Database-Driven Talent Matching Platform (Bridgeway)

Database Implementation Report

[Database design overview and schemas] [20]

The BridgewayDB system is designed as a full two-sided marketplace database that manages engineers, clients, jobs, applications, vetting, ratings, and platform-level analytics. At its core, the schema uses tbl_User as the parent table, which is extended 1:1 into tbl_Engineer_Profile and tbl_Client_Profile. Supporting entities such as skills, job requirements, engineer skills, vetting reviews, and endorsement ratings form a normalized relational structure that ensures both flexibility and scalability.

To optimise real-world usage patterns, the schema includes several specialised components: partitioning on the high-volume tbl_Job_Application table, a denormalized rating cache, search-optimised views, and multiple nonclustered indexes. The system is fully modularized; each major operational component (matching engine, vetting pipeline, search engine, job lifecycle) has dedicated stored procedures, functions, triggers, and views. As a result, the final system is both scalable (tested with large synthetic datasets) and operationally rich, supporting automated matching, soft deletes, cascading updates, vetting workflows, dashboards, and search algorithms.

[Feature descriptions] [20]

1. **User and profile management:**
   The database uses a central user table that is extended into engineer and client profiles using one-to-one relationships. This structure supports role separation and clean management of availability, experience, company information, and other identity attributes.

2. **Skill and job requirement modelling**
   Skills, engineer skills, and job skills are stored in normalized tables with many-to-many bridge structures. Required and preferred skills are represented separately to support accurate filtering and matching.

3. **Partitioned job applications**
   The job application table is partitioned by year based on the created_at column. This improves performance, supports efficient archival, and allows the system to scale to very large datasets.

4. **Rating cache for performance**
   A dedicated rating cache table stores average ratings and rating counts for each engineer. This reduces repeated aggregation cost in searches and dashboards.

5. **Matching engine functions and procedures**
   Matching is implemented through functions that compute eligibility, skill coverage, experience alignment, rating contribution, and timezone compatibility. Stored procedures calculate match scores for all relevant engineers per job and refresh scores for all open jobs.

6. **Vetting and trust system**
   The vetting pipeline includes functions that compute vetting scores and final statuses. Stored procedures insert reviews and finalise decisions. Triggers automatically recalculate vetting outcomes after new reviews. A soft delete mechanism archives removed engineers and updates their pending applications.

7. **Job and application lifecycle management**
   Procedures support creating jobs, applying to jobs, and updating application or job statuses. Triggers ensure data integrity by assigning default match scores and updating job status when an application is accepted.

8. **Search and filtering engine**
   The system supports multi-criteria searches based on skills, vetting status, experience, rating, and timezone. Supporting functions handle rating calculations and skill list parsing. Views provide simplified projections for efficient search operations.

9. **Analytics and dashboard views**
   Views aggregate job information, engineer profiles, application summaries, candidate rankings, and open jobs with top candidates. These projections simplify reporting and monitoring tasks for the platform.

10. **Reporting and statistical procedures**
    The system includes procedures that compute monthly platform activity, engineer-level statistics, match score patterns, and historical metrics. A utility function standardises date boundaries for reporting.

11. **Indexing and performance optimisation**
    Nonclustered indexes support common query paths across users, jobs, job applications, vetting reviews, and ratings. Filtered indexes improve performance for active application states. A maintenance procedure rebuilds indexes and updates statistics.

12. **Synthetic data generation**
    A generator procedure creates large volumes of synthetic engineers, clients, jobs, skills, applications, vetting reviews, and ratings. This supports scalability testing and performance evaluation under production-like workloads.

13. **Additional automated behaviours**
    Triggers enforce soft deletion, cascade rejections for removed engineers, update job statuses during application changes, and maintain data consistency across workflows.

Names	**Roll numbers**
Muhammad Shazil Nadeem	27100183
Haider Wajahat	27100252
Muhammad Taimur Jahanzeb	27100028
Areeba Naveed	27100239
Syed Zaid Ahmed	27100192