — container **5432**.

Optional tuning for larger sweeps:
`shared_buffers=2GB`, `effective_cache_size=6GB`, `work_mem=64MB`, `maintenance_work_mem=512MB`, enable parallel query.

14.2 Appendix B Data Dictionary

14.2.1 B.1 Clinic dataset (clinic_visit_data, 50,000 rows)

column	type	description
mask	bigint	Bitmask over flags [booked, checked_in, seen_by_doctor, canceled, rescheduled] (bit 0 = booked).
patient_mrn	text	Synthetic MRN (seeded).
sex	text	M,F,Other (weighted).
language	text	EN,AR (weighted).
city	text	Riyadh,Jeddah,Dammam,Mecca,Medina (weighted).
appt_type	text	{NewPatient, FollowUp, Urgent, Procedure, Teleconsult}.
site	text	{Main, Annex, Downtown}.
age_group	text	{Peds, Adult, Geriatric}.
department	text	{General, Cardiology, Orthopedics, Imaging, Pediatrics}.
provider_role	text	{Attending, Resident, NP/PA}.
modality	text	{InPerson, Virtual}.
visit_hour	text	{08:00, 09:00, 10:00, 11:00, 14:00}.
weekday	text	{Mon–Fri}.
insurance	text	{SelfPay, Private, Government}.

Integrity w.r.t. model (examples): - `Teleconsult` -> `modality='Virtual'`.
- `department='Pediatrics'` -> `age_group='Peds'`.
- `modality='Virtual'` -> `department IN ('Imaging','Orthopedics')` is avoided in the generator.
- A small (<6%) “noisy” tail deliberately breaks some rules to exercise validators.

14.2.2 B.2 Inpatient dataset (inpatient_admission_data, 50,000 rows)

column	type	description
mask	bigint	Bitmask over flags [booked, checked_in, in_icu, discharged, expired, transferred] (bit 0 = booked).
patient_mrn	text	Synthetic MRN (seeded).
sex	text	M,F,Other.
language	text	EN,AR.
city	text	Riyadh,Jeddah,Dammam,Mecca,Medina.
admission_type	text	{Elective, Emergency, Transfer}.
site	text	{Main, Annex}.
age_group	text	{Adult, Peds}.
ward	text	{Medical, Surgical, ICU, StepDown}.
payer	text	{SelfPay, Private, Public}.
arrival_source	text	{ED, Clinic, Transfer, Direct}.
admit_hour	text	{00:00, 04:00, 08:00, 12:00, 16:00, 20:00}.
weekday	text	{Mon-Sun}.

Integrity w.r.t. model (examples): - in_icu=1 cases preferentially get ward='ICU'.
- discharged=1 with arrival_source='Transfer' is suppressed (rule forbids it).
- A small (~5%) noisy tail injects counter-examples (e.g., ICU in non-ICU wards) to test explainers.

14.3 Appendix C Re-run Recipe (end-to-end)

```
# 1) Helpers
./acbp.sh reinstall-db-utils

# 2) Compile & apply
./acbp.sh compile-apply models/clinic_visit.v0.json
./acbp.sh compile-apply models/inpatient_admission.v0.json

# 3) Generate data (50k each; writes dataset/*.csv)
python make_data.py clinic_visit --rows 50000 --seed 42
python make_data.py inpatient_admission --rows 50000 --seed 43

# 4) Tables (match data dictionaries)
./acbp.sh psql-c "
DROP TABLE IF EXISTS clinic_visit_data CASCADE;
CREATE TABLE clinic_visit_data(
  mask bigint NOT NULL,
  patient_mrn text, sex text, language text, city text,
  appt_type text, site text, age_group text, department text, provider_role text,
  modality text, visit_hour text, weekday text, insurance text
);
DROP TABLE IF EXISTS inpatient_admission_data CASCADE;
CREATE TABLE inpatient_admission_data(
  mask bigint NOT NULL,
  patient_mrn text, sex text, language text, city text,
  admission_type text, site text, age_group text, ward text, payer text,
  arrival_source text, admit_hour text, weekday text
);
"

# 5) Import
docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY clinic_visit_data
↪ FROM STDIN WITH CSV HEADER" < dataset/clinic_visit_data_part1.csv
docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY clinic_visit_data
↪ FROM STDIN WITH CSV HEADER" < dataset/clinic_visit_data_part2.csv
docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY
↪ inpatient_admission_data FROM STDIN WITH CSV HEADER" <
↪ dataset/inpatient_admission_data_part1.csv
docker exec -i acbp-pg psql -U postgres -d postgres -c "\COPY
↪ inpatient_admission_data FROM STDIN WITH CSV HEADER" <
↪ dataset/inpatient_admission_data_part2.csv

# 6) Stats, indexes, present-only mats
./acbp.sh vacuum clinic_visit_data
./acbp.sh vacuum inpatient_admission_data
```

```
./acbp.sh psql-c "SELECT
↪  acbp_create_matching_index('clinic_visit','clinic_visit_data'); SELECT
↪  acbp_materialize_present('clinic_visit','clinic_visit_data');"
./acbp.sh psql-c "SELECT
↪  acbp_create_matching_index('inpatient_admission','inpatient_admission_data');
↪  SELECT
↪  acbp_materialize_present('inpatient_admission','inpatient_admission_data');"

# 7) Bench & export
./acbp.sh paper-bench clinic_visit 12
./acbp.sh paper-bench inpatient_admission 12
./acbp.sh paper-bench-dashboard clinic_visit 12 3
./acbp.sh paper-bench-dashboard inpatient_admission 12 3

# 8) Make snippet & inject into paper
./acbp.sh paper-results-md
./acbp.sh paper-update-paper
```

14.4 Appendix D What the helpers do

- acbp_create_matching_index('<model>','<data>')
  Builds an index tailored to the decision-space join pattern: leading on mask, then the model's category columns in the same order as <model>_decision_space.
  Result: a btree index named idx_<data>_match. Improves joins from <data> -> <model>_decision_space(_mat) and <model>_present_mat.

- acbp_materialize_present('<model>','<data>')
  Computes and materializes <model>_present_mat as the intersection of the model's decision space with SELECT DISTINCT (mask, <categories...>) FROM <data>. Also runs ANALYZE so plans are calibrated.
  Idempotent: safe to re-run when <data> changes.
  Related: acbp_refresh_present('<model>') refreshes the matview concurrently (if supported).

- acbp_bench_full_join('<model>','<data>', include_bits, topN) / acbp_bench_full_join_present('<model>','<data>', topN)
  Emit consistent **top-N groupings** over the full decision space vs the present-only space, enabling apples-to-apples latency comparisons. Signatures may include a boolean to include bit columns in the projection.

- acbp_time_ms_many(labels[], queries[], iters, warm)
  Server-side timing harness. Executes each SQL text in a **single backend** for iters repetitions and returns (label, ms) aggregates. When warm=true, you typically pre-prime shared buffers (e.g., via pg_prewarm) before calling it.

14.5 Appendix E Interpreting `EXPLAIN` (quick notes)

- **Operators you should see**
  - `Bitmap Index Scan` / `Bitmap Heap Scan` on `*_decision_space_mat` for `top_groups_full`.
  - `Seq Scan` + `GroupAggregate` may appear on small `*_present_mat` (it can be cheaper to scan all).
- **Buffers**
  - Check `Buffers: shared hit/read` to distinguish warm vs cold behavior. Warm runs should show far fewer `reads`.
  - `Rows Removed by Filter` should be near **zero** thanks to compile-time pruning.
- **Other indicators**
  - `Planning Time` + `Execution Time` give end-to-end cost; both are included in our harness.
  - Parallel plans are typically off under default settings; enabling parallelism can change operators and timing.
  - JIT may be enabled/disabled depending on your build; for short queries it often doesn't help.

14.6 Appendix F Glossary

- **ACBP** — *Al Anazi Categorical-Boolean Paradigm*: a DSL that compiles domain rules into SQL artifacts.
- **Mask (F)** — Ordered bits over boolean flags, packed into a `bigint`.
- **Categories (C)** — Finite string dimensions (e.g., site, ward) crossed with masks.
- **Valid masks (M)** — The subset of masks that satisfy all **bit-only** rules.
- **Decision space (D)** — All `(mask, categories)` pairs that satisfy **all** rules (bit + category).
- **Present-only (D__present)** — `D` intersected with the `(mask, categories)` tuples that actually occur in source data (from `<data>`).

14.7 Appendix G Validation queries (parity & noise)

```sql
-- A) Valid-row counts by the function vs the join (they must match)

-- Clinic
WITH f AS (
  SELECT COUNT(*) AS n FROM clinic_visit_data d
  WHERE acbp_is_valid__clinic_visit(d.mask)
), j AS (
  SELECT COUNT(*) AS n FROM clinic_visit_data d
  JOIN clinic_visit_valid_masks_mat vm ON vm.mask = d.mask
)
SELECT 'clinic_function' AS method, n FROM f
UNION ALL
SELECT 'clinic_join'     AS method, n FROM j;

-- Inpatient
WITH f AS (
  SELECT COUNT(*) AS n FROM inpatient_admission_data d
  WHERE acbp_is_valid__inpatient_admission(d.mask)
), j AS (
  SELECT COUNT(*) AS n FROM inpatient_admission_data d
  JOIN inpatient_admission_valid_masks_mat vm ON vm.mask = d.mask
)
SELECT 'inpatient_function' AS method, n FROM f
UNION ALL
SELECT 'inpatient_join'     AS method, n FROM j;

-- B) Fraction of intentionally invalid (noisy) rows observed in the data

SELECT 'clinic'    AS model,
       COUNT(*) FILTER (WHERE NOT acbp_is_valid__clinic_visit(mask))::float /
        ↪ COUNT(*) AS invalid_share
FROM clinic_visit_data
UNION ALL
SELECT 'inpatient' AS model,
       COUNT(*) FILTER (WHERE NOT
        ↪ acbp_is_valid__inpatient_admission(mask))::float / COUNT(*) AS
        ↪ invalid_share
FROM inpatient_admission_data;
```

(Run these and, if you want, capture the numbers into your paper.)

14.8 Appendix H System details (fill-in)

- CPU: ,
- RAM:
- Storage: , , filesystem
- OS: <Windows 11 + WSL2 / Linux distro + kernel>, Docker
- PostgreSQL: 16-alpine; JIT ; parallel query
- Key GUCs (if non-default): shared_buffers=…, work_mem=…, effective_cache_size=…, maintenance_work_mem=…

Last updated: 2025-08-18 (Asia/Riyadh).