RQL : un langage “à la SQL” pour découvrir des règles à partir des données

Brice Chardin
LIAS, ISAE-ENSMA
France

Emmanuel Coquery
Université Lyon 1
CNRS, LIRIS, UMR 5205
France

Marie Pailloux
Université Blaise Pascal
CNRS, LIMOS, UMR 6158
France

Jean-Marc Petit
INSA Lyon
CNRS, LIRIS, UMR 5205
France

ABSTRACT

RQL (pour *Rule Query Language*) est un langage de requêtes “à la SQL” qui étend et généralise les dépendances fonctionnelles [1] à de nouvelles catégories de règles. RQL apporte aux analystes de données un outil pratique pour découvrir les implications logiques entre attributs d'une base de données. Ces implications peuvent mettre en évidence des problèmes de qualité de données ou de nouvelles corrélations inattendues entre les attributs. Le traitement de ces requêtes RQL est basé sur une technique de réécriture qui délègue un maximum de calculs au SGBD sous-jacent. Cette contribution vise à renforcer le lien entre la fouille de données et les bases de données et de faciliter l'utilisation de techniques de fouille par des analystes ou des étudiants habitués au SQL.

1. INTRODUCTION

La fouille de motifs peut être vue comme une partie automatisée de l'exploration de données. Par exemple, les dépendances fonctionnelles ou les dépendances fonctionnelles conditionnelles sont particulièrement utiles pour comprendre les données et identifier des problèmes de qualité [5]. Cependant, les techniques de fouille de motifs sont rarement utilisables directement par les analystes. La plupart du temps, ils ont à réaliser du pré-traitement entre différents systèmes et formats. Les codes de fouille eux-mêmes nécessitent souvent d'être compilés à partir de langages de programmation spécifiques. Toutes ces étapes sont hors de portées pour de nombreux analystes, tournant le procédé de bout en bout en cauchemar. La génération automatique de règles peut également submerger l'analyste par une quantité énorme de motifs, et rendre l'extraction d'information utile difficile. D'autres techniques ont été proposées pour interagir avec les données et fournir des retours utiles à l'analyste.

1. Nous utiliserons indifféremment les termes *règles*, *implications*, *dépendances* et *motifs* par la suite.

(c) 2014, Copyright is with the authors. Published in the Proceedings of the BDA 2014 Conference (October 14, 2014, Grenoble-Autrans, France). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.
(c) 2014, Droits restant aux auteurs. Publié dans les actes de la conférence BDA 2014 (14 octobre 2014, Grenoble-Autrans, France). Redistribution de cet article autorisée selon les termes de la licence Creative Commons CC-by-nc-nd 4.0.
BDA 14 octobre 2014, Grenoble-Autrans, France.

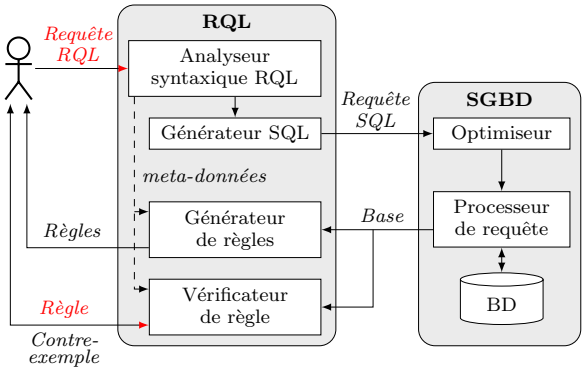


Figure 2: Traitement des requêtes RQL

Contribution de la démonstration Pour faciliter l'utilisation des techniques de fouille de motifs pour l'exploration de données, nous introduisons un langage de requête pour les règles, RQL (pour *Rule Query Language*), qui permet aux analystes habitués au langage SQL d'utiliser les techniques de fouille de motifs à l'aide une interface interactive et simple à utiliser. Dans cette démonstration, nous montrons l'utilisation de cette interface Web du point de vue de ces analystes. Nous nous focalisons sur l'expressivité de RQL à travers divers exemples, montrant ainsi la facilité de concevoir de nouvelles règles avec un langage très simple dérivé de SQL. Nous introduisons également comment les analystes peuvent interagir avec le système par l'intermédiaire des requêtes RQL et des contre-exemples issus de la base de données. Au cours de la démonstration, les participants sont invités à formuler leurs propres requêtes sur des bases de données prédéfinies pour découvrir les relations entre les attributs à l'aide des règles générées et des contre-exemples.

La figure 1 donne un aperçu de l'interface Web [2] pour RQL, disponible pour la recherche et l'éducation. Cette interface fournit un accès unifié aux données de l'utilisateur et aux techniques de fouille de motifs en utilisant deux langages déclaratifs : SQL et RQL.

À partir des travaux précédents [1, 2, 6], RQL vérifie les

2. `http://rql.insa-lyon.fr`

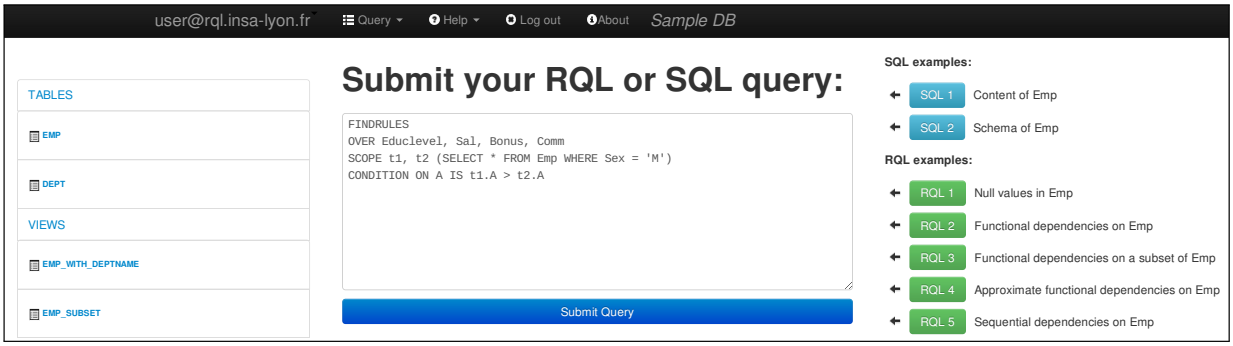


Figure 1: Interface Web pour RQL

axiomes d'Armstrong, i.e. le langage généralise les dépendances fonctionnelles à une nouvelle classe de dépendances basée sur les implications logiques (expressions de type : si ... alors). Nous avons prouvé que ces dépendances peuvent être calculées efficacement à partir de la base de données à l'aide d'un traitement en deux étapes. Premièrement, une requête SQL non triviale est générée pour calculer la *base* du système de fermeture associé aux implications recherchées. Cette base est également appelée un *contexte* dans la terminologie de l'analyse de concepts formels [8]. Ensuite, un algorithme tiré de l'état de l'art [12] est utilisé pour générer une couverture canonique des règles à partir de cette base. Cette approche permet à RQL de bénéficier de l'optimisation des requêtes du SGBD pour accéder aux données. Outre l'efficacité de l'optimisation, cette approche évite l'écueil de changer de système pour les analystes contribuant à leur simplifier leur tâche de découverte.

La figure 2 donne un aperçu de cette architecture vis-à-vis du traitement des requêtes RQL. Pour les requêtes SQL, l'application se contente de les faire suivre au SGBD sous-jacent, ce qui rend la transition entre SQL et RQL transparente pour l'utilisateur. L'objectif final de ce travail est d'intégrer les techniques de fouille de motifs au cœur des SGBDs [14].

Travaux connexes Définir des langages dédiés à la fouille de motifs est un objectif poursuivi de longue date [3], par exemple en utilisant des techniques de programmation par contrainte [10]. Cependant, nous soutenons que ces langages devraient bénéficier d'extensions directes du SQL, car les données sont souvent gérées par des SGBDs. D'autres approches pratiques, aussi proches que possible des SGBDs, ont été proposées pour interagir plus directement avec leurs moteurs de requête [7, 15, 4].

2. LE LANGAGE RQL

Pour illustrer le langage RQL, nous considérons l'exemple donné en figure 3 avec la relation *EMP*. L'attribut *Educlevel* représente le nombre d'années d'éducation formelle, *Sal* le salaire annuel, *Bonus* le bonus annuel et *Comm* la commission annuelle. La signification des autres attributs est immédiate.

Pour commencer, nous souhaitons extraire les dépendances fonctionnelles (DF) de la relation *Emp*. Pour mémoire, une DF $X \rightarrow Y$ est vérifiée dans r si pour tout tuple $t1, t2 \in r$, et pour tout attribut $A \in X$ tel que $t1[A] = t2[A]$ alors pour tout $A \in Y$, $t1[A] = t2[A]$. Avec RQL, les DFs sont exprimées d'une manière similaire.

Exemple 1. Q_1 découvre les DFs de *Emp* sur un sous-ensemble d'attributs.

Q_1 :
```
FINDRULES
  OVER Empno, Lastname, Workdept, Job,
       Sex, Bonus, Mgrno
  SCOPE t1, t2 Emp
  CONDITION ON $A IS t1.$A = t2.$A
```

À noter comment la clause `CONDITION` correspond à l'implication logique précédente. Cet exemple restreint aussi la découverte de DFs sur un sous-ensemble de sept attributs dans la clause `OVER`. Dans cet exemple, une couverture canonique des DFs vérifiés dans *Emp* est générée (composée de vingt-quatre DFs), dont les DFs *Empno* $\rightarrow$ *Lastname* ou *Workdept* $\rightarrow$ *Job*.

Plus précisément, une requête RQL possède la forme suivante :

```
FINDRULES
OVER [ensemble d'attributs : A1, ..., An]
SCOPE [variable de tuple : t1, ..., tn]
WHERE [condition sur (t1, ..., tn)]
CONDITION ON [variable d'attribut : $A]
  IS [condition sur ($A, t1, ..., tn)]
```

Le mot-clé `FINDRULES` identifie une requête RQL, qui génère des règles de la forme $X \rightarrow Y$ avec X et Y des ensembles disjoints d'attributs issus de la clause `OVER`. La clause `SCOPE` définit des variables de tuples sur des relations obtenues par des requêtes SQL classiques. Une clause optionnelle `WHERE` définit les relations entre variables de tuples, de manière similaire à la clause SQL `WHERE`. La clause `CONDITION ON $A` définit le prédicat devant être vérifié pour chaque attribut `$A` apparaissant dans la partie gauche et la partie droite des règles.

Pour illustrer l'expressivité des requêtes RQL, nous fournissons maintenant plusieurs exemples.

Exemple 2. Considérons les valeurs nulles (*null*), fréquentes dans les bases de données. Avec RQL, l'analyste a l'opportunité de découvrir des relations d'implication entre attributs par rapport aux valeurs nulles, en utilisant par exemple la requête Q_2.

EMP	Empno	Lastname	Workdept	Job	Educlevel	Sex	Sal	Bonus	Comm	Mgrno
	10	SPEN	C01	FINANCE	18	F	52750	500	4220	20
	20	THOMP	-	MANAGER	18	M	41250	800	3300	-
	30	KWAN	-	FINANCE	20	F	38250	500	3060	10
	50	GEYER	-	MANAGER	16	M	40175	700	3214	20
	60	STERN	D21	SALE	14	M	32250	500	2580	30
	70	PULASKI	D21	SALE	16	F	36170	700	2893	100
	90	HENDER	D21	SALE	17	F	29750	500	2380	10
	100	SPEN	C01	FINANCE	18	M	26150	800	2092	20

Figure 3: Exemple de relation

Q_2 :
```
FINDRULES
  OVER Empno, Lastname, Workdept, Job,
       Sex, Bonus, Mgrno
  SCOPE t1 Emp
  CONDITION ON $A IS t1.$A IS NULL
```

La règle *Mgrno* → *Workdept* est vérifié dans *Emp* car à chaque fois que l'attribut *Mgrno* possède une valeur nulle, alors *Workdept* est également nul pour le même tuple (seul l'employé No. 20 dans cet exemple).

À noter que la différence entre Q_1 et Q_2 provient naturellement du prédicat à évaluer, mais également du nombre de variables de tuples nécéssaires. Le prédicat de Q_1 est évalué sur des paires de tuples, tandis que Q_2 considère chaque tuple individuellement.

Exemple 3. La requête Q_1' restreint la portée de Q_1, amenant à la notion de dépendance fonctionnelle conditionnelle [5] en ne considérant que les employés avec un niveau d'éducation supérieur à 16.

Q_1' :
```
FINDRULES
  OVER Empno, Lastname, Workdept, Job,
       Sex, Bonus
  SCOPE t1, t2 (SELECT * FROM Emp
        WHERE Educlevel > 16)
  CONDITION ON $A IS t1.$A = t2.$A
```

Ici, *Sex* → *Bonus* est vérifiée avec cette restriction, ce qui signifie qu'à partir d'un certain niveau d'éducation (16), le sexe détermine le bonus.

Exemple 4. La requête Q_1'' est une approximation de Q_1 pour les valeurs numériques, de manière similaire aux dépendances fonctionnelles métriques [11], où l'égalité stricte est assouplie pour prendre en compte des variations inférieures à 10%. Par exemple, les salaires 41250 et 38250 sont considérés proches (avec une différence de 7.5%), mais pas les salaires 41250 et 36170 (différence de 13.1%).

Q_1'' :
```
FINDRULES
  OVER Educlevel, Sal, Bonus, Comm
  SCOPE t1, t2 Emp
  CONDITION ON $A IS
    2*ABS(t1.$A-t2.$A)/(t1.$A+t2.$A)<0.1
```

Dans ce cas, *Sal* → *Comm* est vérifié, ce qui signifie que les employés percevant un salaire similaire reçoivent des commissions équivalentes.

Les exemples montrés jusqu'à présent sont inspirés des implications (en FCA) et des dépendances fonctionnelles (en BD). Cependant, RQL n'est pas limité à cet type de requête et peut servir à exprimer de nombreuses autres règles.

Exemple 5. supposons que nous soyons intéressés par un type de dépendance séquentielle [9], i.e. des dépendances montrant un comportement similaire entre les attributs. Q_3 découvre des attributs numériques qui varient conjointement (i.e., $X \rightarrow Y$ signifie que si X croit, alors Y croit également).

Q_3 :
```
FINDRULES
  OVER Educlevel, Sal, Bonus, Comm
  SCOPE t1, t2 Emp
  CONDITION ON $A IS t1.$A > t2.$A
```

Sal → *Comm* et *Comm* → *Sal* sont vérifiées dans *Emp*, ce qui signifie qu'un salaire plus important correspond toujours à une commission plus élevée.

Exemple 6. En continuant l'exemple précédent, on suppose que l'analyste souhaite se concentrer sur les employés de sexe masculin.

Q_3' :
```
FINDRULES
  OVER Educlevel, Sal, Bonus, Comm
  SCOPE t1, t2 (SELECT * FROM Emp
        WHERE Sex='M')
  CONDITION ON $A IS t1.$A > t2.$A
```

Dans ce cas, *Educlevel* → *Bonus* est vérifiée, ce qui signifie que les hommes avec un niveau d'éducation plus élevé reçoivent des bonus plus importants.

Exemple 7. Au lieu de réduire la portée d'une requête, les conditions définies par l'utilisateur peuvent lier les variables de tuple entre elles par une relation ad-hoc définie dans la clause `WHERE` de la requête RQL. Par exemple, Q_4 découvre des disparités entre les managers et leurs employés, i.e. des règles sur les attributs pour lesquels les managers possèdent des valeurs supérieures ou égales à leurs employés.

Q_4 :
```
FINDRULES
  OVER Educlevel, Sal, Bonus, Comm
  SCOPE t1, t2 Emp
  WHERE t1.Empno = t2.Mgrno
  CONDITION ON $A IS t1.$A >= t2.$A
```

Dans cet exemple, $\emptyset \rightarrow$ *Bonus* est vérifiée dans *Emp*, ce qui signifie que les managers reçoivent toujours un bonus supérieur ou égal à leurs employés.

3. RETOURS AVEC CONTRE-EXEMPLES

Étant donné une requête RQL, un analyste peut – en plus de générer une couverture canonique – interagir avec le système pour vérifier si une règle est valide ou non. Il peut fournir une règle au système et deux cas sont envisageables : soit une règle est vérifiée et l'analyste est notifié que cette

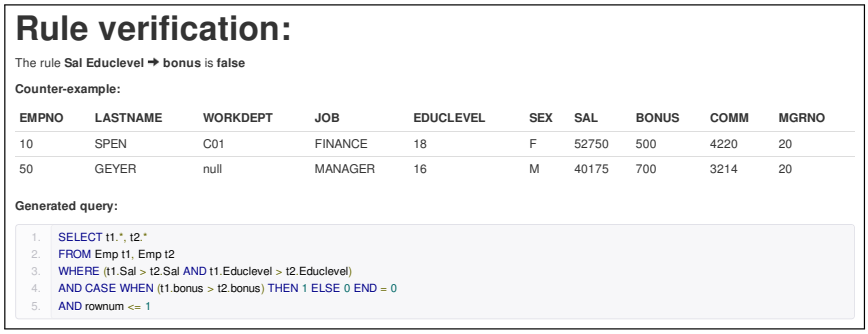


Figure 4: Génération de contre-exemples avec RQL

règle est valide, soit la règle n'est pas vérifiée ce qui signifie qu'au moins un contre-exemple existe et l'un d'eux est fourni par le système. Cette notion de contre-exemple est notamment utilisée depuis longtemps pour les dépendances fonctionnelles et les relations d'Armstrong. Ces relations exemples donnent un retour très intéressant pour l'analyste sur ses propres données (comme ce fut le cas pour la conception par l'exemple des bases de données dans les années 80). Avec RQL, le nombre de tuples requis pour fournir un contre-exemple dépend du nombre de variables de tuple de la requête. Pour Q_3, il faut au moins deux tuples, alors que pour Q_2, un tuple (par exemple l'employé No. 30) suffit pour prouver par exemple que la règle *Workdept* $\rightarrow$ *Mgrno* est fausse.

Pour illustrer les contre-exemples, supposons que l'analyste souhaite explorer son hypothèse comme quoi un salaire et un niveau d'éducation plus élevés amènent à des bonus plus importants (*Salary*, *Educlevel* $\rightarrow$ *Bonus*), en utilisant Q_3. La figure 4 donne un aperçu du retour fourni par RQL comme contre-exemple à cette règle, i.e. deux tuples pour lesquels l'un (employé No. 10) possède un salaire et un niveau d'éducation plus élevé que l'autre (employé No. 50), mais pas un bonus plus important.

Avec ce contre-exemple comme point de départ, et en particulier la requête SQL générée pour l'extraire de la base de donnée, l'analyste pour rapidement passer au langage SQL afin de comprendre pourquoi cette règle n'est pas vérifiée. Dans cet exemple, les employés qui sont soit de sexe féminin soit travaillant dans la finance sont rapidement mis en évidence comme ayant un meilleur salaire et un meilleur niveau d'éducation, mais pourtant un bonus plus bas que les autres. L'analyste peut ensuite affiner sa requête RQL, par exemple en réduisant la portée des données, comme pour Q'_3 où un meilleur niveau d'éducation suffit à entraîner un bonus plus important.

Nous sommes convaincus que les contre-exemples constituent un outil indispensable pour aider l'analyste à comprendre pourquoi une règle n'est pas vérifiée, et affiner son analyse si nécessaire en suivant un processus itératif.

4. IMPLÉMENTATION ET APPLICATION

L'application Web RQL a été implémentée en Java à l'aide du Framework Play [16]. Des outils externes ont été utilisés pour la partie la plus coûteuse du processus de génération de règles : l'énumération des transversaux minimaux d'un hypergraphe [13]. Le SGBD choisi est Oracle 11g Release 2.

L'interface fournit les requêtes SQL générées par le système aussi souvent que possible, en particulier pour identifier le (ou les) contre-exemple(s), pour que l'analyste puisse concevoir sa propre requête et en identifier davantage.

RQL peut être utilisé de deux manières : (i) une BD jouet pré-remplie est fournie avec un ensemble de requêtes pour donner un aperçu rapide du langage (ii) un mode bac à sable autorise l'utilisateur à mettre en ligne ses propres données (limité à 3 tables et 200 ko) et à formuler ses propres requêtes.

A titre expérimental, RQL a été utilisé par des étudiants (niveau L3) en marge d'un cours de bases de données à l'INSA de Lyon. L'idée était de fournir aux étudiants la possibilité de manipuler les dépendances fonctionnelles avec RQL. Sans surprise, RQL a été facilement appréhendé par les étudiants et la notion de contre-exemple a été largement mise en pratique. RQL a été apprécié par sa capacité à faire le lien entre le langage SQL et les dépendances fonctionnelles pour la conception de bases de données, et pour proposer une interface unifiée pour les requêtes SQL et RQL.

Des travaux précédent [6] ont mis en évidence l'efficacité de RQL en tant que processus à deux étapes, même sur des bases de données volumineuses.

5. CONCLUSION

RQL est introduit comme une interface Web pour découvrir des règles sur des bases de données relationnelles. RQL englobe les expressions SQL en fournissant un moyen de spécifier et d'obtenir des résultats sous la forme d'un ensemble de règles et de contre-exemples. Le problème de la fouille de motifs est vu comme un problème d'optimisation de requêtes, pour lequel nous avons proposé une technique de réécriture permettant de déléguer au maximum le traitement aux SGBD sous-jacent [6]. RQL permet aux développeurs habitués au SQL d'extraire des informations précises sans requérir des connaissances préalables en fouille de données.

6. REFERENCES

[1] M. Agier, C. Froidevaux, J.-M. Petit, Y. Renaud, and J. Wijsen. On Armstrong-compliant logical query languages. In *Proceedings of the 4th International Workshop on Logic in Databases*, LID '11, pages 33–40, 2011.

[2] M. Agier, J.-M. Petit, and E. Suzuki. Unifying framework for rule semantics : Application to gene

expression data. *Fundamenta Informaticae*, 78(4) :543–559, 2007.

[3] H. Blockeel, T. Calders, E. Fromont, B. Goethals, A. Prado, , and C. Robardet. A practical comparative study of data mining query languages. In Springer, editor, *Inductive Databases and Constraint-Based Data Mining*, pages 59–77. 2010.

[4] H. Blockeel, T. Calders, É. Fromont, B. Goethals, A. Prado, and C. Robardet. An inductive database system based on virtual mining views. *Data Min. Knowl. Discov.*, 24(1) :247–287, 2012.

[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering*, ICDE '07, pages 746–755, 2007.

[6] B. Chardin, E. Coquery, B. Gouriou, M. Pailloux, and J.-M. Petit. Query Rewriting for Rule Mining in Databases. In *Languages for Data Mining and Machine Learning, in conjunction with ECML/PKDD*, pages 35–49, 2013.

[7] L. Fang and K. LeFevre. Splash : ad-hoc querying of data and statistical models. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 275–286, 2010.

[8] B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, 1999.

[9] L. Golab, H. J. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1) :574–585, 2009.

[10] T. Guns, S. Nijssen, and L. D. Raedt. Itemset mining : A constraint programming perspective. *Artif. Intell.*, 175(12-13) :1951–1983, 2011.

[11] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1275–1278, Washington, DC, USA, 2009. IEEE Computer Society.

[12] S. Lopes, J. Petit, and L. Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*, pages 350–364, 2000.

[13] K. Murakami and T. Uno. Efficient algorithms for dualizing large-scale hypergraphs. *CoRR*, 1102.3813, 2011.

[14] A. Netz, S. Chaudhuri, J. Bernhardt, and U. M. Fayyad. Integration of data mining with database technology. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 719–722, 2000.

[15] C. Ordonez and S. K. Pitchaimalai. One-pass data mining algorithms in a DBMS with UDFs. In *SIGMOD Conference*, pages 1217–1220, 2011.

[16] Play framework. http ://www.playframework.com/.