Enabling Collaborative Administration and Safety Fences: Factored Privileges in SQL Databases

Arnon Rosenthal
The MITRE Corporation
202 Burlington Rd
Bedford, MA, USA
arnie@mitre.org

Edward Sciore
Computer Science Dept
Boston College
sciore@bc.edu

Abstract

An access policy has many aspects, concerning both information and physical execution. Agglomerating them into a single SQL grant makes policies much harder to administer, especially at enterprise scale where administrators need to collaborate. We present a way to specify policies as a conjunction of factors, in a simple, regular way. Each factor decision poses a simple question, and when a circumstance changes, only the relevant factor needs to be revisited. Factors are also help for establishing "safety fences" on an administrator's work, and for separating (global) information privileges to enable more powerful inference rules. To ease integration into existing systems, factor privileges employ the same interfaces and rules as ordinary SQL privileges, rather than multiple new top-level, awkwardly-interacting constructs (such as "autonomy" and "prohibition").

1 Introduction

The SQL access control model was designed in the 1970s for databases to be used within a closed organization. Either the DBA or the data owner could be relied on to deal with all aspects of a permission decision. In contrast, today's enterprise (and virtual enterprise) database has many stakeholders. For example, data is distributed across multiple platforms and networks, each with their own security, performance, and charge-back concerns. DBAs are typically concerned with giving applications the data they want; security officers wish to minimize the privileges granted to ensure confidentiality, integrity and availability; privacy officers want traditional security, but also want to ensure that individuals are guaranteed appropriate controls over data that describes them (e.g., to know about and reply to unfavorable information).

Today, interaction among these administrators occurs informally or via memoranda. The DBMS access-control system neither documents the stakeholders' separate policies nor enforces them. Instead, each grant requires the consideration of all aspects; when circumstances change, all aspects must be reviewed.

Today, administrators who receive *grant option* are not expected to be guided by whim, or to judge each situation from scratch. They should operate within bounds given by organizational policy. Enterprises may give their members guidance (generalized policies and principles) that they use to make specific decisions [Bark03].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Higher administrators may turn the guidance into boundaries (i.e., *safety fences*) that others should not cross. Unfortunately SQL gives no automated support for such fences.

Many authors have sought to meet this pressing need with "negative privileges" (also known as "prohibitions") [BJS99, BJWW02]. Such mechanisms are quite controversial. For systems of moderate size, opponents argue that they require administrators to flip between positive and negative, and provide unpredictable behavior, especially when overrides are needed. Enterprises have further cause for concern. Most proposals include new features that are not orthogonal to existing ones (notably with respect to views and autonomy), and there is no easy way for many administrators to work together in setting the fences.

Our research goal is to extend the SQL privilege model and services in order to better support collaborative, decentralized administration. In this paper we discuss the part of our approach that involves *factoring* privileges into smaller, easier to use granules. We then show how the use of factor granules can solve the above problems.

2 Factor Granules

The decision to grant a privilege (i.e., for an action on an object) is usually the conjunction of several, smaller decisions. We represent the types of decisions as a *factor tree*, where each factor represents an aspect to be considered in a privilege decision. The root of the tree denotes the complete, full privilege; each leaf of the tree denotes an atomic decision; and an intermediate node denotes an aggregate decision (in effect, the conjunction of all its leaves). A *factor granule* is a "partial privilege" associated with a node of the factor tree, and denotes that from that aspect, the privilege is appropriate.

We extend SQL so that granules (and not privileges) are the unit of granting. A user receives a privilege when it obtains granules for each of the privilege's factors. Granules can be granted at any level; granting a non-leaf granule is equivalent to granting all of the granules in its subtree. Granting the root granule is thus equivalent to granting the privilege.

Factor granules can be independently granted, revoked, and inferred on views (just like SQL privileges), but are better units of management and collaboration. The idea is that different administrators will have responsibility over different factors. Obtaining a granule can be thought of as having the appropriate granule administrator "sign off" on the privilege.

Figure 1 depicts an example of a factor tree that an enterprise might adopt.[1] The root has two children. One child heads the subtree of *information factors*, which represent decisions about the information content of the protected object, independent of a particular physical implementation. The other child heads the subtree of *execution factors*, which may represent decisions about the physical object's environment (e.g., should the grantee be running on this machine? on this copy of the data?).

When an object is created, we grant administrative rights on all information factors to the object creator, and execution factors to the DBA of the DBMS in which it is created. This initial split reduces the threat from either of the two insiders. The SQL model (applied to factors) lets them further delegate their rights to administrators for each factor, ideally using a script that runs automatically upon creation.

2.1 Information Factors

The *information content* of the database expresses "external world" facts (e.g., that John Smith receives penicillin), not the contents of individual memory cells on specific machines. Information factors specify whether a privilege is appropriate, independent of any physical representation. Figure 1 depicts two kinds of information factor. An *ordinaryInfo* factor granule is granted by a business expert, and denotes that it is reasonable (from the point of view of the business) for the requestor (user or roles) to access the specified information. OrdinaryInfo

[1] Our full model (forthcoming) allows a different factor tree for each operation, and includes ways to create and delete factors transparent to other administrators.

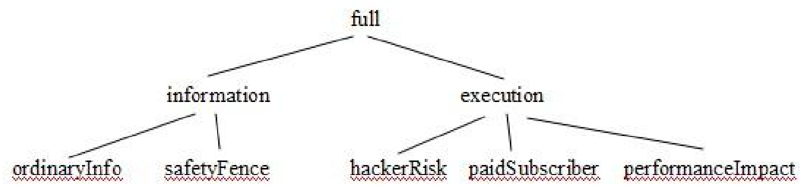

Figure 1

granules should typically be fine-grained, and the right to grant them may be widely delegated among many business experts. SafetyFence granules (Section 2.3) are granted by the security administrator, and denote those users who satisfy general access criteria.

2.2 Execution Factors

All decisions influenced by the physical system are called *execution factors*. Most execution factors permit administrators to control what operations may execute at a particular service locus (e.g., site, machine, DBMS, application server, output device and time, location). Figure 1 depicts three useful kinds of execution factors, which represent the physical-access limitations of allowing only paid subscribers, keeping potential hackers off the machine, and controlling workload. The *paidSubscriber* factor might be managed by a sales administrator, the *hackingRisk factor* by a system manager or security officer, and the *performanceImpact* factor by a DBA.

2.3 Safety Fence Factors

Many business experts and DBAs value functionality over security, and give users what they want [Wi01]. Security-conscious administrators (such as a data provider, local system administrator, or security officer) need to bound the full privileges that a class of users may receive. A *safetyFence* granule can be used for this purpose. The security administrator grants safetyFence granules only to the "acceptable" user population. Since a user cannot receive a privilege without obtaining all of its granules, the system can guarantee that a user will never receive a privilege without approval from the security administrator. SafetyFence granules protect the system from errors and abuses, and also allow veto power by a site's security administrator. They can also enforce general organizational guidelines, such as "only managers and analysts can see product profitability figures."

One can insert as many safety fence factors as are convenient, e.g., on either an execution or an information factor. They are similar in spirit to the *prohibitions* of [Bert99], but phrased positively. The advantages (versus prohibitions) are:

- The semantics are inherited from the general concept "factor" - there is no need for administrators to learn new rules. Furthermore, factors behave *exactly* like full privileges in their interactions with other model features (e.g., views, restrictive predicates, ...).

- One can insert as many safety fence factors as there are independent administrators who can veto a privilege. This greatly simplifies coordination. (There is no separate "override" concept. To give an administrator the power to override a safety fence factor, one gives them the Grant privilege on that factor).

We anticipate that SafetyFence grants will often be to high level roles (e.g., "any EMPLOYEE can execute on the internal portal") and cover large sets of objects. Due to large granularity, we expect that their administration will not be a severe burden.

3 Benefits of Factoring

Factor granules improve privilege administration in several important ways:

- They allow collaboration among administrators.
- They reduce the complexity of administration.
- They allow information privileges to be inferred from view privileges, even if the query does not reference the view.
- They guide security models for virtual and materialized views.
- They fit elegantly info the SQL model.

We discuss each benefit in turn.

3.1 Collaboration

Factor granules make it easy for multiple administrators to collaborate. The factor hierarchy for a privilege explicitly shows the decisions that are required to grant the privilege. The responsibility for each decision can be assigned to an administrator having the appropriate expertise (e.g. business expert, security officer, system administrator, etc.). Each administrator can grant factor granules independently and autonomously, with a privilege being given to a user only if all administrators agree.

Because access-control authority is distributed, there is no need for any one administrator to be the ”super-user”. Instead, each administrator has local authority over his particular factor(s) but no others. For example, many system administrators will no longer require the ability to see the data in order to do their job, and the DBMS will enforce this.

3.2 Less Complexity

Each leaf node in the factor hierarchy denotes a single decision. The question of whether to grant that factor granule is therefore focused and concrete, and its answer typically depends on a narrow slice of technical or domain knowledge. In contrast, a privilege requires consideration of the issues motivating every factor. Consequently, it is easier to determine the correctness and appropriateness of granting a factor granule than it is for granting a privilege. Although there are more granules to administer, each granule is more easily understood, and some factors can be granted to broad roles (e.g. EMPLOYEE) or large sets of objects. The hope is that the number of decisions increases very little, with each decision becoming easier to make and maintain. Testing this hypothesis is an important open problem.

3.3 Implicit Privileges

For information factors, the nature of the computation is not relevant - only the result matters. In [RoSc00], we used this to infer privileges because a computation *could* have been accomplished employing only views for which the user had adequate privileges. The next item further exploits this observation.

3.4 Materialized Views

Explicit management of information factors clarifies inference of privileges on replicated data (and by extension, materialized views). The information privileges are independent of replication (assuming replication delays do not change sensitivity). Autonomy difficulties do not apply to information privileges.[2] The following table summarizes the privileges given to creators of base tables, views, and materialized views. The difference between these three constructs is easily seen when information factors are considered separately from execution factors.

	information	execution
Base Table	all	granted by System Administrator
View	inferred from its definition	inferred from its definition
Materialized View	inferred from its definition	granted by System Administrator

3.5 Elegance

Factor granules behave exactly the same as privileges. Granules apply to both base and derived objects, and have all the administration operations (Grant, Revoke, grant option) identical to privileges. In fact, a privilege is just the special case of a granule for the factor called "full". This elegance means that our model retains the simplicity of SQL, without the need to expose administrators to special cases or entirely-new constructs (e.g., "inference", "autonomy", "privilege override") as is done elsewhere [CVS97, GuOl99, BJWW02]. In addition, previous distributed data security models mostly concern federations. We achieve most of their goals, extending to for *any* derived data, in *any* distributed architecture.

4 Summary and Future Work

In this paper, we proposed splitting privileges into smaller, factor granules. Although this extension to SQL is conceptually simple, it has far-reaching benefits: It simplifies privilege administration, helps administrators to collaborate, and allows policies to be more accurately specified and more easily understood. Moreover, functionality (such as "prohibitions" and "autonomy") that previously had required significant data model extensions, can be expressed straightforwardly using this single mechanism. In work to be reported elsewhere, we integrate factors smoothly into the privilege model for derived data.

There remain many open problems. Theoretical problems include: a formal comparison between prohibition models (with and without override) versus safety fences; should negatives be supported (just) as a user interface option? We need metrics for administration complexity, and performance models that can aid designers. We also need models for specifying and reasoning about privileges at different granularities.

Systems tasks include efficient enforcement of factor assertions at different granularities (perhaps by caching inferred privileges), plus environments for administering access policies in distributed systems. Most enforcement needs to be done in DBMSs (for efficiency and trust), but any large system will include DBMSs that differ in their capabilities. Therefore one may want a "permission manager" that installs privileges for enforcement in the component DBMSs.

Administration methodologies are perhaps the biggest challenge. They need to be flexible and yet easy to learn. Administrators need clear explanations of both the decisions required, and the guidance in making them. Perhaps a fixed (but subset-able) set of factor types can satisfy most needs? How should safety fences be managed, among multiple administrators? What are the needs for switching between positive and negative viewpoints?

[2] [GuOl] also had this insight, but formalized it less thoroughly. Instead of separating information from execution issues, they had "global" privileges. These presumably would be granted only by federation administrators, and (depending on power relationships) could be overridden by local system administrators.

5 References

[Bark03] W. Barker, "Guide for Mapping Types of Information and Information Systems to Security Categories", National Institute of Standards and Technology, 2003. http://csrc.nist.gov/publications/drafts.html

[BJS99] E. Bertino, S. Jajodia, P. Samarati, "A Flexible Authorization Mechanism for Relational Data Management Systems", *ACM Trans. Information Systems*, April 1999.

[BJWW02] Claudio Bettini, Sushil Jajodia, Sean Wang, Duminda Wijesekera, "Provisions and obligations in policy rule management and security applications," *VLDB Conference*, 2002.

[CVS97] S. De Capitani di Vimercati, P. Samarati, "Authorization Specification and Enforcement in Federated Database Systems", *Journal of Computer Security*, vol. 5, n. 2, 1997, pp. 155-188.

[GuOl99] E Gudes and MS Olivier, "Security Policies in Replicated and Autonomous Databases," in S. Jajodia (ed), *Database Security XII: Status and Prospects*, 93-107, Kluwer, 1999.

[RoSc00] A. Rosenthal, E. Sciore, "View Security as the Basis for Data Warehouse Security ", *CAiSE Workshop on Design and Management of Data Warehouses*, Stockholm, 2000.
http://www.mitre.org/tech/itc/staffpages/arnie/index.html

[Wi01] G. Wiederhold: "Collaboration Requirements: A Point of Failure in Protecting Information"; *IEEE Transactions on Systems, Man and Cybernetics*, Vol.31 No.4, July 2001, pages 336-342.