

Projektfeladat

Téma: Hűtőpanelek adatainak elemzése (2007–2023)

Készítették: Fodor Csaba Krisztián és csapata ...

-
-
-
-

1. Milyen adatbázist használtunk és miért?

A projekthez SQLite adatbázist választottunk, mert könnyen kezelhető, nem kell hozzá külön szerver, és jól működik együtt Pythonban.

A cél az volt, hogy a CSV fájlokból gyorsan betölthető, strukturált adatbázist kapjunk, amit egyszerűen lehet lekérdezni és riportokat generálni belőle.

Kezdetben gondolkodtunk MySQL-en is, de felesleges lett volna a telepítés, ezért maradtunk az SQLite-nál.

2. Az adatbázis megtervezése (ER-diagram)

Az adatbázis három fő táblából áll:

panel – a hűtőpanelek adatai (id, név, mértékegység, min/max tartomány)

batch – adagok (id, kezdési idő, befejezési idő, időtartam)

measurement – mérések (panel_id, időpont, érték, quality_code)

Ezek közül a measurement kapcsolódik mind a panel, mind a batch táblához.

Készítettünk egy nézetet (v_measurement_in_batch) is, ami összeköti az adatokat az adagok időintervalluma alapján.

Az ER-diagramot több verzióban újrarajzoltuk, mire logikusan átlátható lett (főleg a batch–measurement kapcsolatnál).

3. Az adatbázis létrehozása

A sémát SQL-ben írtuk meg (sql/001_schema.sql), hogy bármikor újra fel lehessen építeni.

A sémát a 01_build_db.bat futtatja, ami automatikusan lefuttatja a .sql fájlt, majd a loader.py betölti az adatokat.

A választott metódus azért jó, mert egyetlen kattintással reprodukálható a teljes projekt.

4. Adatok betöltése és ellenőrzése

A loader.py a data mappában található CSV-ket tölti be.

Az Adagok.csv a batch táblába kerül.

A Hűtőpanelek.csv alapján töltjük a measurement adatokat.

A betöltés során több ellenőrzés történik:

dátum/idő konverzió többféle formátumból

számok konvertálása float és int típusra

hibás vagy tartományon kívüli értékek kihagyása

automatikus min/max beállítás (-60°C–200°C)

A betöltés végén automatikusan készül biztonsági mentés a backup mappába.

5. Jogosultságok és tranzakciókezelés

Az SQLite-ban nincs klasszikus DCL, de ha lenne, a következő jogosultságokat adnánk:

Collector: csak INSERT jogosultság a measurement táblára

Engineer: SELECT + UPDATE jog a panel és batch táblákra

Admin: mindenhez teljes hozzáférés

Tranzakciót is használtunk (pl. `conn.commit()`), így a betöltés során, ha hiba van, rollback-elhető.

Egy példát is beépítettünk a loaderbe – demonstrációs céllal.

6. Lekérdezések, JOIN-ok, aggregáció

Több riport is ilyen logikával készült:

`last_values.csv`: minden panel utolsó mérése

`daily_avg.csv`: napi átlaghőmérsékletek

`batch_avg.csv`: adagok szerinti átlagok

`outliers.csv`: 3σ -n kívüli mérések

`daily_minmax.csv`: napi minimum és maximum értékek

Ezek mind SQL-lel készültek JOIN és GROUP BY használatával.

Céljuk a panelek állapotának és stabilitásának ellenőrzése volt.

7. Allekérdezések, UNION

A `outliers.csv` riportban CTE (WITH) szerkezetet használunk, ami tulajdonképpen egy allekérdezés.

Ezen belül kiszámoljuk a panelekenkénti szórást és átlagot, majd 3 szórásnál nagyobb eltéréseket listázunk ki.

UNION példát is kipróbáltunk (pl. naponta és adag szerint egyesített átlagok), de végül nem volt szükség rá a riportokban.

8. Teljesítményoptimalizálás

Egyedi kulcs: (panel_id, ts_utc) azonosítja az egyedi méréseket.

Index: ix_measurement_panel_day gyorsítja a napi riportokat.

10 évnyi adat esetén javaslat: partícionálás évenként, vagy Parquet formátum + DuckDB elemzés.

A riportok így is gyorsan lefutnak, mert a napi átlagok számításánál csak a szükséges oszlopokat használjuk.

9. Adattisztítás

A loader.py rengeteg hibás adatot kiszűr:

nem konvertálható dátumok vagy értékek

tartományon kívüli mérések

duplikált timestampek

Az outliereket külön riportban listázzuk ki, így később manuálisan is ellenőrizhetők.

10. A hibák azonosítása és érvényesítés

A hibás adatokat fizikai-logikai ismeretek alapján szűrtük:

pl. -60°C alatti vagy 200°C feletti hőmérséklet fizikailag nem reális

a panel 7-nél adathiány volt, ezt dokumentáltuk

Az adatbevitelhez a jövőben érdemes lenne CHECK (value BETWEEN -60 AND 200) típusú kényszereket alkalmazni.

11. Ipari adatgyűjtő szerepkörök

Ha ez egy valós gyárban futna, a következő szerepkörök lennének:

Collector: szenzoradatok mentése

Operator: riportok megtekintése, alap szűrés

Engineer: hibás mérések javítása

Admin: teljes hozzáférés, mentések kezelése

12. Biztonsági mentés

Két megoldásunk van:

Automatikus mentés minden betöltés végén (loader.py → backup/project_backup_YYYYMMDD_HHMMSS.db)

Kézi mentés SQLite parancsból:

```
VACUUM INTO 'backup/project_snapshot.sqlite';
```

A backup mappában több verzió is elérhető, így könnyen visszaállítható az adatbázis.

Összefoglalás, tapasztalatok

A projekt közben több kisebb zsákutcába is belefutottunk (pl. panel7 hiányzó adatai, karakterkódolási hibák a CSV-ben), de minden problémát sikerült megoldani.

Sokszor újra kellett tölteni a DB-t és javítani a scripteket, viszont ez segített abban, hogy most már teljesen automatikusan működik a betöltés és riportkészítés.

A grafikonokat matplotlib-pel készítettük, és nagyon jól látszanak rajtuk az esetleges kiugró értékek vagy ingadozások.

A végeredmény szerintünk stabil, átlátható, és minden pontot lefed, amit a feladatkéresek kért.

Záró gondolat

Őszintén szólva a projekt sokkal tanulságosabb volt, mint gondoltuk: egy CSV-ből induló, “egyszerű” adatbázis-feladattól végül egy teljesen működő mini-adatgyűjtő rendszer lett.

Ha ezt ipari környezetben futtatnánk, csak a hardveres adatgyűjtőt kéne rákötni, és máris élő rendszerként működne.

Köszönjük a figyelmet!