Progetto di basi di dati integrato con ingegneria del software

A cura di:

Massimiliano Carrara

Numero di matricola: 645481

Nadia Castiglioni

Numero di matricola: 647769

Anno accademico 2002-2003
CdL: Ingegneria Informatica NO

1) Analisi della situazione reale

Descrizione delle attività del Tosem

Il TOSEM è una rivista scientifica che, in quanto tale, riceve degli articoli da studiosi del settore e si occupa della loro pubblicazione a fronte di un processo di revisione.
Tale revisione deve controllare l'originalità, il significato e l'importanza dell'argomento, la coerenza con gli argomenti della rivista e la qualità dell'esposizione.
Il processo di controllo è articolato nei seguenti passi.
Come primo passo viene inviata dal 'conctat autor' una copia dell'articolo (in formato elettronico PDF), unitamente ad un abstract e ad una lettera di presentazione, all' Editor in Cheif (EIC) che è colui che sovrintende all'intero processo. L'EIC, in funzione dell'argomento dell'articolo da esaminare, contatta un Associated Editor (AE) il quale è un esperto della particolare area e fornisce all'EIC una lista di possibili revisori per quell'articolo. Nel caso l'articolo venga immediatamente giudicato non accettabile dallo stesso AE non si assegnano i revisori, ma si procede immediatamente alla bocciatura.
In caso di giudizio positivo dell'AE, l'EIC provvede a contattare questi revisori e, se accettano di revisionare l'articolo, si negozia la data entro la quale dovrà pervenire (sempre all'EIC) la relazione. Quando l'articolo ha un numero sufficiente di revisori il processo di assegnazione si interrompe e si aspettano le revisioni. A volte la data promessa non viene rispettata e bisogna quindi gestire il processo di scambio dei messaggi. Se un revisore non risponde dopo un certo tempo può ricevere una serie di ammonizioni che possono anche comportare l'essere cacciato dal gruppo TOSEM.
Nel caso che vi sia un conflitto di interessi tra EIC e l'autore, l'EIC provvede ad inviare ad un EIC-in-ombra l'articolo in questione. Quest'ultimo è una figura analoga a quella dell'EIC e si occupa solo degli articolo con eventuali conflitti, gestendo autonomamente tutto il processo, che diviene completamente trasparente per l'EIC e se ne conosce solo l'esito finale (bocciato o pubblicato).
Quando l'EIC ha un numero sufficiente di revisioni, badando anche al parere dell'AE, prende una decisione sull'articolo. Le possibili soluzioni sono quattro.

- Articolo pubblicato: tutte le revisioni concordano, l'articolo viene così pubblicato ed esce dal processo di gestione degli articoli.
- Articolo bocciato: l'articolo non viene accettato, esce dal processo di gestione per quanto se ne tenga memoria.
- Articolo accettato con riserva: 'Major revision'. L'articolo non è accettabile e viene rispedito al mittente per delle modifiche; in questo caso l'articolo deve subire dei grandi cambiamenti e quando verrà rispedito all'EIC, egli lo considererà un articolo nuovo (assegnandogli dunque un nuovo codice identificativo) e verrà sottoposto ad un nuovo processo di revisione.
- Articolo accettato con riserva: 'Minor revision'. L'articolo è accettato ma con delle riserve che prevedono che l'articolo subisca dei leggeri cambiamenti e quando viene rispedito all'EIC non lo si considera un nuovo articolo, ma semplicemente gli si assegna una nuova versione viene poi revisionato una nuova volta.

Gli articoli con riserva, sia essa Minor o Major revision, rientrano nel processo di revisione e usciranno solo in vista della pubblicazione e del rifiuto.
Un articolo viene considerato "pubblicato" una volta che esce dalla responsabilità dell'EIC perché ha concluso il suo processo di revisione e viene inviato per la pubblicazione.

Diagrammi degli stati

Diagramma degli stati di un reviewer

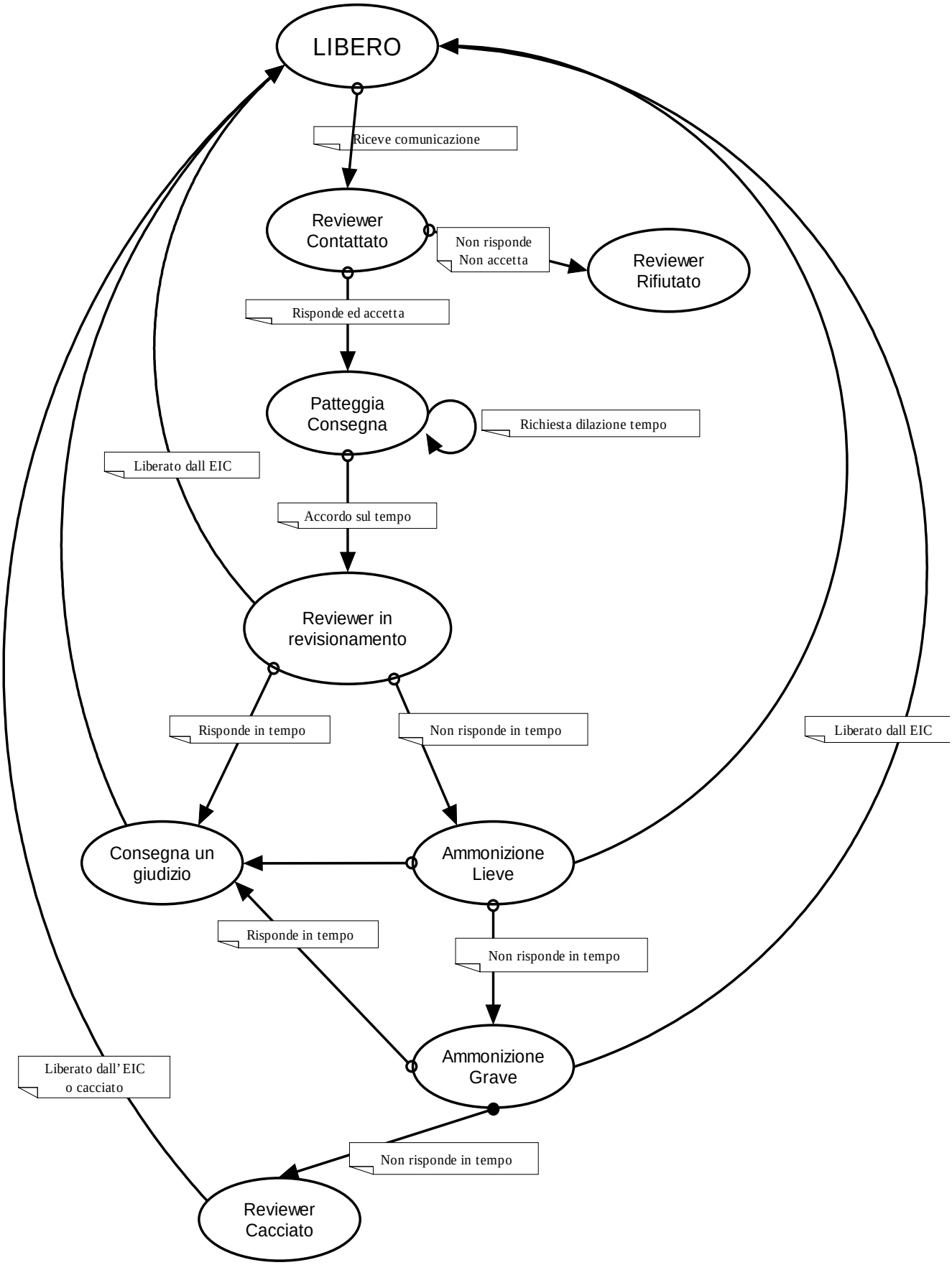
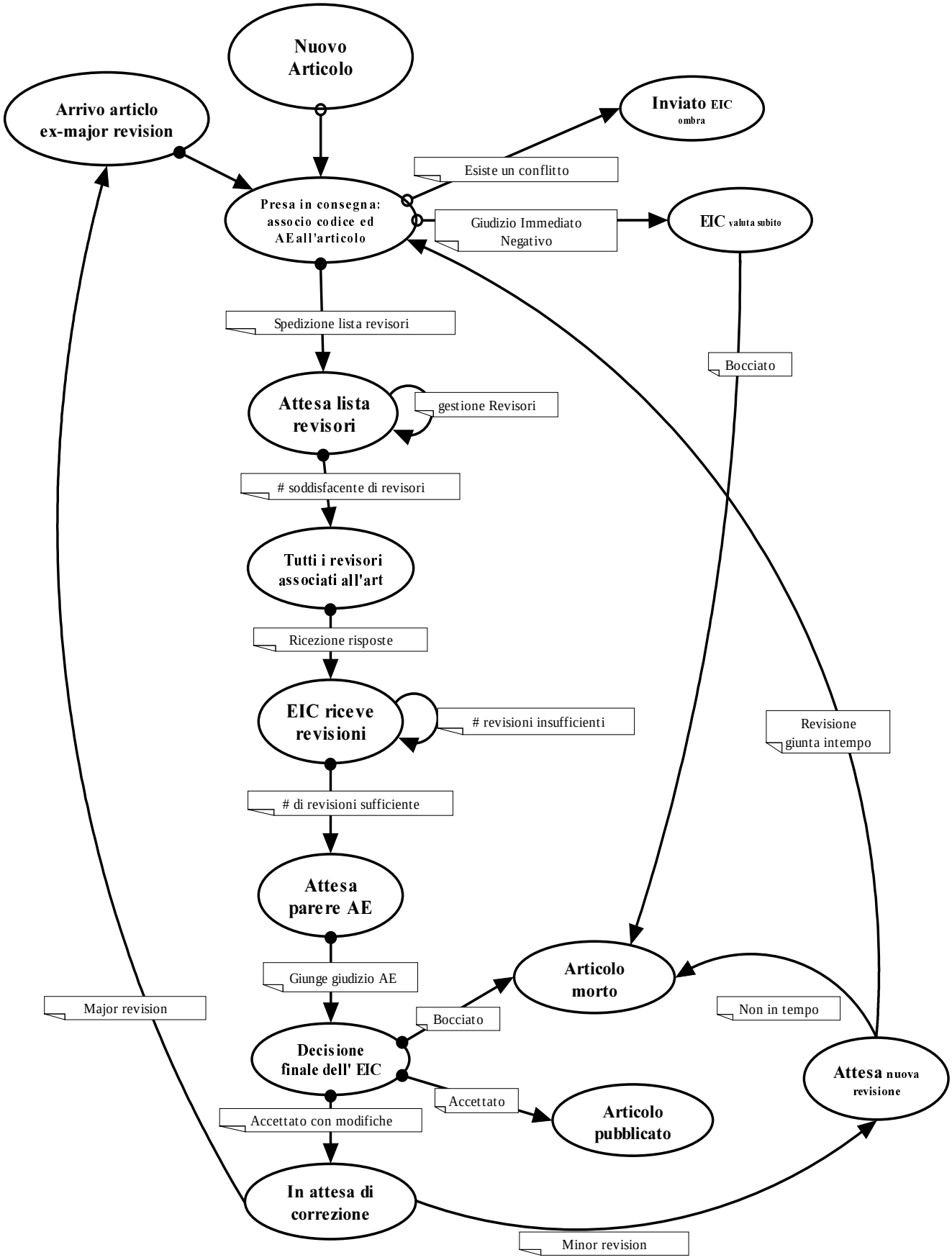


Diagramma degli stati dell'articolo



2) Progetto della base di dati

Specifiche in forma logico – verbale

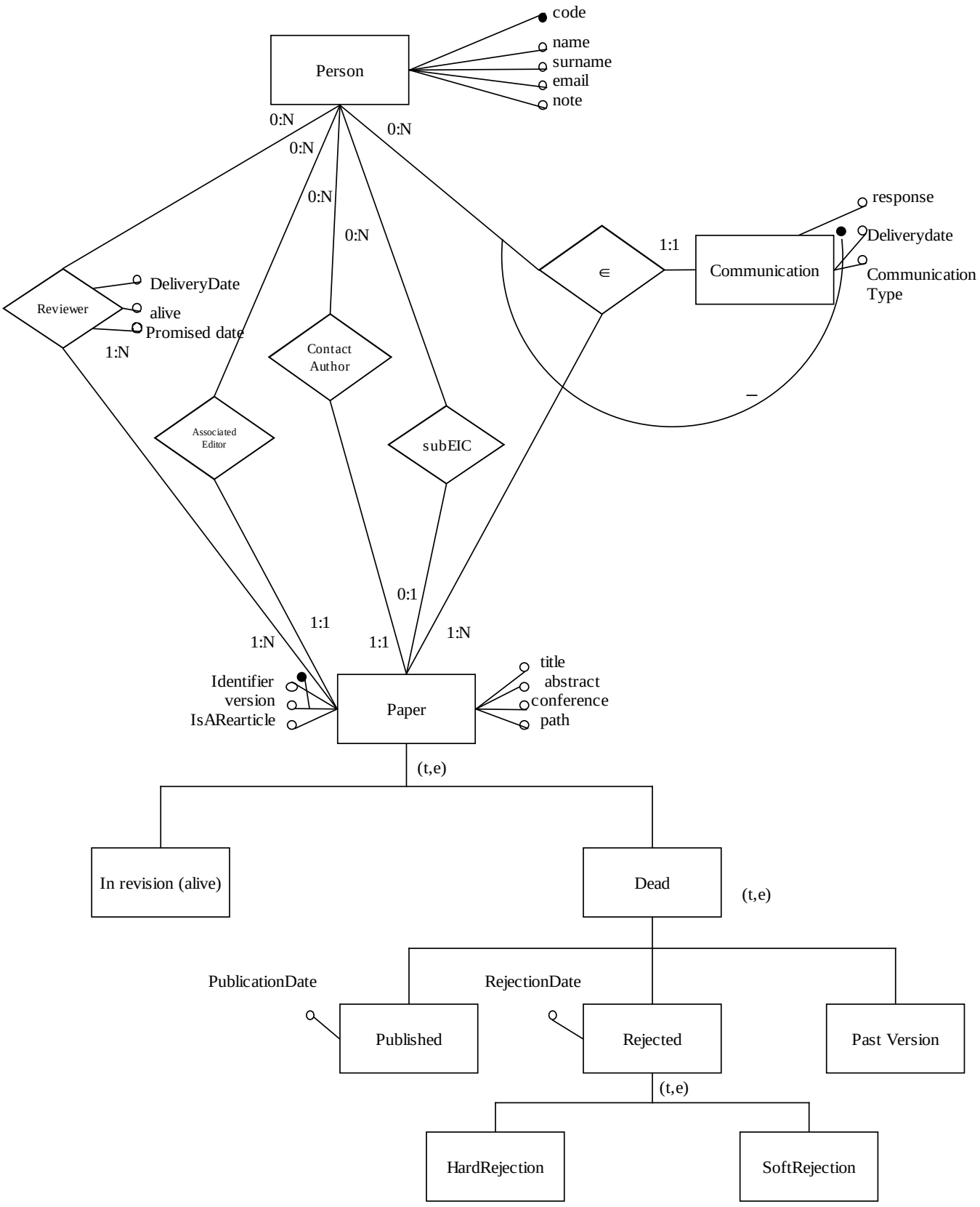
Con la realizzazione di questa base di dati si desidera gestire il processo di revisione di articoli per la rivista scientifica Tosem. Tale processo è gestito da un Editor-in-chief (EIC), eventualmente coadiuvato da un'editor assistant; egli può esaminare e gestire in ogni momento i vari stati di revisione in cui si trova l'articolo, oltre a controllare e contattare i vari studiosi che intervengono sull'articolo stesso.
Per ciascun articolo si mantiene memoria del titolo, della versione cui si fa attualmente riferimento, del path che identifica il percorso nella memoria in cui si trova l'articolo e del path dell'abstract dell'articolo stesso, eventualmente il titolo della conferenza di cui l'articolo è la rielaborazione ed un riferimento al processo di revisione precedente tale articolo si trova in Major revision.
Un articolo può essere "vivo" (alive), quando è nel processo di revisione, oppure "morto"(dead), quando esce dallo spettro di controllo dell'Editor-in-chief per la pubblicazione, o quando è stato sostituito da una nuova versione per Minor revision, oppure quando è stato rifiutato in uno qualsiasi delle fasi di revisione dell'articolo. Si desidera, infatti, mantenere traccia di tutti gli articoli scartati, sia che essi siano stati rifiutati a priori, senza essere sottoposti a revisione, sia che essi siano stati bocciati a fronte del processo di revisione, sia che necessitino di un ulteriore processo di controllo, sia esso una minor revision, oppure una major revision.
Questa classificazione in stati è totale: non è ammessa l'esistenza di articoli che non siano né in revisione, né rifiutati, né pubblicati, né sospesi. Tali stati sono esclusivi perché un articolo entra nello stato in revisione solo se non è rifiutato a priori ed esce da questo stato soltanto se subisce qualche grado di rifiuto dopo la revisione, oppure se viene pubblicato o sostituito da una nuova versione. Gli articoli pubblicati e rifiutati sono contraddistinti dalla data di pubblicazione e di rifiuto rispettivamente.
La bocciatura subita da un articolo può presentare diversi gradi di rifiuto; a tal proposito, è opportuno distinguere fra gli articoli definitivamente rifiutati e quindi senza più alcuna speranza di revisione e pubblicazione, e quelli cui viene assegnato un nuovo codice per subire una MajorRevision, ossia entrare nuovamente, dopo aver subito sostanziali modifiche, in un processo di revisione. La bocciatura subita in questo caso è percepita come lieve perché, seppure in altra forma, questi articoli hanno ancora speranza di pubblicazione.
Anche per quanto riguarda i gradi di rifiuto di un articolo, la distinzione è totale e non ammette sovrapposizioni: non esistono altri gradi di rifiuto rispetto a quelli indicati e fra di essi non ci sono articoli condivisi, poiché un articolo rifiutato completamente non ha nuovamente accesso al processo di revisione ed un articolo che figura fra quelli che subiscono una Major Revision, non può essere considerato completamente rifiutato.
Sull'articolo intervengono nelle diverse fasi di revisione gli studiosi.
Per tutti gli studiosi è sufficiente mantenere memoria dei dati anagrafici essenziali, come nome, cognome e email per poterli contattare con l'aggiunta di un campo per eventuali annotazioni.
Uno studioso può assumere diversi ruoli nei confronti dell'articolo: può esserne l'autore, può venirgli assegnato come associate editor, oppure come reviewer, oppure può esserne l'EIC-ombra.
Ad ogni articolo sono associati almeno un autore, almeno un reviewer ed uno ed un solo associated editor; non si esclude che gli autori e i reviewers siano più d'uno.
Gli studiosi possono gestire da 0 a N articoli nei vari ruoli; non si esclude che la medesima persona possa ricoprire più di un ruolo per articoli diversi.

Per quanto concerne uno studioso che interviene sull'articolo per un processo di revisione in qualità di reviewer, si desidera mantenere memoria di tutte le revisioni avvenute, sia nel passato, sia attualmente in svolgimento. Per ogni revisore è opportuno ricordare la data in cui ha ricevuto l'articolo e l'ultima data per la quale ha promesso di ultimare il lavoro.
Tuttavia ci può essere stata più di una promessa riguardante la data di consegna e si desidera poter ricostruire tutta la "storia" delle promesse fatte da ciascun revisore per ciascun articolo.

L'evoluzione del processo di revisione avviene mediante una serie di comunicazioni fra l'EIC e gli studiosi, quando questi ricoprono il ruolo di Associate editor oppure di reviewer.

Infatti, l'EIC può richiedere ad un AE una lista di possibili revisori e, a revisioni ultimate, un parere sull'articolo. Al contrario, le comunicazioni intercorse fra i revisori e l'EIC possono riguardare la richiesta di revisione, il patteggiamento della data di consegna, i solleciti a svolgere il lavoro e i ringraziamenti. E' opportuno mantenere memoria di tali comunicazioni, comprensive della data di emissione e del genere di comunicazione inoltrata.
Per ogni articolo ci deve essere almeno una comunicazione uno studioso (la richiesta di lista di reviewers), ma ce ne possono essere in numero indefinito. Possono essere intercorse diverse mails fra l'EIC e uno studioso, ma non è necessario che tutti gli studiosi abbiano avuto comunicazioni. Infatti si tiene traccia solo si quelle intercorse con studiosi nel ruolo di associate editor oppure di reviewer. Ogni comunicazione ha associato uno ed un solo studioso e riguarda uno ed un solo articolo.
In alcuni rari casi si verifica una situazione di conflitto di interesse, ad esempio quando l'autore dell'articolo è un collaboratore dell'EIC; in questo caso l'articolo viene inviato ad uno studioso, detto Editor-in-chief ombra (subEIC), che gestisce autonomamente tutto il processo di revisione. Per tali articoli, è sufficiente mantenere traccia della loro esistenza e ricordare che la revisione è gestita dall'EIC ombra. Infatti, non si associano a tale articolo né revisori, né associated editor, in quanto questi vengono decisi dall'EIC in ombra in maniera completamente indipendente.

Schema entità – relazione



Commenti allo schema entità – relazione: il dizionario dei dati

Tabella delle entità

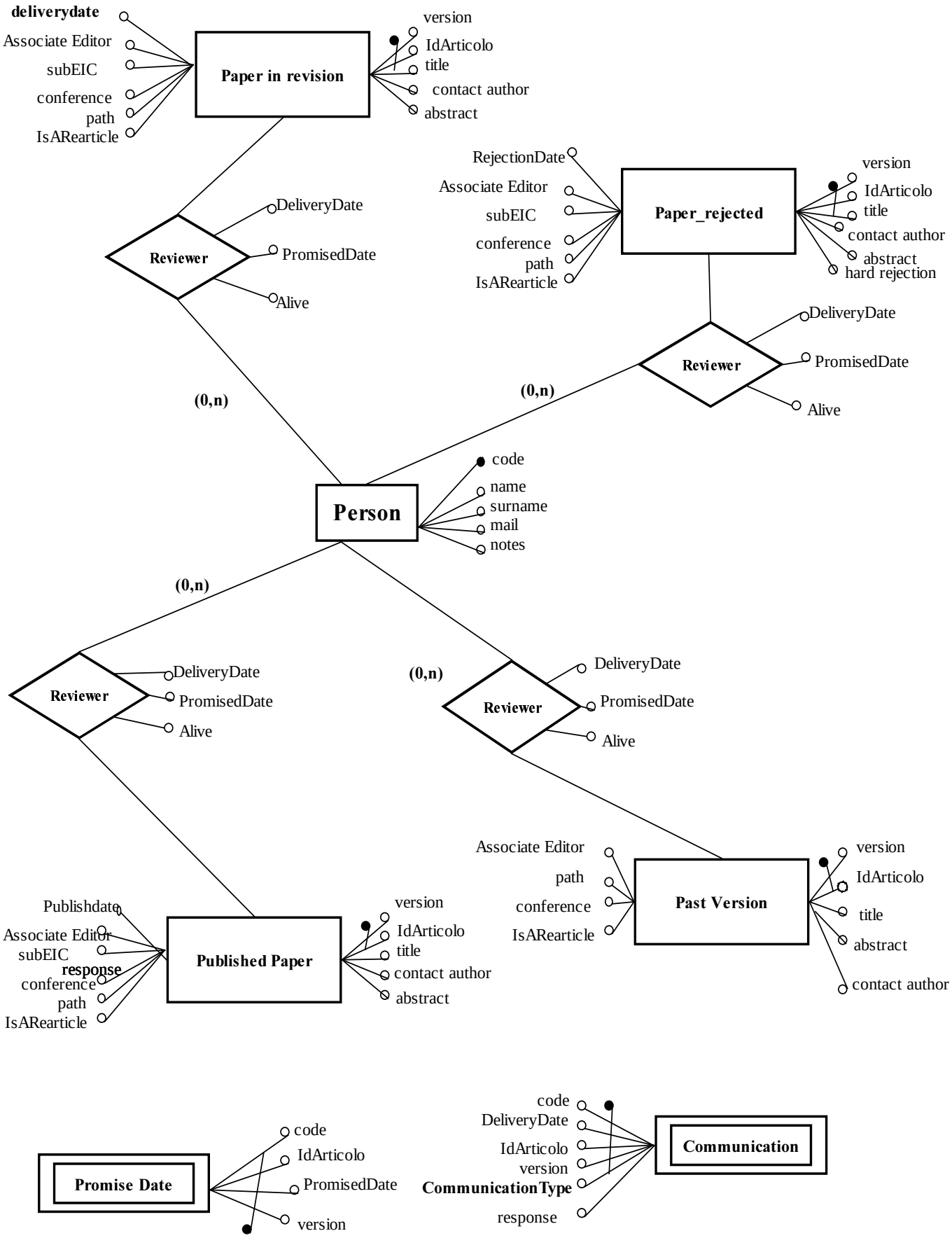
ENTITA'	DESCRIZIONE	ATTRIBUTI	IDENTIFICATORE
PERSON	Entità studioso, che può intervenire sull'articolo con diverse modalità	Name, surname, email, code	Code
PAPER	Articolo, l'oggetto del processo di revisione	Identifier, version, title, abstract (path), conference, path (of the body), isARearticle (indica l'eventuale identifier del primo processo di revisione subito)	Identifier insieme con version
PAPER ALIVE	Articolo attualmente in revisione.	Quelli ereditati da Paper,	Identifier insieme con version ereditati da Paper
PAPER DEAD	Articolo uscito dalla responsabilità dell'EIC per la pubblicazione o per il rifiuto. E' un sottotipo di Paper	Quelli ereditati da Paper	Identifier insieme con version ereditati da Paper
PUBLISHED PAPER	Articolo ormai completamente revisionato, pronto per la pubblicazione.E' un sottotipo di Dead Paper	Quelli ereditati da Paper, Publicationdate	Identifier insieme con version ereditati da Paper
PAST VERSION	Versioni degli articoli ormai sostituite da nuove versioni a fronte di minor revision. E' sottotipo di Dead Paper.	Quelli ereditati da Paper	Identifier insieme con version, ereditati da Paper

REJECTED PAPER	Articolo rifutato in un qualsiasi stadio del processo di revisione. E' sottotipo di Dead paper.	Quelli ereditati da Paper, RejectionDate	Identifier insieme con version ereditati da Paper
HARD REJECTION	Articolo definitivamente rifiutato. E' sottotipo di Rejected Paper.	Quelli ereditati da Rejected Paper.	Identifier insieme con version ereditati da Paper
SOFT REJECTION	Articolo rifiutato per sottoporlo ad una Major Revision. Esso rientra nel processo di revisione con altro codice e quindi il rifiuto subito è di grado lieve.	Quelli ereditati da Rejected Paper.	Identifier insieme con version ereditati da Paper
COMMUNICATION	Email spedita oppure ricevuta dall'EIC.	Deliverydate, data di spedizione/ricezione della mail, communication type, campo stringa che indica l'oggetto della mail, person, che identifica se il mittente/destinatario è un AE oppure un RW, response, che indica se la comunicazione è pendente.	L'identificativo dell'articolo insieme alla sua versione, il codice dello studioso, la data e il tipo di comunicazione.

Tabella delle relazioni

RELAZIONE	DESCRIZIONE	ENTITA' COINVOLTE	ATTRIBUTI
ASSOCIATED EDITOR	Associa a ciascun articolo uno studioso in veste di editore, che fornisce i nominativi dei possibili revisori	Person (0,N) Paper (1,1)	
REVIEWER	Associa a ciascun articolo almeno un revisore.	Person (0,N) Paper (1:N)	Delivery date (data di consegna dell'articolo), Promised date (date promessa per la revisione) Alive(flag che identifica se il processo di revisione è attualmente in corso per quel revisore)
CONTACT AUTHOR	Associa a ciascun articolo un autore di riferimento per le comunicazioni.	Person (0,N) Paper (0,1)	
SUB EIC	Associa ad un articolo l'eventuale EIC ombra che ne gestisce il processo di revisione.	Person (0,N) Paper (0,1)	
COMMUNICATIONə PERSON ə PAPER	Tiene memoria delle comunicazioni intercorse tra l'EIC e l'associated editor o i reviewers di un articolo.	Person(0,N) Paper (1,N) Communication (1,1)	

Schema entità – relazione ristrutturato



Vincoli fra le tabelle

- **Paper_revision, Published paper, Rejected paper, Past Version:**
  - Esiste un vincolo d'integrità referenziale fra i campi "contact_author", "associatedEditor", "subEIC" delle tabelle "paper in revision" , "published_paper" , "rejected paper", "past version" e il campo "code" della tabella "person", poiché solo gli studiosi registrati nella base di dati possono ricoprire questi ruoli per un articolo;
  - Esiste un vincolo di integrita' referenziale fra il campo "IsARearticle" delle tabelle "paper in revision" , "published_paper" , "rejected paper", "past version" e il campo "identifier" della tabella "Rejected Paper". Infatti "IsARearticle" contiene il codice dell'articolo nella sua eventuale revisione precedente. Più precisamente, IsARearticle si riferisce a quegli articoli della tabella "RejectedPaper" per cui il flag "HardRejection" è settato a false, cioè quelli in stato di bocciatura non grave.
  - *Nonostante le gerarchie siano collassate verso il basso e ciascun sottotipo degli articoli abbia il proprio identifier, onde evitare confusione fra gli stati inconciliabili dell'articolo, i codici di identificazione con le relative versioni devono rimanere univoci non solo all'interno della medesima tabella, ma anche fra le quattro tabelle "paper revision","past version", "rejected paper", "published paper".*
- **Person:**
  - *E' opportuno controllare mediante il campo "code" della tabella person, che un medesimo studioso non stia ricoprendo diversi ruoli nei confronti del medesimo articolo: l'autore non può essere il reviewer del medesimo articolo e nemmeno l'associated editor o il subEIC, ruoli fra cui non può esserci sovrapposizione.*
- **Review:**
  - Il campo code si riferisce al campo omonimo della tabella "Person", in quanto solo uno studioso registrato nella base di dati può assumere il ruolo di reviewer;
  - Esiste un vincolo di integrità referenziale fra il campo Promisedate della tabella "reviewers" e il campo Promisedate della tabella "Promisedate", in quanto nella tabella "reviewers" viene riportata l'ultima promessa fatta riguardo la data di consegna della revisione;
  - *Sussiste un vincolo di integrità referenziale fra le tabelle "paper in revision", "published_paper", "rejected_paper", "past version" e la tabella "review", in quanto solo agli articoli registrati in queste tabelle possono essere (stati) associati dei reviewers. Gli attributi di integrita' referenziale sono l'identifier dell'articolo insieme alla sua versione. Più precisamente, tali campi si riferiscono agli omonimi della tabella "paper revision" se il flag "alive" è settato a true, altrimenti si riferiscono ad una ed una soltanto delle altre tre tabelle.*
- **PromisedDate:**
  - Esiste un vincolo di integrità referenziale fra i campi"version", "ID", "code" della tabella "promised Date" e i campi "version", "ID", "code" della tabella "reviewers"; infatti le date promesse di revisione hanno senso solo in relazione agli articoli in revisione.
- **Communication:**
  - Il campo "code" di questa tabelle si riferisce al campo "code" della tabella "person", poiché gli AE e i reviewer con i quali si comunica sono studiosi registrati nella base di dati;
  - *Il vincolo di integrità referenziale fra i campi ID e version della tabella "communication" sussiste con i campi omonimi della tabella "paper_revision", se il flag alive è "true"; in caso contrario, tali campi sono riferiti agli attributi omonimi delle tabelle "rejected_paper", "past_version", "published_paper";*

Note

Le due entità principali dello schema entità – relazione sono l'articolo e lo studioso; essi sono messi in relazione dai ruoli che il medesimo studioso può assumere nei confronti di scritti diversi.
Si è scelto di rappresentare tali ruoli con delle relazioni e non con una gerarchia per motivi di carattere implementativo.
Infatti, è utile hai fini dell'applicazione mantenere un'indirizzario con le informazioni rilevanti di ciascuna persona, a prescindere dal fatto che sia attualmente impegnata in uno qualsiasi dei ruoli del processo di revisione. Tuttavia è altrettanto indispensabile mantenere schemi relazionali distinti per gli articoli attualmente in revisione e per identificare i vari ruoli ricoperti.
Se si fossero rappresentati questi ruoli con una gerarchia, essa sarebbe stata di tipo parziale, perché non si mantiene nella base di dati solamente gli studiosi impegnati in qualche processo, e di tipo overlapping, perché la medesima persona può ricoprire ruoli diversi per articoli diversi; quindi, tale gerarchia avrebbe dovuto collassate verso l'alto, producendo una tabella con numero considerevole di campi settati a null, poiché uno studioso ricopre un numero limitato di ruoli e, articolo per articolo, ne ricopre uno soltanto.
Le relazioni "ContactAuthor" ed "AssociatedEditor", sono di tipo 1 a molti, perciò si è deciso di includerle come attributi dei Paper perché comunque non opzionali ma obbligatori (eccezion fatta per gli articoli affidati al sub-EIC, che costituiscono un caso a sé stante per cui questi attributi non hanno valore, sono ignoti).
Anche subEIC è una relazione uno a molti, ma non obbligatoria; infatti, solo alcuni articoli saranno gestiti con questo processo alternativo e dunque si è mantenuto un flag in Paper che indicasse se tale processo è gestito in maniera indipendente rispetto agli altri. Poiché il processo di revisione è completamente trasparente all'EIC effettivo, per questo genere di articoli, comunque, i campi inerenti all'associated editor e ai reviewers saranno nulli.
La relazione "reviewer" è di tipo molti a molti e perciò genera una propria tabella, distinta da quelle di "Paper" e di "Person". Essa registra non solo gli articoli attualmente in revisione, ma anche tutte le associazioni revisori –articoli che sono avvenute nel passato.
Per questa ragione, si è scelto di mantenere un flag di nome "alive" fra gli attributi, in modo da indicare se il revisore ha già ultimato il suo processo di controllo. Infatti, un articolo permane nello stato di revisione fino a al momento in cui tutte le revisioni non sono state ultimate, ma ciascuno studioso conclude la propria attività in momenti indipendenti da quelli degli altri revisori e per questo è bene mantenere memoria della situazione individuale di ciascuno.
La relazione "reviewer" presenta l'attributo multivalore "promisedDate" e, poichè si è scelto di mantenere traccia della storia delle date promesse, questo attributo multivalore è stato tradotto in un'apposito schema relazionale, rappresentato da un'entità debole nello schema entità relazioni ristrutturato.
Si è comunque scelto di mantenere un attributo "promiseddate" in "reviewer" per tenere un accesso facile all'ultima data promessa cui si deve far riferimento.
Poichè si desidera mantenere memoria di tutte le revisioni avvenute e non soltanto di quelle in corso di svolgimento, lo schema entità – relazioni ristrutturato presenta 4 volte la relazione "reviewer", una per ciascuno dei possibili stati dell'articolo: in tale relazione, un reviewer può essere associato ad articoli i cui codici figurano già nei pubblicati oppure nei rifiutati, in quanto il loro processo di revisione è ormai concluso.
In realtà, lo schema logico dimostra che questa scelta è puramente di carattere grafico, in quanto non dà origine a quattro schemi relazionali distinti, ma viene mantenuta un'unica tabella che opera la distinzione mediante il flag "alive".
Al contrario di ciò che era avvenuto per i ruoli degli studiosi e l'entità "person", si è deciso di mantenere distinte le gerarchie fra gli articoli in revisione e quelli pubblicati, rifiutati e sospesi, proprio per mantenere memoria dei processi passati, distinguendo gli work-in-progress da quelli che già hanno ultimato il loro processo.

Infatti queste gerarchie sono totali, perché non esistono articoli che non siano né in revisione, né pubblicati, né rifiutati, né sostituiti da nuove versioni, ed esclusive, perché è inammissibile la sovrapposizione fra questi stati: un articolo entra nello stato di revisione solo se non è rifiutato a priori e ne esce solo per una bocciatura a fronte della revisione o per la pubblicazione o per la sostituzione mediante una nuova versione.
Dal momento che questa gerarchia è totale ed esclusiva, essa collassa verso il basso, producendo quattro schemi relazionali differenti e la replicazione degli attributi della tabella padre in tutte le tabelle figlie. Questa scelta non produce un esubero di cambi settati a null, perché tali attributi sono obbligatori, eccezion fatta per articoli affidati al subEIC, i quali possono comunque essere considerati una rarità.
Per le passate versioni, non esiste un campo subEIC perché gli articoli che attraversano questo particolare processo non possono avere associati né revisori e nemmeno versioni; se esistessero, di sicuro non sarebbero visibili all'EIC effettivo.
La sottogerarchia degli articoli rifiutati, ovverosia la distinzione fra articoli che hanno subito un rifiuto definitivo da quelli che ne hanno subito uno lieve, è anch'essa totale, in quanto non esistono altri possibili gradi di rifiuto, ed esclusiva, poiché un articolo definitivamente rifiutato non rientra neppure con altro codice nel processo di revisione e, se rientrasse in tale processo, non figurerebbe ad alto grado di rifiuto.
Nonostante presenti questo genere di caratteristiche, la gerarchia è collassata verso l'alto in un flag della tabella "Rejected Paper" poiché ai fini implementativi non è necessario mantenere due schemi relazionali distinti, ma è bene solamente ricordare l'esistenza di questa sottile differenza fra i gradi di bocciatura, differenza che è utile nel periodo in cui l'articolo attende di poter essere reinserito nel processo di controllo per major Revision, per identificare che la bocciatura subita non era definitiva.
Effettivamente anche gli articoli cui si assegna una nuova versione hanno subito un qualche grado di rifiuto, poichè anch'essi dovranno rientrare nel processo di revisione dopo aver subito qualche modifica. Tuttavia i cambiamenti che saranno apportati a questi articoli sono minimi ed è per questo che rientrano nel processo di revisione con il medesimo codice e l'aggiornamento della sola versione. Al contrario, gli articoli che subiscono una major revision subiscono sostanziali modifiche ed è per questo che rientrano nel processo con un nuovo codice: essi possono a ragione essere considerati degli articoli differenti dai precedenti.
Dunque si è deciso di distinguere le versioni precedenti (past_version) degli articoli rifiutati per bocciatura o per major revision perché la riassegnazione del codice all'articolo indica una modificha sostanziale.
Inoltre, se si fossero raggruppati tutti i possibili gradi di rifuto, si sarebbe ottenuto un nuovo livello nella gerarchia di paper (le past_versions sarebbero sottotipo di soft_rejection insieme alle major_revision), producendo un schema eccessivamente complesso rispetto alla situazione da rappresentare.
L'ultima entità dello schema ER riguarda le comunicazioni. Esse sono contraddistinte dalla data di spedizione (o di ricezione), da un campo che indica il tipo, ovverosia l'oggetto della mail, e dall'attributo person, che distingue se la relazione è avvenuta con un associate editor oppure con un reviewer. Infatti, le comunicazioni memorizzate possono avvenire soltanto con studiosi che ricoprono uno di questi ruoli ed è bene mantenere questa distinzione per visualizzare in maniera più coerente le mail. Le comunicazioni si collegano con una relazione ternaria sia allo studioso, sia all'articolo, poiché è doveroso sapere chi sia che ha spedito/ricevuto la mail e quale articolo riguardasse. Naturalmente non tutti gli studiosi sono associati a comunicazioni per questioni di ruolo, come pure a un articolo può non essere associata alcuna comunicazione quando viene bocciato a priori dall'EIC. Ciascuna comunicazione riguarda uno ed un solo articolo e avviene con uno ed un solo studioso. Per questo, la relazione ternaria che collegava queste tre entità viene inclusa in "communication", che è un'entità debole perché necessita nella sua chiave primaria anche del codice dello studioso e dell'identifier dell'articolo.

Progetto Logico

- **person(<u>code</u> , name , surname , email, notes)**
- **paper_revision(<u>ID</u> , <u>version</u>, title, abstract, conference, contact_author, associateEditor, subEIC, deliverydate, rearticle, path)**
- **paper_published(<u>ID</u> , <u>version</u> , title, abstract, conference, contact_author, associateEditor, subEIC, publisheddate, rearticle, path)**
- **paper_rejected(<u>ID</u> , <u>version</u> , title, abstract, conference, contact_author, associateEditor, subEIC, rejectiondate, rearticle, path, hard_rejection)**
- **past_version (<u>ID</u> , <u>version</u>, title, abstract, conference, contact_author, associateEditor, rejectiondate, rearticle, path)**
- **reviewer (<u>code</u> , <u>ID</u> , <u>version</u>, deliverydate, promisedate, alive)**
- **promise_date(<u>code</u> , <u>ID</u> , <u>version</u> , <u>promisedate</u>)**
- **communication (<u>code</u>, <u>ID</u>, <u>version</u>, <u>deliverydate</u>, <u>communicationtype</u>, person, response)**

Implementazione in MySQL

```
create table person
(
        code numeric(6) primary key,
        name character(20),
        surname character(30),
        email character(50),
        notes character(200),
        unique (name, surname, email)
);
```

*Nota alla tabella **person**:*
I soli campi ' nome, cognome, email ' avrebbero potuto anche costituire la chiave primaria, ma per motivi di efficienza abbiamo preferito inserirvi un codiceID.
Tuttavia abbiamo ritenuto significativo mettere il vincolo " unique" su questi attributi in modo da proteggerci dal caso, neppure assai improbabile, nel quale si tenta di inserire una persona che è già presente nel DB. Se si verifica questo caso il DBMS verifica una violazione nei vincoli di integrità ed impedisce l'inserimento del nuovo record. Verrà poi gestito via SW il problema di avvisare l'utente dell'errore causato.

```
create table paper_revision
(
        ID numeric(6) not null,
        version numeric(3) not null,
        title character(100) not null ,
        abstract character(255) not null ,
        conference character(100) ,
        contact_author numeric(6) not null  references person(code)
                                                        on delete cascade
                                                        on update no action ,
        associateEditor numeric(6) not null references person(code)
                                                         on delete cascade
                                                         on update no action ,
        subEIC numeric(6) default null references person(code)
                                                        on delete cascade
                                                        on update no action ,
        deliverydate date,
        rearticle numeric(6) references paper_rejected(ID),
        path character(255) not null,
        primary key ( ID, version )
);



create table paper_published
(
        ID numeric(6) not null,
        version numeric(3) not null,
        title character(100) not null ,
        abstract character(255) not null ,
        conference character(100) ,
        contact_author numeric(6) not null  references person(code)
                                                        on delete cascade
                                                        on update no action ,
        associateEditor numeric(6) not null references person(code)
                                                       on delete cascade
                                                        on update no action ,
        subEIC numeric(6) default null references person(code)
                                                        on delete cascade
                                                        on update no action ,
        publisheddate date ,
        rearticle numeric(6) references paper_rejection(ID),
        path character(255) not null,
        primary key ( ID, version )
);
```

```
create table paper_rejected
(       ID numeric(6) not null,
        version numeric(3) not null,
        title character(100) not null ,
        abstract character(255) not null ,
        conference character(100) ,
        contact_author numeric(6) not null  references person(code)
                                                        on delete cascade
                                                        on update no action ,
        associateEditor numeric(6) not null references person(code)
                                                        on delete cascade
                                                        on update no action ,
        subEIC numeric(6) default null references person(code)
                                                        on delete cascade
                                                        on update no action ,
        rejectiondate date ,
        rearticle numeric(6) references paper_rejection(ID),
        path character(255) not null,
        hard_rejection character(3),
        primary key ( ID, version )
);
create table past_version
(       ID numeric(6) not null,
        version numeric(3) not null,
        title character(100) not null ,
        abstract character(255) not null ,
        conference character(100) ,
        contact_author numeric(6) not null  references person(code)
                                                        on delete cascade
                                                        on update no action ,
        associateEditor numeric(6) not null references person(code)
                                                        on delete cascade
                                                        on update no action ,
        rejectiondate date ,
        rearticle numeric(6) references paper_rejection(ID),
        path character(255) not null,
        primary key ( ID, version )
);
```

*Nota alle tabelle **paper_revision, paper_rejected, paper_published**:*
alla politica di reazione 'on update' abbiamo deciso di associare 'no action' (cioè di impedire la modifica) in quanto sarebbe priva di senso la modifica del codice di uno studioso: al più è ammissibile la rimozione (Ricordiamo che se cambiasse il suo ruolo il suo codice deve restare immutato).
I campi ID e version di tali tabelle devono rimanere univoci non solamente all'interno del medesimo schema relazionale, ma anche fra tutte e quattro le tabelle.
Dal momento che la versione di Mysql da noi utilizzata non supporta l'uso delle assertion, costrutti mediante cui sarebbe possibile gestire vincoli di questo tipo, il controllo di questo vincolo verrà gestito mediante software di interfacciamento.
*Notiamo infine come la tabella **Past_version** serva solo per mantenere le versioni passate di un articolo e da qui non verranno mai cancellati degli elementi. IN questa tabella non compare il campo subEIC poiché non è possibile visionare eventuali versioni passate di articoli il cui processo di revisione è perfettamente trasparente per l'EIC.*

```
create table reviewer
(
        code numeric(6) not null  references person(code)
                                                on delete cascade
                                                on update no action ,
        ID numeric(6) not null ,
        version numeric(3) not null ,
        deliverydate date,
        promisedate date not null references promise_date(promisedate)
                                                on delete cascade
                                                on update no action,
        alive char(3) default 'Y' ,
        primary key (code, ID, version)
);
```

*Note alla tabella **review**:*

Notiamo come grazie all'attributo 'on delete cascade' sul vincolo 'code' si ha la possibilità, all'eliminazione di un revisore dalla base di dati, di cancellare anche gli articoli che ha revisionato (o che sta revisionando) in quanto non avrebbe senso mantenere traccia di questo revisore se esso è stato eliminato dalla tabella degli studiosi.

*Il campo "promisedate" contiene l'ultima data di consegna promessa dal revisore. Questo campo contiene le stesse informazioni presenti nell'ultima tupla (relativa ad i dati di un revisore) della tabella **promise_date**. Questa replicazione di dati risulta essere indispensabile, altrimenti si avrebbe una complessità delle query non gestibile con MySQL (infatti le query nidificate non ssono ancora ben supportate se non ai livelli più semplici; un maggiore supporto si dovrebbe avere nella versione 4.1 che in questo momento è in alfatest).*

Notiamo che per mantenere aggiornato il campo in questione ci basta sovrascrivere il seguente campo ogni volta che perviene una nuova data promessa.

In realtà ID e Version dell'articolo si riferiscono ad articoli realmente presenti nella nella base di dati, ma poiché nella tabella "review" si mantiene informazione anche di delle revisioni ormai ultimate, questi campi possono far riferimento agli attributi omonimi sia della tabella paper_revision, sia della tabella "paper_rejected", sia della tabella "paper_published", o "past_version". Anche questo genere di vincolo non è gestibile con la versione di MySQL disponibile e perciò verrà rimandata la sua gestione all'implementazione dell'interfaccia software.

```
create table  promise_date
(       code numeric(6) not null  ,
        ID numeric(6) not null,
        version numeric(3) not null ,
        promisedate date not null,
        primary key ( code, ID, version, promisedate),
        foreign key (code, ID, version) references reviewer(code, ID, version)
                                                on delete cascade
                                                on update no action
);
```

*Note alla tabella **promise_date**:*

Questa tabella serve essenzialmente per implementare l'attributo multivalore promisedate. Infatti i Database relazionali ammettono solo attributi monovalore.

Abbiamo messo un vincolo di integrità referenziale fra i campi code, ID e version di questa tabella e i campi omonimi della tabella reviewer in modo che solo per articoli e versioni effettivamente in revisione si possa avere la serie delle date promesse.

```
create table communication(
        code numeric(6) not null  references person(code),
        ID numeric(6) not null,
        version numeric(3) not null ,
        deliverydate date not null,
        communicationType char(255) not null,
        person char(5) not null,
        primary key ( code, ID, version, deliverydate,communicationType)
) ;
```

*Note alle tabelle **communication**:*
I campi ID e version che identificano l'articolo dovrebbero avere un vincolo di integrità referenziale con gli schemi relazionali che tengono traccia delle informazioni relative agli articoli; tuttavia tale vincolo non è stato espresso poiché tali schemi relazionali tengono memoria di tutti gli articoli, a prescindere dallo stato di revisione in cui si trovano. Questo implicherebbe che i campi suddetti possano riferirsi o alla tabella paper_revision, oppure alla tabella rejected_paper, oppure alla tabella published_paper . Questo genere di integrità referenziale estesa fra molteplici tabelle non è gestibile mediante Mysql; sarà gestita mediante software di interfacciamento.

Note sulla tipizzazione in MySQL

1. Nella versione di linguaggio da noi utilizzata, non esiste il tipo booleano. Le tabelle risulterebbero decisamente più chiare e leggibili con un simile tipo per l costruzione dei flag e quindi avremmo voluto crearlo, ma il nostro DBMS purtroppo non permette nemmeno la creazione di nuovi tipi.
   Facciamo notare come non sia neanche presente un tipo BIT o perlomeno è presente ma non ha dei vincoli di integrità: si può tranquillamente assegnargli qualsiasi valore numerico). Dunque useremo semplicemente delle stringhe di testo per creare questo tipo.

2. Nota ai tipi degli attributo 'abstract' e 'percorso' : abbiamo deciso di implementarli con un semplice tipo stringa di 255 caratteri (lunghezza massima consentita dal DBMS per una stringa). Nel caso ciò non fosse sufficiente si potrebbe usare il tipo BLOB che in mySQL contiene una stringa di 2^16-1 caratteri (se non fosse sufficiente c'è anche il LARGE BLOB che conta 2^32 – 1 caratteri..).

3. Nota al tipo date: in mySQL i valori sono del tipo 'YYYY-MM-DD' e per l'inserimento è anche abbastanza elastico in quanto accetta l'inserimento di stringhe con comandi come:
   insert into prova1 values('03/02/25')
   con la nota che non è necessario specificare le prime due cifre dell'anno in quanto si assume che siano appartenenti all'intorno dell'anno dove risiede il programma (es:se si inserisce '03' assume sia del XXI secolo, mentre se si inserisce '99' assume sia relativo al Xxsec). Se si vuole inserire altre date di altri secoli va specificato completamente l'anno; es: '1899-02-25'

Complementi al sistema

Come naturale complemento al nostro DB appena creato sarebbe opportuno introdurre una serie di asserzioni per esprimere ulteriori vincoli che non sono presenti nella creazione delle tabelle.
Però c'è l'insormontabile problema che mySQL non ha un costrutto per le asserzioni: dunque non le implementeremo ma le presenteremo lo stesso al fine di dettagliare ulteriormente il nostro DB.
Per cominciare presentiamo delle asserzioni che controllano che un articolo non stia in più tabelle (cioè che non si trovi in più stati contemporaneamente in quanto sarebbe impossibile)

```
create assertion èsoloinrevisione
(
check( paper_revision(ID) NOT IN (SELECT ID
                                  FROM paper_published
                                  UNION
                                  SELECT ID
                                  FROM paper_rejected)    )
);

create assertion èsoloinpubblicato
(
        check( paper_published(ID) NOT IN (
                                  SELECT ID
                                  FROM paper_rejected)    )
);
```

Ora con quest'ultima asserzione controlliamo che nella nostra tabella degli studiosi sia presente almeno una persona in quanto non ha molto senso che sia una tabella vuota:

```
create assertion almenouno
(
check ( 1<= ( SELECT count(*)
              FROM studioso))
);
```

Nota implementativa:

Ultima nota sulla implementazione del codice in mySQL. Come si vede sarebbero molte le righe di codice da scrivere con l'elevata probabilità di commettere qualche errore di battitura (non correggibile se non ridigitando l'intera operazione). Per questo abbiamo deciso di usare uno script per l'inserimento.
La sintassi per gli script (eseguibili da una shell) è:
shell> mysql <bach file
Oppure si può usare (come abbiamo fatto) l'analoga sintassi per la consolle:
mysql> source comandi.txt
Notiamo che per usare quest'ultimo metodo basta copiare il nostro script nella cartella script all'interno della directory di mysql.

Query

DATA MANIPULATION LANGUAGE:

- *ARTICOLI*

1) Verifica della pubblicazione di un articolo:
   a. *Mediante il titolo*

```
SELECT    title, author, abstract
FROM      paper_published
WHERE     title like 'XXX'
```

   b. *Mediante titolo e autore*

```
SELECT    title, author, abstract
FROM      paper_published
WHERE     title like 'XXX' AND author like 'XXX'
```

2) Elenco di tutti gli articoli che sono stati pubblicati a partire da una certa data:

```
SELECT    title, contact_author, conference, publisheddate
FROM      paper_published
WHERE     publisheddate > 'XXX'
```

3) Elenco di tutti gli articoli che sono stati pubblicati nell'anno corrente;

```
SELECT    title, contact_author, conference, publisheddate
FROM      paper_published
WHERE     YEAR(publisheddate)=YEAR(CURRENT_DATE);
```

4) Elenco degli articoli gestiti dal sub-EIC e da quanto tempo gli sono stati affidati;

```
SELECT    paper_revision .title, paper_revision .contact_author,
           paper_revision .conference, paper_revision .deliverydate,
FROM      (paper_revision INNER JOIN person ON subEIC=code)
WHERE      subEIC IS NOT NULL;
```

5) Elenco di tutti gli articoli rifutati per Major Revision nell'ultimo anno:

```
SELECT    paper_rejected .ID, paper_rejected.version, paper_rejected
          .title, paper_rejected.contact_author,
          paper_rejected.rejectdate,
          paper_rejected.path,
           person.name, person.surname, person.email
FROM      (paper_rejected INNER JOIN person ON
          contact_author=code)
WHERE      paper_rejected.hardrj=FALSE AND
           YEAR(rejectiondate)>=YEAR(current_date)-1;
```

6) Elenco di tutti gli articoli, sia rifiutati, sia pubblicati, sia in revisione a partire dallo scorso anno:

```
SELECT    paper_revision .ID, paper_revision .version,
          paper_revision .title, person.name, person.surname,
          paper_revision .conference, paper_revision .deliverydate,
          paper_revision .path , person.name, person.email
FROM      (paper_revision INNER JOIN person ON
          contact_author=code)
WHERE      YEAR(deliverydate)>=YEAR(current_date)-1
UNION
SELECT    paper_rejected .ID, paper_rejected .version, paper_rejected
          .title, person.name, person.surname,
          paper_rejected .conference, paper_rejected .rejectiondate,
          paper_rejected .path
FROM      (paper_rejected INNER JOIN person ON
          contact_author=code)
WHERE      YEAR(rejectiondate)>=YEAR(current_date)-1
UNION
SELECT    paper_published .ID, paper_published .version,
          paper_published .title, Person.name, person.surname,
          paper_published .conference, paper_published .rejectdate,
          paper_published .path
FROM      (paper_published INNER JOIN person ON
          contact_author=code)
WHERE      YEAR(publisheddate)>=YEAR(current_date)-1;
```

7) Elenco di tutti gli articoli completamente rifutati (hard-rejection):

```
SELECT    paper_rejected .ID, paper_rejected.version,
          paper_rejected .title, paper_rejection.rejectdate,
          paper_rejected.path, paper_rejected.contact_author,
           person.name, person.surname, person.email
FROM      (paper_rejected INNER JOIN person ON
          contact_author=code)
WHERE      paper_rejected.hardrj='Y';
```

Nell'ultimo anno:

```
SELECT    paper_rejected .ID, paper_rejected.version,
          paper_rejected .title, paper_rejected.contact_author,
          paper_rejected.rejectdate, paper_rejected.path,
           person.name, person.surname, person.email
FROM      (paper_rejected INNER JOIN person ON
          contact_author=code)
WHERE      paper_rejected.hardrj=TRUE AND
           YEAR(rejectiondate)>=YEAR(current_date)-1;
```

- *REVISIONI*

8) Visualizzazione del numero di revisori associati ad un articolo

```
SELECT    paper_revision.title, review.ID, review.version, count(*)
FROM      (reviewer INNER JOIN paper_revision ON
            reviewer.ID=paper_revision.ID)
WHERE     reviewer.ID='XXX' AND version='YYY'
GROUP BY  paper_revision.title, reviewer.ID, reviewer.version;
```

9) Visualizzazione dell'elenco dei revisori associati ad un articolo.

```
SELECT    reviewer.ID, reviewer.version, person.name, person.surname,
          person.email, person.note, reviewer.alive
FROM      (review INNER JOIN person ON person.code = reviewer.code)
WHERE     reviewer.ID = 'XXX' AND reviewer.version='YYY';
```

10) Visualizzazione di tutte le revisioni in corso, comprensivamente del tempo rimanente per la consegna (tempoassoluto)

```
SELECT    reviewer.ID, reviewer.version, paper_revision.title,
          paper_revision.path, reviewer.code, reviewer.deliverarydate,
          reviewer.promisedate, reviewer.alive
FROM      (reviewer INNER JOIN paper_revision ON
          reviewer.ID=paper_revision.ID)
```

Questa query si propone di visualizzare il tempo "assoluto", ovverosia semplicemente le date promesse per la revisione.La successiva opera anche il calcolo del tempo trascorso fra la data attuale e la data promessa.

11) Visualizzazione di tutte le revisioni in corso, comprensivamente del tempo rimanente per la consegna (tempo relativo)

```
SELECT    reviewer.ID, reviewer.version, paper_revision.title,
          paper_revision.path, reviewer.code, reviewer.alive,
          reviewer.deliverarydate, reviewer.promisedate,
          (month (promisedate) – month(current_date)) –
          12*(year(promisedate)<year(current_date)) +
          12*(year(promisedate)>year(current_date))
          AS timeleft
FROM      (reviewer INNER JOIN paper_revision ON
          reviewer.ID=paper_revision.ID);
```

Questa query sfrutta le funzionalità offerte da mysql per calcolare i mesi di tempo rimasto per la consegna; essa calcola semplicemente la differenza fra i mesi nella data, successivamente va a sottrarre 12 mesi se l'anno della data odierna è superiore all'anno della data promessa, oppure somma 12 mesi se la data promessa è prevista per l'anno successivo. Si sfrutta il fatto che i valori booleani (0/1) ritornati dalle condizioni di verità sono interpretati a tutti gli effetti come numeri interi. Sfortunatamente, nel caso di coincidenza fra la data prevista e la data corrente, non si visualizzano i giorni di scarto esistenti.

12) Visualizzazione di tutte le revisioni in corso per un certo articolo.

Selezione dell'articolo attraverso codice e versione:

```
SELECT   paper_revision.title, reviewer.code, person.name,
         person.surname, person.email, reviewer.deliverarydate,
         reviewer.promisedate, reviewer.alive
FROM     (reviewer INNER JOIN paper_revision ON
         reviewer.ID=paper_revision.ID)INNER JOIN person ON
         reviewer.code=person.code
WHERE    reviewer.ID="XXX" and reviewer.version="XXX"
```

Selezione dell'articolo attraverso il titolo:

```
SELECT   paper_revision.title, reviewer.code, person.name,
         person.surname, person.email, reviewer.deliverarydate,
         reviewer.promisedate, reviewer.alive
FROM     (reviewer INNER JOIN paper_revision ON
         reviewer.ID=paper_revision.ID) INNER JOIN person ON
         reviewer.code=person.code
WHERE    paper_revision.title ='XXX'
```

13) Visualizzazione di tutte le revisioni in corso per un certo articolo, comprensive di tempo mancante per la consegna:

```
SELECT   reviewer.ID, reviewer.version, paper_revision.title,
         paper_revision.path, reviewer.code, reviewer.alive,
         reviewer.deliverarydate, reviewer.promisedate,
         (month (promisedate) – month(current_date)) –
         12*(year(promisedate)<year(current_date)) +
         12*(year(promisedate)>year(current_date))
         AS timeleft
FROM     (reviewer INNER JOIN paper_revision ON
         reviewer.ID=paper_revision.ID)
WHERE    reviewer.ID="XXX" and reviewer.version="XXX";
```

Questa query ha la medesima struttura e sfrutta le medesime funzionalità della numero 11, con la sola differenza che seleziona i dati di un solo articolo.

14) Visualizzazione delle promesse fatte da un revisore, comprensiva della data di patteggiamento, per un articolo.

Selezione attraverso il codice del revisionatore dell'articolo

```
SELECT   promised_date.ID, promised_date.version, paper_revision.title,
         person.code, person.name, person.surname,
         promised_date.promisedate
FROM     ((promised_date INNER JOIN person
         ON promised_date.code = person.code) INNER JOIN
         paper_revision ON paper_revision.ID = promised_date.ID)
WHERE    promised_date.code='XXX'AND promised_date.ID='YYY'
         AND promised_date.version='ZZZ'
```

Selezione attraverso il codice del revisonatore e il titolo dell'articolo

```
SELECT    promised_date.ID, promised_date.version, paper_revision.title,
          person.code, person.name, person.surname,
          promised_date.promisedate
FROM      ((promised_date INNER JOIN person
          ON promised_date.code = person.code) INNER JOIN
          paper_revision ON paper_revision.ID = promised_date.ID)
WHERE     promised_date.code='XXX' AND paper_revision.title='YYY';
```

Questa query sfrutta il fatto che òla tabella promise_date contenga sia un riferimento al reviewer nel campo code, sia un riferimento all'articolo mediante ID e version, campi che ci permettono di visualizzare i dati anagrifici dello studioso e le caratteristiche dell'articolo.

15) Visualizzazione del numero di promesse fatte da un revisore per un articolo.

```
SELECT    promised_date.ID, promised_date.version, paper_revision.title,
          person.code, person.name, person.surname, count(*)
FROM      ((promised_date INNER JOIN person
          ON promised_date.code = person.code) INNER JOIN
          paper_revision ON paper_revision.ID = promised_date.ID)
WHERE     promised_date.code='XXX' AND promised_date.ID='YYY'
          AND promised_date.version='ZZZ'
GROUPBY   promised_date.ID, promised_date.version, paper_revision.title,
            person.code, person.name, person.surname
```

Rispetto alla precedente, questo comando no visualizza il dettaglio delle promesse, ma solo il numero di patteggiamenti effettuato.

16) Visualizzazione delle promesse fatte da un revisore, comprensiva della data di patteggiamento per gli articoli revisionati.

Selezione attraverso il codice del revisionatore

```
SELECT    promised_date.ID, promised_date.version, paper_revision.title,
          person.code, person.name, person.surname,
          promised_date.promisedate
FROM      ((promised_date INNER JOIN person
          ON promised_date.code = person.code) INNER JOIN
          paper_revision ON paper_revision.ID = promised_date.ID)
WHERE     promised_date.code='XXX'
```

Selezione attraverso il nome del revisonatore

```
SELECT    promised_date.ID, promised_date.version, paper_revision.title,
          person.code, person.name, person.surname,
          promised_date.promisedate
FROM      ((promised_date INNER JOIN person
          ON promised_date.code = person.code) INNER JOIN
          paper_revision ON paper_revision.ID = promised_date.ID)
WHERE     person.surname='XXX'
```

Così si visualizzano tutte le promesse fatte articolo per articolo: questo può consentire una valutazione dell'efficienza del reviewer.

17) Elenco di tutti gli articoli in revisione:

```
SELECT    paper_revision.ID, paper_revision.version,
          paper_revision.title, paper_revision.path
FROM      paper_revision;
```

Nella tabella reviewer si mantiene traccia solamente degli articoli in revisione.

18) Elenco di tutte le revisioni correnti:

```
SELECT    reviewer.ID, reviewer.version, paper_revision.title,
          reviewer.code, reviewer.alive,
          person.name, person.surname, person.email
FROM      (reviewer INNER JOIN paper_revision ON
          reviewer.ID=paper.revision.ID)
          INNER JOIN person ON person.code=reviewer.code
GROUP BY (reviewer.ID, reviewer.version, paper_revision.title);
```

Le revisioni correnti sono quelle che riguardano articoli il cui processo non è ancora ultimato. Non si sfrutta il flag alive perché quello viene aggiornato conforme l'attività del singolo reviewer e questo implica che alcuni articoli si trovano ancora nello stato di revisione perché solo alcuni dei loro revisionatori hanno terminato il lavoro, mentre altri non ancora.

19) Elenco di tutte le revisioni ancora non concluse per un certo articolo:

```
SELECT    reviewer.ID, reviewer.version, paper_revision.title,
          reviewer.code,  person.name, person.surname, person.email
FROM      ((reviewer INNER JOIN paper_revision ON
          reviewer.ID=paper_revision.ID AND
          reviewer.version=paper_revision.version)
          INNER JOIN person ON person.code=reviewer.code)
WHERE     reviewer.alive='Y' AND reviewer.ID='XXX' AND
          reviewer.version='YYY'
GROUP BY  reviewer.ID, reviewer.version, paper_revision.title;
```

20) Visualizzazione di tutti i precedenti revisori di un articolo ex Major Revision

```
SELECT    paper_revision.ID, paper_revision.title, person.name,
          person.surname, person.email
FROM      ((paper_revision INNER JOIN reviewer ON
          paper_revision.rearticle=reviewer.ID)INNER JOIN person ON
          person.code=reviewer.code)
WHERE     paper_revision.ID='XXX'
```

Si noti come il campo "rearticle" di paper_revision contenga il codice associato all'articolo nella revisione precedente e quindi possa essere sfruttato per rintracciare i revisori precedenti attraverso un join con la tabella "review".

21) Visualizzazione della storia passata di un articolo

Visualizzazione della storia di un articolo in Major Revision

```
SELECT    paper_revision.ID, paper_revision.title, paper_revision.path,
          person.name, person.surname, person.email
FROM      ((paper_revision INNER JOIN reviewer ON
          paper_revision.rearticle=reviewer.ID)INNER JOIN person ON
          person.code=reviewer.code)
WHERE     paper_revision.ID='XXX'
UNION
SELECT    paper_revision.ID, paper_revision.title, paper_revision.path,
          person.name, person.surname, person.email
FROM      (paper_revision INNER JOIN person ON
          person.code=reviewer.code)
WHERE     paper_revision.ID='XXX'
```

Visualizzazione della storia di un articolo in Minor Revision

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          person.name, person.surname, person.email
FROM      (paper_revision INNER JOIN person ON
          person.code=reviewer.code)
WHERE     paper_revision.ID='XXX'
GROUPBY (paper_revision.ID, paper_revision.version)
```

In questo modo posso selezionare tutti i revisori avuti da un articolo nei processi di revisione subiti. L'ipotesi, comunque non fittizia, su cui si basano le query rappresentata dal fatto che l'articolo non abbia subito più di una Minor revision, oppure più di una major revision. Per esplorare quest'ultima possibilità la potenza di SQL è limitata dal momento che non offre la ricorsione fra i suoi metodi.

- *PERSONALE*

22) Visualizzazione di tutti i vari ruoli ricoperti da uno studioso

Selezione mediante il solo codice:

```
SELECT    reviewer.ID, reviewer.version, paper_revision.title,
          paper_revision.abstract
FROM      (reviewer INNER JOIN paper_revision ON
          reviewer.ID=paper_revision.ID
          AND reviewer.version=paper_revision.version)
WHERE     person.code='XXX'
UNION
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          paper_revision.abstract
FROM      paper_revision
WHERE     paper_revision.associatededitor='XXX' OR
          paper_revision.subEIC='XXX' OR
          paper_revision.contact_author='XXX' ;
```

Selezione mediante il nome e il cognome

```
SELECT    reviewer.ID, reviewer.version, paper_revision.title,
          paper_revision.abstract
FROM      (reviewer INNER JOIN paper_revision ON
          reviewer.ID=paper_revision.ID AND
          reviewer.version=paper_revision.version) INNER JOIN person
          ON
          reviewer.code=person.code
WHERE     person.name='XXX' AND person.surname='ZZZ'
UNION
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          paper_revision.abstract
FROM      ((paper_revision INNER JOIN person ON
          paper_revision.associateEditor=person.code)
WHERE     person.name='XXX' AND person.surname='ZZZ'
UNION
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          paper_revision.abstract
FROM      ((paper_revision INNER JOIN person ON
          paper_revision.contact_author=person.code)
WHERE     person.name='XXX' AND person.surname='ZZZ'
UNION
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          paper_revision.abstract
FROM      ((paper_revision INNER JOIN person ON
          paper_revision.subEIC=person.code)
WHERE     person.name='XXX' AND person.surname='ZZZ' ;
```

Purtroppo non è possibile con MySQL distinguere i vari ruoli ricoperti da uno studioso e dunque si lascerà all'interfaccia software la distinzione, ad esempio mediante via grafica degli articoli per cui lo studioso è reviewer e quelli per cui ricopre un 'altro ruolo.
A MySQL si affida soltanto il reperimento di codice e titolo degli articoli in questione.

23) Visualizzazione dell'elenco degli studiosi

```
SELECT  (*)
FROM  person;
```

- *ASSOCIATED EDITOR*

24) Visualizzazione del carico di lavoro di ciascun Associated Editor:

Visualizzazione del numero di articoli gestiti

```
SELECT    person.code, person.name, person.surname, count(*)
FROM      (paper_revision INNER JOIN reviewer ON
            paper_revision.ID=reviewer.ID) INNER JOIN person
            ON paper_revision.associatedEditor=person.code)
WHERE       reviewer.alive='Y'
GROU BY   person.code, person.name, person.surname
```

Visualizzazione degli articoli gestiti

```
SELECT    person.code, person.name, person.surname, paper_revision.ID,
          paper_revision.version, paper_revision.title,
          paper_revision.contact_author, paper_revision.abstract
FROM      (paper_revision INNER JOIN person ON
           associatedEditor= person.code)
GROU BY   person.code, person.name, person.surname;
```

25) Visualizzazione degli articoli gestiti da un Associate Editor

Selezione mediante il codice dell'AE

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          paper_revision.path, paper_revision.rearticle
FROM      paper_revision
WHERE     paper_revision.associateEditor='XXX';
```

Selezione mediante il nome del'AE

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          paper_revision.abstract, paper_revision.rearticle
FROM      (paper_revision INNER JOIN person ON
          associatedEditor= person.code)
WHERE     person.name='XXX' AND person.surname='YYY';
```

26) Visualizzazione di tutte le ultime comunicazioni con gli Associate Editor

```
SELECT    person.code, person,name, person.surname, person.email,
          paper_revision.ID, paper_revision.version, paper_revision.title,
          communication.deliverydate,
          communication.communicationtype,
FROM      (communication INNER JOIN paper_revision ON
          communication.ID= paper_revision.ID AND
          communication.version=paper_revision.version) INNER JOIN
          person ON communication.code=person.code
WHERE     communication.person='AE'  AND
          MONTH(communication.deliverydate)>=MONTH(current_date)-1;
GROUPBY   person.code, person,name, person.surname, person.email
```

Questa query fa uso della funzione MONTH per estrarre il mese dalla data e confrontarlo con quello della data corrente come se fosse un numero. Questo ci consente di visualizzare le comunicazionidell'ultimo mese.

27) Visualizzazione delle comunicazioni con un certo Associate Editor

Selezione mediante il nome

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          communication.deliverydate,
          communication.communicationtype, communication.response
FROM      (communication INNER JOIN paper_revision ON
          communication.ID= paper_revision.ID AND
          communication.version=paper_revision.version) INNER JOIN
          person ON communication.code=person.code
WHERE     person.name='XXX' AND person.surname='YYY'
GROUPBY   paper_revision.ID, paper_revision.version, paper_revision.title;
```

Selezione mediante il codice

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          communication.deliverydate,
          communication.communicationtype, communication.response
FROM      (communication INNER JOIN paper_revision ON
          communication.ID= paper_revision.ID AND
          communication.version=paper_revision.version)
WHERE     communication.code='XXX'
GROUPBY   paper_revision.ID, paper_revision.version, paper_revision.title
```

28) Visualizzazione delle comunicazioni pendenti con un certo Associate Editor

Selezione mediante il nome

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          communication.deliverydate,
          communication.communicationtype, communication.response
FROM      (communication INNER JOIN paper_revision ON
          communication.ID= paper_revision.ID AND
          communication.version=paper_revision.version) INNER JOIN
          person ON communication.code=person.code
WHERE     person.name='XXX' AND person.surname='YYY' AND
          communication.response='N'
GROUPBY   paper_revision.ID, paper_revision.version, paper_revision.title;
```

Selezione mediante il codice

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          communication.deliverydate,
          communication.communicationtype, communication.response
FROM      (communication INNER JOIN paper_revision ON
          communication.ID= paper_revision.ID AND
          communication.version=paper_revision.version)
WHERE     communication.code='XXX' AND
          communication.response='N'
GROUPBY   paper_revision.ID, paper_revision.version, paper_revision.title
```

La prima query riguardante le comunicazioni comprende indiscriminatamente tutti gli AE; successivamente per un singolo AE si visualizza la storia delle comunicazioni, infine si offre la possibilità di visualizzare solo quelle in sospeso.

29) Visualizzazione dei reviewers gestiti da un Associate Editor

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          reviewer.code, person.name, person.surname, person.email
FROM      ((paper_revision INNER JOIN review ON
          paper_revision.ID= reviewer.ID AND
          paper_revision.version=reviewer.version) INNER JOIN person
          ON reviewer.code=person.code)
WHERE     paper_revision.associateEditor='XXX'
GROUPBY   paper_revision.ID, paper_revision.version, paper_revision.title;
```

- *REVISORI*

30) Visualizzazione del carico di lavoro di ciascun reviewer

Visualizzazione del numero di articoli da revisionare:

```
SELECT    person.code, person.name, person.surname, count(*)
FROM      (reviewer INNER JOIN person
          ON reviewer.code = person.code)
WHERE     reviewer.alive='Y'
GROUP BY  person.code, person.name, person.surname;
```

Visualizzazione dell'elenco degli articoli in revisione per ciascun reviewer

```
SELECT    person.code, person.name, person.surname, paper_revision.ID,
          paper_revision.version, paper_revision.title,
          paper_revision.abstract, reviewer.deliverydate
FROM      ((reviewer INNER JOIN person ON
          reviewer.code=person.code) INNER JOIN paper_revision ON
          reviewer.ID=paper_revision.ID AND
          reviewer.version=paper_revision.version)
WHERE     reviewer.alive='Y'
GROUP BY  person.code, person.name, person.surname;
```

31) Visualizzazione della mail di un reviewer

```
SELECT    name, surname, email, note
FROM      person
WHERE     name='XXX' OR code ='XXX';
```

32) Visualizzazione di tutti gli articoli revisionati da un certo reviewer nell'ultimo:

```
SELECT DISTINCT paper_revision.ID, paper_revision.version,
                paper_revision.title,
                paper_revision.abstract
FROM      (paper_revision INNER JOIN reviewer
          ON paper_revision.ID = reviewer.ID AND
          paper_revision.code=reviewer.code)
WHERE     YEAR(deliverydate)=YEAR(CURRENT_DATE);
```

33)
Visualizzazione del numero di promesse fatte da un reviewer per gli articoli revisionati.

```
SELECT    person.code, person.name, person.surname, promised_date.ID,
          promised_date.version, paper_revision.title,
          paper_revision.abstract, count(*)
FROM      ((promised_date INNER JOIN person
          ON promised_date.code = person.code) INNER JOIN
          paper_revision ON paper_revision.ID = reviewer.ID AND
          paper_revision.version=reviewer.version)
WHERE     promised_date.code='XXX' OR (person.name='YYY' AND
          person.surname='ZZZ');
GROUP BY  person.code, person.name, person.surname, promised_date.ID,
          promised_date.version, paper_revision.title;
```

34) Visualizzazione di tutti gli articoli attualmente in revisione per quel revisore

Selezione mediante codice del reviewer

```
SELECT    reviewer.ID, reviewer.version, paper_revision.title,
          paper_revision.abstract, paper_revision.deliverydate
FROM      (reviewer INNER JOIN paper_revision
          ON reviewer.ID = paper_revision.ID AND
          review.version= paper_revision.version)
WHERE     reviewer.code='XXX';
```

Selezione mediante il nome del reviewer

```
SELECT    promised_date.ID, promised_date.version, paper_revision.title,
          paper_revision.abstract, paper_revision.deliverydate
FROM      ((review INNER JOIN paper_revision
          ON reviewer.ID = paper_revision.ID AND reviewer.version=
          paper_revision.version) INNER JOIN person ON
          reviewer.code=person.code)
WHERE     person.name='XXX' AND person.surname='YYY';
```

35) Visualizzazione del nome di tutti i revisori che hanno lavorato a qualche articolo in un anno

```
SELECT DISTINCT(person.code, person.name, person.surname,
          person.email)
FROM      (reviewer INNER JOIN person ON
          reviewer.code=person.code)
WHERE     reviewer.deliverydate>='1/1/200X' AND
          reviewer.deliverydate<='31/1/200X' ;
```

36) Visualizzazione del nome di tutti i revisori che hanno lavorato a qualche articolo lo scorso anno

```
SELECT DISTINCT(person.code, person.name, person.surname,
          person.email)
FROM      (reviewer INNER JOIN person ON
          reviewer.code=person.code)
WHERE     YEAR(reviewer.deliverydate)>=YEAR(current_date);
```

37) Visualizzazione del nome di tutti i revisori che hanno lavorato a qualche articolo raggruppati per anno

```
SELECT    YEAR(reviewer.deliverydate) AS years, person.code,
          person.name, person.surname, person.email
FROM      (reviewer INNER JOIN person ON
          reviewer.code=person.code)
GROUPBY   years;
```

38) Visualizzazione di tutte le ultime comunicazioni con i reviewers:

```
SELECT      person.code, person,name, person.surname, person.email,
            paper_revision.ID, paper_revision.version, paper_revision.title,
            communication.deliverydate,
            communication.communicationtype,
FROM        (communication INNER JOIN paper_revision ON
            communication.ID= paper_revision.ID AND
            communication.version=paper_revision.version) INNER JOIN
            person ON communication.code=person.code
WHERE       communication.person='RW'  AND
        MONTH(communication.deliverydate)>=MONTH(current_date)-1;
GROUPBY  person.code, person,name, person.surname, person.email
```

39) Visualizzazione delle comunicazioni con un certo reviewer:

Selezione mediante il nome

```
SELECT      paper_revision.ID, paper_revision.version, paper_revision.title,
            communication.deliverydate,
            communication.communicationtype, communication.response
FROM        (communication INNER JOIN paper_revision ON
            communication.ID= paper_revision.ID AND
            communication.version=paper_revision.version) INNER JOIN
            person ON communication.code=person.code
WHERE       person.name='XXX' AND person.surname='YYY'
GROUPBY   paper_revision.ID, paper_revision.version, paper_revision.title;
```

Selezione mediante il codice

```
SELECT      paper_revision.ID, paper_revision.version, paper_revision.title,
            communication.deliverydate,
            communication.communicationtype, communication.response
FROM        (communication INNER JOIN paper_revision ON
            communication.ID= paper_revision.ID AND
            communication.version=paper_revision.version)
WHERE       communication.code='XXX'
GROUPBY    paper_revision.ID, paper_revision.version, paper_revision.title
```

40) Visualizzazione delle comunicazioni pendenti con un certo reviewer

Selezione mediante il nome

```
SELECT      paper_revision.ID, paper_revision.version, paper_revision.title,
            communication.deliverydate,
            communication.communicationtype, communication.response
FROM        (communication INNER JOIN paper_revision ON
            communication.ID= paper_revision.ID AND
            communication.version=paper_revision.version) INNER JOIN
            person ON communication.code=person.code
WHERE       person.name='XXX' AND person.surname='YYY' AND
            communication.response='N'
GROUPBY   paper_revision.ID, paper_revision.version, paper_revision.title;
```

Selezione mediante il codice

```
SELECT    paper_revision.ID, paper_revision.version, paper_revision.title,
          communication.deliverydate,
          communication.communicationtype, communication.response
FROM      (communication INNER JOIN paper_revision ON
          communication.ID= paper_revision.ID AND
          communication.version=paper_revision.version)
WHERE     communication.code='XXX' AND
          communication.response='N'
GROUPBY   paper_revision.ID, paper_revision.version,
          paper_revision.title;
```

41) Visualizzazione dei reviewers che stanno già lavorando ad un articolo

```
SELECT    person.name, person.surname, person.email, person.note,
          reviewer.alive
FROM      (reviewer INNER JOIN person
           ON person.code = review.code)
WHERE     reviewer.ID = 'XXX';
```

DATA DEFINITION LANGUAGE:

- *ARTICOLI*

1. Inserimento di un nuovo articolo fra quelli in revisione:

   INSERT INTO paper_revision VALUES (<identificatore>,<versione>,<title>,<abstract>,<<conference>>,<contact_author>,<<ae>>,<<subEIC>>,<deliverydate>,<<codiceprecedente>>,<path>)

   Si noti come i primi quattro campi, insieme all'autore, alla data di ricezione e al path, saranno inseriti sicuramente, mentre l'associate editor può essere associato in un momento successivo mediante il comando:

   UPDATE paper_revision SET associateEditor='<ae>' WHERE ID='XXX' AND version= 'YYY' ;

   L'identificatore e la versione verranno calcolati automaticamente via software.

2. Inserimento di un nuovo articolo che sarà gestito dal subEIC

   INSERT INTO paper_revision VALUES (<identificatore>,<versione>,<title>,<abstract>,<<conference>>,<contact_author>,NULL,<subEIC's code> ,<deliverydate>,NULL,<path>)

   Si noti come gli articoli gestiti dal subEIC non abbiano alcun associate editor associate, come del resto non verranno inseriti nella tabella review, poiché tutto il processo di revisione è completamente mascherato al EIC effettivo, il quale tiene traccia solamente dell'esistenza di tali articoli.

3. Inserimento di una nuova versione degli articoli

```
INSERT INTO past_version
        SELECT      ID, version, title, abstract, conference,
                    contact_author,associateEditor, current_date, rearticle,
                    path
        FROM        paper_revision
        WHERE ID='XXX'
        ;

UPDATE paper_revision SET version=version+1 WHERE ID='XXX' ;
```

Le versioni precedenti degli articoli vengono salvate nell'apposita tabella past_version; successivamente si provvede ad aggiornare il valore della versione, in paper_revision e in review.Si noti come la tabella Past_version memorizza non la data di consegna deliberi date, ma la data di rifiuto rejection date che sarebbe la data corrente al momento in cui si svolge l'operazione. Essa si ottiene mediante il comando CURRENT_DATE di Mysql.

4. Modifica generale di un articolo

```
UPDATE paper_revision SET <attributo> WHERE ID="XXX" AND
version="YYY"
```

Questo è un comando generico per la definizione di nuovi valori degli attributi.

5. Pubblicazione di un articolo

```
INSERT INTO paper_published
        SELECT *
        FROM        paper_revision
        WHERE ID='XXX'
        ;
UPDATE review SET alive="N" where ID ="XXX" AND version="xxx"
DELETE FROM paper_revision WHERE ID="XXX"
```

La pubblicazione comporta l'inserimento dell'articolo nello schema relazionale degli articoli pubblicati, la modifica del flag "alive" nella tabella delle revisioni per individuare che la revisione è conclusa, l'eliminazione dell'articolo dalla tabella "paper_revision".

6. Rifiuto (hard rejection) di un articolo

```
INSERT INTO paper_rejected
        SELECT      ID, version, title, abstract, conference,
                    contact_author,associateEditor, current_date, rearticle,
                    path,  'Y'
        FROM         paper_revision
        WHERE ID='XXX' ;

UPDATE review SET alive="N" where ID ="XXX" AND version="xxx";
DELETE FROM paper_revision WHERE ID="XXX";
```

Anche per il rifiuto, l'articolo deve essere "trasferito" nella tabella dei rifiutati, con il flag "hard_rejection" settato a "Y" ad identificare che il rifiuto è definitivo. Inoltre, si aggiorna la tabella "review" per indicare che il processo è concluso e si elimina l'articolo fra quelli in revisione. Si noti nuovamente come si fa uso del comando "CURRENT_DATE" per stabilire la data di rifiuto, che coincide con la data corrente in cui si effettua l'operazione.

7. Rifiuto (soft rejection) di un articolo per Major revision

```
INSERT INTO paper_rejected
    SELECT    ID, version, title, abstract, conference,
              contact_author,associateEditor, current_date, rearticle,
              path,  'N'
    FROM      paper_revision
    WHERE ID='XXX' ;

UPDATE review SET alive="N" where ID ="XXX" AND version="xxx";

UPDATE paper_revision SET ID=<new identifier> AND version="1"
```

Al contrario del rifiuto definitivo, il rifiuto per Major Revision non comporta la cancellazione dell'articolo dalla tabella paper_revision, ma semplicemente la sua ridenominazione con un nuovo codice.Per il resto la struttura della query è identica a quella della bocciatura, eccezion fatta per il flag "hard_rejection" che viene settato su 'N' ad indicare che il rifuto non è definitivo.

- *REVISIONI*

8. Aggiornamento delle date per cui è stata promessa la revisione:

```
INSERT INTO promise_date VALUES (<insert code>, <insert ID> , <insert
version> , <new insert date>);
UPDATE review SET promisedate= <new date> WHERE code=<insert
code>AND ID=<insert ID> AND version=<insert version>;
```

Nota: questa query serve semplicemente per inserire una nuova data promessa. La parte di UPDATE serve per mantenere aggiornato il campo promisedate della tabella reviewer.

9. Associazione di un revisore al rispettivo articolo:

```
INSERT INTO reviewer VALUES (<code>, <ID> , <version> ,
<deliverydate>, <promisedate>,'Y');
```

Semplicemente si aggoirna la tabella reviewer. L'articolo figura nella tabella degli articoli in revisione dal momento in cui l'EIC se lo prende a carico, quindi al momento dell'associazione dei revisori è già presente.

10. Aggiornamento a revisione ultimata

```
UPDATE reviewer SET alive='N' WHERE ID = 'XXX' AND version='x'
AND code='YYY';
```

SI setta il valore del flag 'alive' per una certa associazione revisore – articolo. IL flag alive, infatti, vale singolarmente per ogni reviewer.

- *PERSONALE*

11. Inserimento di uno studioso nella base di dati:

    INSERT INTO person VALUES (<code>, <name>, <surname>, <email>, <<notes>>);

12. Aggiornamento del campo note e dei dati anagrafici:

    UPDATE person SET notes =’xxxxxxxxxxxxx’ WHERE code=’XXX’;
    UPDATE person SET email =’xxx’ WHERE code=’XXX’;

13. Rimozione di uno studioso dalla base di dati:

    DELETE FROM person WHERE code=’XXX’ OR (name=’YYY’ AND surname=’ZZZ’);

    Si noti come mediante l’operatore logico OR è possibile a scelta effettuare una cancellazione o mediante il codice oppure mediante nome e cognome.

- *ASSOCIATE EDITORS*

14. Aggiornamento delle comunicazioni con gli associate editor

    INSERT INTO communication VALUES (<code>, <ID>, <version>, CURRENT_DATE, <oggetto>, ‘AE’, ‘Y’);

    Si fa uso dell’operatore current_date supponendo di aggiornare la base di dati al momento della spedizione della mail;l’oggetto della mail può essere:
    -Richiesta di una lista di reviewers per un articolo;
    -Sollecito della spedizione di una lista;
    -Richiesta di un parere su di un articolo;
    -Sollecito di tale parere;
    -Ricezione di una risposta.

- *REVIEWERS*

15. Aggiornamento delle comunicazioni con i reviewers:

    INSERT INTO communication VALUES (<code>, <ID>, <version>, CURRENT_DATE, <oggetto>, ‘RW’, ‘Y’);

    Si fa uso dell’operatore current_date supponendo di aggiornare la base di dati al momento della spedizione della mail;l’oggetto della mail può essere:
    -Richiesta di revisione;
    -Patteggiamento sulla data di consegna;
    -Sollecito di una consegna;
    -Ricezione di una risposta.

Nota al database:
poiché, come si è detto la versione di mySQL da noi utilizzata non ha le assertion , abbiamo progettato una query che, a seconda dei risultati ritornati, ci avrebbe aiutato a capire se ci fossero degli errori nell'intergrità referenziale. Tuttavia con la presente query possiamo solo scoprire se vi sono degli errori, ma non siamo in grado di capire quale sia la tabella contenente l'errore. Per esempio se un articolo risulta sia pubblicato che bocciato non possiamo sapere in quale stato si trovi effettivamente, sappiamo solo che in uno stato scorretto. Tali errori devono essere evitati al momento dell'inserimento dei dati.
La query che esegue il controllo è:

```
SELECT    p1.ID
FROM      paper_rejected as p1, paper_revision as p2
WHERE     p1.ID =p2.ID
UNION
SELECT    p1.ID
FROM      paper_rejected as p1, paper_published as p3
WHERE     p1.ID =p3.ID
UNION
SELECT    p2.ID
FROM      paper_revision as p2, paper_published as p3
WHERE     p2.ID =p3.ID
```

Questa deve sempre ritornare un valore vuoto, se ritorna un valore,cioè l'ID di un articolo, vuol dire che questo compare in più di una tabella e c'è un errore. La verifica che la query sia una stringa vuota viene fatta via SW attraverso un opportino metodo Java.

Il resto dei vincoli e ulteriori controlli di sicurezza verranno affidati al software di interfacciamento. AD esempio l'integrità referenziale fra diverse tabelle per gli articoli sarà fattibile con la generazione automatica dei codici, così come evitare duplicazioni sarà possibile eseguendo controlli transazionali al momento della pubblicazione o del rifiuto di un articolo. Dunque, si integrerà Mysql con altro software per garantire le proprietà acide. (atomicità, consistenza, isolamento, persistenza.