
Compendio di documentazione

Raccolta della documentazione tecnica

Indice

01.	Benvenuto per sviluppatori	3
02.	Glossario dati	13
03.	Report tecnico del database	16
04.	Colori e aree	28
05.	Riepilogo database	30
06.	Manuale FK	32
07.	Cronologia modifiche	33
08.	GDPR e dati sensibili	35
09.	Report di analisi del codice	37
10.	Controllo di sicurezza	55
11.	Prestazioni	67
12.	Conformità del database	80
13.	Tabelle non utilizzate	88
14.	Mappa di utilizzo	89
15.	Estrazione query	90
16.	Interazione codice e database	92
17.	Indice di complessità	100
18.	Software Catalog Management	101
19.	Software Customer Management	102
20.	Software Location Management	103
21.	Software Payment Management	104
22.	Software Rental Management	105
23.	Software Sales Reporting	106
24.	Software Security and Authentication	107
25.	Software Staff Management	108
26.	Software Store Management	109



Manuale di Benvenuto per Sviluppatori

Production environment — database — Generato il 14/09/2026 13:36

Questo documento riassume la struttura del database — pensato per far orientare velocemente un nuovo sviluppatore o un team in outsourcing, senza dover leggere l'intero schema a mano.

Le 10 tabelle più centrali

Le tabelle con più relazioni (in entrata e in uscita) sono quasi sempre il cuore del modello dati — capirle bene copre la maggior parte della logica applicativa.

1. FILM 27 relazioni

Descrive i film disponibili a catalogo, con titolo, descrizione, anno di uscita, lingua, durata del noleggio, tariffa, costo di sostituzione, classificazione e caratteristiche speciali. È collegata a LANGUAGE e referenziata da FILM_ACTOR, FILM_CATEGORY, INVENTORY e FILM_TEXT.

Campo	Tipo	Chiave
FILM_ID	SMALLINT	PK
TITLE	VARCHAR(128)	
DESCRIPTION	TEXT	
RELEASE_YEAR	YEAR	
LANGUAGE_ID	TINYINT	FK
ORIGINAL_LANGUAGE_ID	TINYINT	FK
RENTAL_DURATION	TINYINT	
RENTAL_RATE	DECIMAL(4,2)	
LENGTH	SMALLINT	
REPLACEMENT_COST	DECIMAL(5,2)	
RATING	ENUM('G','PG','PG-13','R','NC-17')	
SPECIAL_FEATURES	SET('TRAILERS','COMMENTARIES','DELETED SCENES','BEHIND THE S	
LAST_UPDATE	TIMESTAMP	

2. STAFF 25 relazioni

Rappresenta i dipendenti dei negozi di noleggio, con dati anagrafici, indirizzo, email, negozio di appartenenza, credenziali di accesso e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT, RENTAL e STORE.

Campo	Tipo	Chiave
STAFF_ID	TINYINT	PK
FIRST_NAME	VARCHAR(45)	
LAST_NAME	VARCHAR(45)	
ADDRESS_ID	SMALLINT	FK
PICTURE	BLOB	
EMAIL	VARCHAR(50)	
STORE_ID	TINYINT	FK
ACTIVE	BOOLEAN	
USERNAME	VARCHAR(16)	
PASSWORD	VARCHAR(40)	
LAST_UPDATE	TIMESTAMP	

3. STORE 25 relazioni

Identifica i negozi di noleggio film, con il direttore responsabile e l'indirizzo della sede. È collegata a STAFF e ADDRESS e referenziata da CUSTOMER, INVENTORY e STAFF.

Campo	Tipo	Chiave
STORE_ID	TINYINT	PK
MANAGER_STAFF_ID	TINYINT	FK
ADDRESS_ID	SMALLINT	FK
LAST_UPDATE	TIMESTAMP	

4. CUSTOMER 20 relazioni

Rappresenta i clienti del noleggio film, con dati anagrafici, email, negozio di riferimento, indirizzo e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT e RENTAL.

Campo	Tipo	Chiave
CUSTOMER_ID	SMALLINT	PK
STORE_ID	TINYINT	FK
FIRST_NAME	VARCHAR(45)	
LAST_NAME	VARCHAR(45)	
EMAIL	VARCHAR(50)	
ADDRESS_ID	SMALLINT	FK
ACTIVE	BOOLEAN	
CREATE_DATE	DATETIME	
LAST_UPDATE	TIMESTAMP	

5. RENTAL 20 relazioni

Documenta i noleggi dei film, con data di noleggio, data di restituzione, cliente, dipendente e copia fisica noleggiata. È collegata a STAFF, INVENTORY e CUSTOMER e referenziata da PAYMENT.

Campo	Tipo	Chiave
RENTAL_ID	INT	PK
RENTAL_DATE	DATETIME	
INVENTORY_ID	MEDIUMINT	FK
CUSTOMER_ID	SMALLINT	FK
RETURN_DATE	DATETIME	
STAFF_ID	TINYINT	FK
LAST_UPDATE	TIMESTAMP	

6. ADDRESS 18 relazioni

Contiene gli indirizzi civici utilizzati da clienti, dipendenti e negozi, con dettagli come via, distretto, città, codice postale, telefono e posizione geografica. È collegata alla tabella CITY e referenziata da CUSTOMER, STAFF e STORE.

Campo	Tipo	Chiave
ADDRESS_ID	SMALLINT	PK
ADDRESS	VARCHAR(50)	
ADDRESS2	VARCHAR(50)	
DISTRICT	VARCHAR(20)	
CITY_ID	SMALLINT	FK
POSTAL_CODE	VARCHAR(10)	
PHONE	VARCHAR(20)	
LOCATION	GEOMETRY	
LAST_UPDATE	TIMESTAMP	

7. INVENTORY 15 relazioni

Rappresenta le copie fisiche dei film disponibili presso ciascun negozio, con un identificativo univoco per ogni copia. È collegata a STORE e FILM e referenziata da RENTAL.

Campo	Tipo	Chiave
INVENTORY_ID	MEDIUMINT	PK
FILM_ID	SMALLINT	FK
STORE_ID	TINYINT	FK
LAST_UPDATE	TIMESTAMP	

8. PAYMENT 15 relazioni

Registra i pagamenti effettuati dai clienti per i noleggi, con importo, data e riferimenti al cliente, al dipendente e al noleggio. È collegata a RENTAL, CUSTOMER e STAFF.

Campo	Tipo	Chiave
PAYMENT_ID	SMALLINT	PK
CUSTOMER_ID	SMALLINT	FK
STAFF_ID	TINYINT	FK
RENTAL_ID	INT	FK
AMOUNT	DECIMAL(5,2)	
PAYMENT_DATE	DATETIME	
LAST_UPDATE	TIMESTAMP	

9. FILM_ACTOR 10 relazioni

Associa gli attori ai film in cui recitano, gestendo la relazione molti-a-molti tra ACTOR e FILM.

Campo	Tipo	Chiave
ACTOR_ID	SMALLINT	PK
FILM_ID	SMALLINT	PK
LAST_UPDATE	TIMESTAMP	

10. FILM_CATEGORY 10 relazioni

Associa i film alle categorie di appartenenza, gestendo la relazione molti-a-molti tra FILM e CATEGORY.

Campo	Tipo	Chiave
FILM_ID	SMALLINT	PK
CATEGORY_ID	TINYINT	PK
LAST_UPDATE	TIMESTAMP	

Solo le relazioni tra le tabelle più centrali elencate sopra — per lo schema completo, usa lo Schema ER di Modej.

CUSTOMER.ADDRESS_ID → ADDRESS.ADDRESS_ID

CUSTOMER.STORE_ID → STORE.STORE_ID

FILM_ACTOR.FILM_ID → FILM.FILM_ID

FILM_CATEGORY.FILM_ID → FILM.FILM_ID

INVENTORY.STORE_ID → STORE.STORE_ID

INVENTORY.FILM_ID → FILM.FILM_ID

PAYMENT.RENTAL_ID → RENTAL.RENTAL_ID

PAYMENT.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID

PAYMENT.STAFF_ID → STAFF.STAFF_ID

RENTAL.STAFF_ID → STAFF.STAFF_ID

RENTAL.INVENTORY_ID → INVENTORY.INVENTORY_ID

RENTAL.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID

STAFF.STORE_ID → STORE.STORE_ID

STAFF.ADDRESS_ID → ADDRESS.ADDRESS_ID

STORE.MANAGER_STAFF_ID → STAFF.STAFF_ID

STORE.ADDRESS_ID → ADDRESS.ADDRESS_ID

CUSTOMER.ADDRESS_ID → ADDRESS.ADDRESS_ID

CUSTOMER.STORE_ID → STORE.STORE_ID

FILM_ACTOR.FILM_ID → FILM.FILM_ID

FILM_CATEGORY.FILM_ID → FILM.FILM_ID

INVENTORY.STORE_ID → STORE.STORE_ID

INVENTORY.FILM_ID → FILM.FILM_ID

PAYMENT.RENTAL_ID → RENTAL.RENTAL_ID

PAYMENT.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID

PAYMENT.STAFF_ID → STAFF.STAFF_ID

RENTAL.STAFF_ID → STAFF.STAFF_ID

RENTAL.INVENTORY_ID → INVENTORY.INVENTORY_ID

RENTAL.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID

STAFF.STORE_ID → STORE.STORE_ID

STAFF.ADDRESS_ID → ADDRESS.ADDRESS_ID

STORE.MANAGER_STAFF_ID → STAFF.STAFF_ID

STORE.ADDRESS_ID → ADDRESS.ADDRESS_ID

CUSTOMER.ADDRESS_ID → ADDRESS.ADDRESS_ID

CUSTOMER.STORE_ID → STORE.STORE_ID

FILM_ACTOR.FILM_ID → FILM.FILM_ID

FILM_CATEGORY.FILM_ID → FILM.FILM_ID

INVENTORY.STORE_ID → STORE.STORE_ID

INVENTORY.FILM_ID → FILM.FILM_ID

PAYMENT.RENTAL_ID → RENTAL.RENTAL_ID

PAYMENT.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID

PAYMENT.STAFF_ID → STAFF.STAFF_ID

RENTAL.STAFF_ID → STAFF.STAFF_ID

RENTAL.INVENTORY_ID → INVENTORY.INVENTORY_ID

RENTAL.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID

STAFF.STORE_ID → STORE.STORE_ID

STAFF.ADDRESS_ID → ADDRESS.ADDRESS_ID

STORE.MANAGER_STAFF_ID → STAFF.STAFF_ID

STORE.ADDRESS_ID → ADDRESS.ADDRESS_ID

CUSTOMER.ADDRESS_ID → ADDRESS.ADDRESS_ID

CUSTOMER.STORE_ID → STORE.STORE_ID

FILM_ACTOR.FILM_ID → FILM.FILM_ID

FILM_CATEGORY.FILM_ID → FILM.FILM_ID

INVENTORY.STORE_ID → STORE.STORE_ID

INVENTORY.FILM_ID → FILM.FILM_ID

PAYMENT.RENTAL_ID → RENTAL.RENTAL_ID

PAYMENT.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID

PAYMENT.STAFF_ID → STAFF.STAFF_ID
RENTAL.STAFF_ID → STAFF.STAFF_ID
RENTAL.INVENTORY_ID → INVENTORY.INVENTORY_ID
RENTAL.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID
STAFF.STORE_ID → STORE.STORE_ID
STAFF.ADDRESS_ID → ADDRESS.ADDRESS_ID
STORE.MANAGER_STAFF_ID → STAFF.STAFF_ID
STORE.ADDRESS_ID → ADDRESS.ADDRESS_ID
CUSTOMER.ADDRESS_ID → ADDRESS.ADDRESS_ID
CUSTOMER.STORE_ID → STORE.STORE_ID
FILM_ACTOR.FILM_ID → FILM.FILM_ID
FILM_CATEGORY.FILM_ID → FILM.FILM_ID
INVENTORY.STORE_ID → STORE.STORE_ID
INVENTORY.FILM_ID → FILM.FILM_ID
PAYMENT.RENTAL_ID → RENTAL.RENTAL_ID
PAYMENT.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID
PAYMENT.STAFF_ID → STAFF.STAFF_ID
RENTAL.STAFF_ID → STAFF.STAFF_ID
RENTAL.INVENTORY_ID → INVENTORY.INVENTORY_ID
RENTAL.CUSTOMER_ID → CUSTOMER.CUSTOMER_ID
STAFF.STORE_ID → STORE.STORE_ID
STAFF.ADDRESS_ID → ADDRESS.ADDRESS_ID
STORE.MANAGER_STAFF_ID → STAFF.STAFF_ID
STORE.ADDRESS_ID → ADDRESS.ADDRESS_ID

Glossario tecnico completo (16 tabelle)

Tabella	Colonne	Relazioni	Descrizione
ACTOR	4	5	Rappresenta gli attori che recitano nei film del catalogo, con nome e cognome. Ogni attore può essere collegato a più film tramite la tabella FILM_ACTOR.

ADDRESS	9	18	Contiene gli indirizzi civici utilizzati da clienti, dipendenti e negozi, con dettagli come via, distretto, città, codice postale, telefono e posizione geografica. È collegata alla tabella CITY e referenziata da CUSTOMER, STAFF e STORE.
CATEGORY	3	6	Definisce le categorie o generi in cui i film sono classificati, come azione, commedia o drammatico. Ogni categoria può essere associata a più film tramite la tabella FILM_CATEGORY.
CITY	4	8	Rappresenta le città in cui risiedono gli indirizzi, con il nome della città e il paese di appartenenza. È collegata alla tabella COUNTRY e referenziata da ADDRESS.
COUNTRY	3	5	Identifica i paesi del mondo utilizzati per localizzare città e indirizzi. È referenziata dalla tabella CITY.
CUSTOMER	9	20	Rappresenta i clienti del noleggio film, con dati anagrafici, email, negozio di riferimento, indirizzo e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT e RENTAL.
FILM	13	27	Descrive i film disponibili a catalogo, con titolo, descrizione, anno di uscita, lingua, durata del noleggio, tariffa, costo di sostituzione, classificazione e caratteristiche speciali. È collegata a LANGUAGE e referenziata da FILM_ACTOR, FILM_CATEGORY, INVENTORY e FILM_TEXT.
FILM_ACTOR	3	10	Associa gli attori ai film in cui recitano, gestendo la relazione multi-a-molti tra ACTOR e FILM.
FILM_CATEGORY	3	10	Associa i film alle categorie di appartenenza, gestendo la relazione multi-a-molti tra FILM e CATEGORY.
FILM_TEXT	3	1	Contiene una versione testuale semplificata delle informazioni del film, con titolo e descrizione, utile per ricerche o visualizzazioni. È collegata alla tabella FILM.
INVENTORY	4	15	Rappresenta le copie fisiche dei film disponibili presso ciascun negozio, con un identificativo univoco per ogni copia. È collegata a STORE e FILM e referenziata da RENTAL.
LANGUAGE	3	10	Definisce le lingue disponibili per i film, utilizzate sia come lingua principale che come lingua originale. È referenziata dalla tabella FILM.
PAYMENT	7	15	Registra i pagamenti effettuati dai clienti per i noleggi, con importo, data e riferimenti al cliente, al dipendente e al noleggio. È collegata a RENTAL, CUSTOMER e STAFF.
RENTAL	7	20	Documenta i noleggi dei film, con data di noleggio, data di restituzione, cliente, dipendente e copia fisica noleggiata. È collegata a STAFF, INVENTORY e CUSTOMER e referenziata da PAYMENT.
STAFF	11	25	Rappresenta i dipendenti dei negozi di noleggio, con dati anagrafici, indirizzo, email, negozio di appartenenza, credenziali di accesso e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT, RENTAL e STORE.
STORE	4	25	Identifica i negozi di noleggio film, con il direttore responsabile e l'indirizzo della sede. È collegata a STAFF e ADDRESS e referenziata da CUSTOMER, INVENTORY e STAFF.

 Questo è solo un assaggio veloce

Analisi Tecnica genera il quadro completo — schema ER visivo, heatmap di utilizzo dal codice sorgente, mappa dei processi, indice di complessità — un report di 20-30 pagine pronto per la consegna al cliente, non solo le tabelle più centrali.

Generato automaticamente da Modej — Documentazione Flash

Glossario Dati

Dizionario Business delle Tabelle

14/09/2026, 11:29:23

ACTOR

4 colonne

Rappresenta gli attori che recitano nei film del catalogo, con nome e cognome. Ogni attore può essere collegato a più film tramite la tabella FILM_ACTOR.

ADDRESS

9 colonne

Contiene gli indirizzi civici utilizzati da clienti, dipendenti e negozi, con dettagli come via, distretto, città, codice postale, telefono e posizione geografica. È collegata alla tabella CITY e referenziata da CUSTOMER, STAFF e STORE.

CATEGORY

3 colonne

Definisce le categorie o generi in cui i film sono classificati, come azione, commedia o drammatico. Ogni categoria può essere associata a più film tramite la tabella FILM_CATEGORY.

CITY

4 colonne

Rappresenta le città in cui risiedono gli indirizzi, con il nome della città e il paese di appartenenza. È collegata alla tabella COUNTRY e referenziata da ADDRESS.

COUNTRY

3 colonne

Identifica i paesi del mondo utilizzati per localizzare città e indirizzi. È referenziata dalla tabella CITY.

CUSTOMER

9 colonne

Rappresenta i clienti del noleggio film, con dati anagrafici, email, negozio di riferimento, indirizzo e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT e RENTAL.

FILM

13 colonne

Descrive i film disponibili a catalogo, con titolo, descrizione, anno di uscita, lingua, durata del noleggio, tariffa, costo di sostituzione, classificazione e caratteristiche speciali. È collegata a LANGUAGE e referenziata da FILM_ACTOR, FILM_CATEGORY, INVENTORY e FILM_TEXT.

FILM_ACTOR

3 colonne

Associa gli attori ai film in cui recitano, gestendo la relazione multi-a-molti tra ACTOR e FILM.

FILM_CATEGORY

3 colonne

Associa i film alle categorie di appartenenza, gestendo la relazione multi-a-molti tra FILM e CATEGORY.

FILM_TEXT

3 colonne

Contiene una versione testuale semplificata delle informazioni del film, con titolo e descrizione, utile per ricerche o visualizzazioni. È collegata alla tabella FILM.

INVENTORY

4 colonne

Rappresenta le copie fisiche dei film disponibili presso ciascun negozio, con un identificativo univoco per ogni copia. È collegata a STORE e FILM e referenziata da RENTAL.

LANGUAGE

3 colonne

Definisce le lingue disponibili per i film, utilizzate sia come lingua principale che come lingua originale. È referenziata dalla tabella FILM.

PAYMENT

7 colonne

Registra i pagamenti effettuati dai clienti per i noleggi, con importo, data e riferimenti al cliente, al dipendente e al noleggio. È collegata a RENTAL, CUSTOMER e STAFF.

RENTAL

7 colonne

Documenta i noleggi dei film, con data di noleggio, data di restituzione, cliente, dipendente e copia fisica noleggiata. È collegata a STAFF, INVENTORY e CUSTOMER e referenziata da PAYMENT.

STAFF

11 colonne

Rappresenta i dipendenti dei negozi di noleggio, con dati anagrafici, indirizzo, email, negozio di appartenenza, credenziali di accesso e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT, RENTAL e STORE.

STORE

4 colonne

Identifica i negozi di noleggio film, con il direttore responsabile e l'indirizzo della sede. È collegata a STAFF e ADDRESS e referenziata da CUSTOMER, INVENTORY e STAFF.

Modej

Report Tecnico Completo

database

Progetto: Production environment

Software: Software

Versione DB

v2

Tipo DB

mysql

Tabelle

16

Processi

10

FK manuali

2

Complessità

67/100

Cliente: Vivaldi

Data: 14 settembre 2026

1. Executive Summary

67

Indice di Complessità

Complessità media — sistema strutturato



16

Tabelle DB

2

FK manuali

3

Tabelle Orfane

10

Processi Mappati

229

Voci Glossario

13

Tab. Analizzate

Alert e Raccomandazioni

● 3 tabelle orfane — mai referenziate nel codice

2. Schema ER — Tabelle per Area

— SENZA AREA (16)

ACTOR 4 col.	ADDRESS 9 col.	CATEGORY 3 col.
CITY 4 col.	COUNTRY 3 col.	CUSTOMER 9 col.
FILM 13 col.	FILM_ACTOR 3 col.	FILM_CATEGORY 3 col.
FILM_TEXT 3 col.	INVENTORY 4 col.	LANGUAGE 3 col.
PAYMENT 7 col.	RENTAL 7 col.	STAFF 11 col.
STORE 4 col.		

Punto rosso = tabella orfana (mai referenziata nel codice sorgente)

3. Glossario Dati

ACTOR

4 colonne

Rappresenta gli attori che recitano nei film del catalogo, con nome e cognome. Ogni attore può essere collegato a più film tramite la tabella FILM_ACTOR.

ADDRESS

9 colonne

Contiene gli indirizzi civici utilizzati da clienti, dipendenti e negozi, con dettagli come via, distretto, città, codice postale, telefono e posizione geografica. È collegata alla tabella CITY e referenziata da CUSTOMER, STAFF e STORE.

CATEGORY

3 colonne

Definisce le categorie o generi in cui i film sono classificati, come azione, commedia o drammatico. Ogni categoria può essere associata a più film tramite la tabella FILM_CATEGORY.

CITY

4 colonne

Rappresenta le città in cui risiedono gli indirizzi, con il nome della città e il paese di appartenenza. È collegata alla tabella COUNTRY e referenziata da ADDRESS.

COUNTRY

3 colonne

Identifica i paesi del mondo utilizzati per localizzare città e indirizzi. È referenziata dalla tabella CITY.

CUSTOMER

9 colonne

Rappresenta i clienti del noleggio film, con dati anagrafici, email, negozio di riferimento, indirizzo e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT e RENTAL.

FILM

13 colonne

Descrive i film disponibili a catalogo, con titolo, descrizione, anno di uscita, lingua, durata del noleggio, tariffa, costo di sostituzione, classificazione e caratteristiche speciali. È collegata a LANGUAGE e referenziata da FILM_ACTOR, FILM_CATEGORY, INVENTORY e FILM_TEXT.

FILM_ACTOR

3 colonne

Associa gli attori ai film in cui recitano, gestendo la relazione multi-a-molti tra ACTOR e FILM.

FILM_CATEGORY

3 colonne

Associa i film alle categorie di appartenenza, gestendo la relazione multi-a-molti tra FILM e CATEGORY.

FILM_TEXT

3 colonne

Contiene una versione testuale semplificata delle informazioni del film, con titolo e descrizione, utile per ricerche o visualizzazioni. È collegata alla tabella FILM.

INVENTORY

4 colonne

Rappresenta le copie fisiche dei film disponibili presso ciascun negozio, con un identificativo univoco per ogni copia. È collegata a STORE e FILM e referenziata da RENTAL.

LANGUAGE

3 colonne

Definisce le lingue disponibili per i film, utilizzate sia come lingua principale che come lingua originale. È referenziata dalla tabella FILM.

PAYMENT

7 colonne

Registra i pagamenti effettuati dai clienti per i noleggi, con importo, data e riferimenti al cliente, al dipendente e al noleggio. È collegata a RENTAL, CUSTOMER e STAFF.

3. Glossario Dati (continua)

RENTAL	7 colonne
Documenta i noleggi dei film, con data di noleggio, data di restituzione, cliente, dipendente e copia fisica noleggiata. È collegata a STAFF, INVENTORY e CUSTOMER e referenziata da PAYMENT.	
STAFF	11 colonne
Rappresenta i dipendenti dei negozi di noleggio, con dati anagrafici, indirizzo, email, negozio di appartenenza, credenziali di accesso e stato di attività. È collegata a STORE e ADDRESS e referenziata da PAYMENT, RENTAL e STORE.	
STORE	4 colonne
Identifica i negozi di noleggio film, con il direttore responsabile e l'indirizzo della sede. È collegata a STAFF e ADDRESS e referenziata da CUSTOMER, INVENTORY e STAFF.	

4. Analisi Codice -> DB

Top 13 Tabelle per Utilizzo

CITY		3	I1	U1
FILM_ACTOR		3	I1	U1
COUNTRY		3	I1	U1
ADDRESS		3	I1	U1
CUSTOMER		3	I1	U1
CATEGORY		3	I1	U1
FILM		3	I1	U1
LANGUAGE		2	I1	U1
FILM_CATEGORY		2	I1	U1
FILM_TEXT		2	I1	U1
INVENTORY		2	I1	U1
ACTOR		1	U1	
STAFF		1	S1	

● **Tabelle Orfane (3)**

Queste tabelle non sono mai referenziate nel codice sorgente:

PAYMENT · RENTAL · STORE

5. Mappa Processi Aziendali

Anagrafica Geografica

- **Gestione Nazioni**
uCountryDAL.pas
- **Gestione Città**
uCityDAL.pas
- **Gestione Indirizzi**
uAddressDAL.pas

Anagrafica Clienti

- **Gestione Dati Cliente**
uCustomerDAL.pas

Catalogo Film

- **Gestione Anagrafica Film**
uFilmDAL.pas
- **Gestione Testi Film**
uFilmTextDAL.pas
- **Associazione Attori-Film**
uFilmActorDAL.pas
- **Associazione Film-Categorie**
uFilmCategoryDAL.pas

Anagrafica Attori

- **Consultazione Attori**
uActorDAL.pas

Anagrafica Categorie

- **Gestione Categorie**
uCategoryDAL.pas

Anagrafica Lingue

- **Gestione Lingue**
uLanguageDAL.pas

Gestione Inventario

- **Gestione Copie di Magazzino**
uInventoryDAL.pas

5. Mappa Processi (continua)

Gestione Noleggi

- Consultazione Noleggi

uRentalDAL.pas

Gestione Pagamenti

- Consultazione Pagamenti

uPaymentDAL.pas

Gestione Risorse Umane e Negozi

- Consultazione Anagrafica Dipendenti

uStaffDAL.pas

- Consultazione Negozi

uStoreDAL.pas

6. FK Manuali (PseudoFK)

Tabella Sorgente	Campo	->	Tabella Dest.	Campo Dest.
CATEGORY	CATEGORY_ID	->	FILM	ORIGINAL_LANGUAGE_ID
FILM_TEXT	FILM_ID	->	FILM	FILM_ID

7. Appendice Tecnica — Struttura Completa

ACTOR		4 colonne
ACTOR_ID	SMALLINT	
FIRST_NAME	VARCHAR(45)	
LAST_NAME	VARCHAR(45)	
LAST_UPDATE	TIMESTAMP	

ADDRESS		9 colonne
ADDRESS_ID	SMALLINT	
ADDRESS	VARCHAR(50)	
ADDRESS2	VARCHAR(50)	
DISTRICT	VARCHAR(20)	
CITY_ID	SMALLINT	
POSTAL_CODE	VARCHAR(10)	
PHONE	VARCHAR(20)	
LOCATION	GEOMETRY	
LAST_UPDATE	TIMESTAMP	

CATEGORY		3 colonne
CATEGORY_ID	TINYINT	
NAME	VARCHAR(25)	
LAST_UPDATE	TIMESTAMP	

CITY		4 colonne
CITY_ID	SMALLINT	
CITY	VARCHAR(50)	
COUNTRY_ID	SMALLINT	
LAST_UPDATE	TIMESTAMP	

COUNTRY		3 colonne
COUNTRY_ID	SMALLINT	
COUNTRY	VARCHAR(50)	
LAST_UPDATE	TIMESTAMP	

CUSTOMER		9 colonne
CUSTOMER_ID	SMALLINT	
STORE_ID	TINYINT	
FIRST_NAME	VARCHAR(45)	
LAST_NAME	VARCHAR(45)	
EMAIL	VARCHAR(50)	
ADDRESS_ID	SMALLINT	
ACTIVE	BOOLEAN	
CREATE_DATE	DATETIME	
LAST_UPDATE	TIMESTAMP	

7. Appendice Tecnica (continua)

FILM		13 colonne
FILM_ID	SMALLINT	
TITLE	VARCHAR(128)	
DESCRIPTION	TEXT	
RELEASE_YEAR	YEAR	
LANGUAGE_ID	TINYINT	
ORIGINAL_LANGUAGE_ID	TINYINT	
RENTAL_DURATION	TINYINT	
RENTAL_RATE	DECIMAL(4,2)	
LENGTH	SMALLINT	
REPLACEMENT_COST	DECIMAL(5,2)	
RATING	ENUM('G','PG','PG-13','R','NC-17')	
SPECIAL_FEATURES	SET('TRAILERS','COMMENTARIES','DELETED SCENES','BEHIND THE S	
LAST_UPDATE	TIMESTAMP	

FILM_ACTOR		3 colonne
ACTOR_ID	SMALLINT	
FILM_ID	SMALLINT	
LAST_UPDATE	TIMESTAMP	

FILM_CATEGORY		3 colonne
FILM_ID	SMALLINT	
CATEGORY_ID	TINYINT	
LAST_UPDATE	TIMESTAMP	

FILM_TEXT		3 colonne
FILM_ID	SMALLINT	
TITLE	VARCHAR(255)	
DESCRIPTION	TEXT	

INVENTORY		4 colonne
INVENTORY_ID	MEDIUMINT	
FILM_ID	SMALLINT	
STORE_ID	TINYINT	
LAST_UPDATE	TIMESTAMP	

LANGUAGE		3 colonne
LANGUAGE_ID	TINYINT	
NAME	CHAR(20)	
LAST_UPDATE	TIMESTAMP	

7. Appendice Tecnica (continua)

PAYMENT

orfana

7 colonne

PAYMENT_ID	SMALLINT
CUSTOMER_ID	SMALLINT
STAFF_ID	TINYINT
RENTAL_ID	INT
AMOUNT	DECIMAL(5,2)
PAYMENT_DATE	DATETIME
LAST_UPDATE	TIMESTAMP

RENTAL

orfana

7 colonne

RENTAL_ID	INT
RENTAL_DATE	DATETIME
INVENTORY_ID	MEDIUMINT
CUSTOMER_ID	SMALLINT
RETURN_DATE	DATETIME
STAFF_ID	TINYINT
LAST_UPDATE	TIMESTAMP

STAFF

11 colonne

STAFF_ID	TINYINT
FIRST_NAME	VARCHAR(45)
LAST_NAME	VARCHAR(45)
ADDRESS_ID	SMALLINT
PICTURE	BLOB
EMAIL	VARCHAR(50)
STORE_ID	TINYINT
ACTIVE	BOOLEAN
USERNAME	VARCHAR(16)
PASSWORD	VARCHAR(40)
LAST_UPDATE	TIMESTAMP

STORE

orfana

4 colonne

STORE_ID	TINYINT
MANAGER_STAFF_ID	TINYINT
ADDRESS_ID	SMALLINT
LAST_UPDATE	TIMESTAMP

AREE & COLORI

Cerca tabella...

16 / 16 assegnate

Tutte

Non assegnate

Filtra area...

+ Nuova area personale

ORDINA	Predefinito	AREE & COLORI			
		ACTOR	ADDRESS	CATEGORY	CITY
		Products	Administration	Products	Administration
Rimuovi colore					
		COUNTRY	CUSTOMER	FILM	FILM_ACTOR
		FILM_CATEGORY	FILM_TEXT	INVENTORY	LANGUAGE
		PAYMENT	RENTAL	STAFF	STORE
		SICUREZZA	SISTEMA	REPORT	QUALITÀ
		SPEDIZIONI	ASSISTENZA	CUCITURA	ALMACEN
		CONTABILIDAD	PRODUCTS	UNKNOWN	ADMINISTRATION

Production environment

database v1



AnalisiDB — database v1

Analisi strutturale profonda dello schema del tuo database.

16

89

22

tabelle

colonne

relazioni



Chiavi Primarie

OK · 100

Ogni tabella dovrebbe avere una chiave primaria che identifica univocamente le righe.

⊙ Nessun problema rilevato in quest'area.



Foreign Key

OK · 94

Le relazioni tra tabelle vanno dichiarate con foreign key per garantire integrità.

⚠ 1 colonne sembrano riferimenti ma non hanno una FK definita: ADDRESS.POSTAL_CODE



Indici

Male · 55

Le colonne usate nei JOIN e nelle ricerche vanno indicizzate per le prestazioni.

⚠ 22 colonne FK dovrebbero avere un indice per prestazioni ottimali nei JOIN.



Tipi di Dato

OK · 96

Scegliere il tipo giusto (DATE, DECIMAL, VARCHAR dimensionato) migliora spazio e correttezza.

⚠ FILM_TEXT.TITLE — VARCHAR(255) generico — dimensiona in base al contenuto reale



Naming

OK · 100

Una nomenclatura coerente rende lo schema leggibile e manutenibile.

⊙ Nessun problema rilevato in quest'area.



Normalizzazione

OK · 100

Evitare tabelle troppo larghe o campi ripetuti riduce ridondanza e anomalie.

⊙ Nessun problema rilevato in quest'area.



Audit & Tracciabilità

Male · 50

Campi come created_at/updated_at aiutano a tracciare le modifiche nel tempo.

⚠ Solo il 0% delle tabelle ha campi di audit (created_at/updated_at). Utili per tracciabilità.

```
-- =====
-- MODEL AnalisiDB – Script di correzione
-- Database: database v1
-- Generato: 14/09/2026 14:06
-- ATTENZIONE: rivedi ogni statement prima di eseguirlo.
-- Esegui su un ambiente di test, mai in produzione senza backup.
-- =====

-- Possibile FK mancante su ADDRESS.POSTAL_CODE – verifica la tabella di destinazione:
-- ALTER TABLE `ADDRESS` ADD CONSTRAINT `fk_ADDRESS_POSTAL_CODE` FOREIGN KEY (`POSTAL_CODE`) REFERENCES `<tabella_target>`(`id`);
CREATE INDEX `idx_ADDRESS_CITY_ID` ON `ADDRESS`(`CITY_ID`);
CREATE INDEX `idx_CITY_COUNTRY_ID` ON `CITY`(`COUNTRY_ID`);
CREATE INDEX `idx_CUSTOMER_ADDRESS_ID` ON `CUSTOMER`(`ADDRESS_ID`);
CREATE INDEX `idx_CUSTOMER_STORE_ID` ON `CUSTOMER`(`STORE_ID`);
CREATE INDEX `idx_FILM_LANGUAGE_ID` ON `FILM`(`LANGUAGE_ID`);
CREATE INDEX `idx_FILM_ORIGINAL_LANGUAGE_ID` ON `FILM`(`ORIGINAL_LANGUAGE_ID`);
CREATE INDEX `idx_FILM_ACTOR_ACTOR_ID` ON `FILM_ACTOR`(`ACTOR_ID`);
CREATE INDEX `idx_FILM_ACTOR_FILM_ID` ON `FILM_ACTOR`(`FILM_ID`);
CREATE INDEX `idx_FILM_CATEGORY_CATEGORY_ID` ON `FILM_CATEGORY`(`CATEGORY_ID`);
CREATE INDEX `idx_FILM_CATEGORY_FILM_ID` ON `FILM_CATEGORY`(`FILM_ID`);
CREATE INDEX `idx_INVENTORY_FILM_ID` ON `INVENTORY`(`FILM_ID`);
CREATE INDEX `idx_INVENTORY_STORE_ID` ON `INVENTORY`(`STORE_ID`);
CREATE INDEX `idx_PAYMENT_CUSTOMER_ID` ON `PAYMENT`(`CUSTOMER_ID`);
CREATE INDEX `idx_PAYMENT_RENTAL_ID` ON `PAYMENT`(`RENTAL_ID`);
CREATE INDEX `idx_PAYMENT_STAFF_ID` ON `PAYMENT`(`STAFF_ID`);
CREATE INDEX `idx_RENTAL_CUSTOMER_ID` ON `RENTAL`(`CUSTOMER_ID`);
CREATE INDEX `idx_RENTAL_INVENTORY_ID` ON `RENTAL`(`INVENTORY_ID`);
CREATE INDEX `idx_RENTAL_STAFF_ID` ON `RENTAL`(`STAFF_ID`);
CREATE INDEX `idx_STAFF_ADDRESS_ID` ON `STAFF`(`ADDRESS_ID`);
CREATE INDEX `idx_STAFF_STORE_ID` ON `STAFF`(`STORE_ID`);
```

ⓘ Questo script è un punto di partenza. Rivedi ogni statement, completa i segnaposto (tabelle target), ed esegui sempre prima su un backup o ambiente di test.

DATABASE DEL PROGETTO

database

MYSQL · 32 tab

2 FK

FK Manuali — database

MYSQL · 2 FK manuali definite

schema_version

v2 — 06/09/2026 12:08 (most_recent)

Aggiungi FK manuale

TABELLA FROM *

COLONNA FROM *

TABELLA TO *

COLONNA TO *

— tabella —

— colonna —

— tabella —

— colonna —

NOME FK (OPZIONALE)

es. FK_ORDINI_CLIENTI

ON DELETE

RESTRICT

ON UPDATE

RESTRICT

NOTE

Motivazione o documentazione della relazione

INSERITA DA

es. mario.rossi

+ Aggiungi FK

#	FROM	TO	ON DEL	ON UPD	NOME / NOTE	DATA	
17256	CATEGORY CATEGORY_ID	FILM ORIGINAL_LANGUAGE_ID	RESTRICT	RESTRICT		10/09/2026	
17257	FILM_TEXT FILM_ID	FILM FILM_ID	RESTRICT	RESTRICT		10/09/2026	

Progetti > Production environment > Storico / Changelog

Production environment

Esporta Markdown

VERSIONE BASE (DA)

v1 — database — 07/08/2026

VERSIONE NUOVA (A)

v2 — database — 06/09/2026

STORICO UPLOAD

v2 — database

NUOVA

06/09/2026 12:08

v1 — database

BASE

07/08/2026 18:23

0	0	16	0
TABELLE AGGIUNTE	TABELLE RIMOSSE	TABELLE MODIFICATE	INVARIATE

Tabelle modificate 16

ACTOR Tipo modificato: ACTOR_ID (SMALLINT)
ADDRESS Campo aggiunto: LOCATION Tipo modificato: ADDRESS_ID (SMALLINT) Tipo modificato: CITY_ID (SMALLINT)
CATEGORY Tipo modificato: CATEGORY_ID (TINYINT)
CITY Tipo modificato: CITY_ID (SMALLINT) Tipo modificato: COUNTRY_ID (SMALLINT)
COUNTRY Tipo modificato: COUNTRY_ID (SMALLINT)
CUSTOMER Tipo modificato: CUSTOMER_ID (SMALLINT) Tipo modificato: STORE_ID (TINYINT) Tipo modificato: ADDRESS_ID (SMALLINT) Tipo modificato: ACTIVE (BOOLEAN)
FILM Tipo modificato: FILM_ID (SMALLINT) Tipo modificato: RELEASE_YEAR (YEAR) Tipo modificato: LANGUAGE_ID (TINYINT) Tipo modificato: ORIGINAL_LANGUAGE_ID (TINYINT) Tipo modificato: RENTAL_DURATION (TINYINT) Tipo modificato: LENGTH (SMALLINT)
FILM_ACTOR Tipo modificato: ACTOR_ID (SMALLINT) Tipo modificato: FILM_ID (SMALLINT)
FILM_CATEGORY Tipo modificato: FILM_ID (SMALLINT) Tipo modificato: CATEGORY_ID (TINYINT)
FILM_TEXT Tipo modificato: FILM_ID (SMALLINT)
INVENTORY Tipo modificato: INVENTORY_ID (MEDIUMINT)

- Tipo modificato: FILM_ID (SMALLINT)
- Tipo modificato: STORE_ID (TINYINT)

LANGUAGE

- Tipo modificato: LANGUAGE_ID (TINYINT)

PAYMENT

- Tipo modificato: PAYMENT_ID (SMALLINT)
- Tipo modificato: CUSTOMER_ID (SMALLINT)
- Tipo modificato: STAFF_ID (TINYINT)
- Tipo modificato: RENTAL_ID (INT)

RENTAL

- Tipo modificato: RENTAL_ID (INT)
- Tipo modificato: INVENTORY_ID (MEDIUMINT)
- Tipo modificato: CUSTOMER_ID (SMALLINT)
- Tipo modificato: STAFF_ID (TINYINT)

STAFF

- Tipo modificato: STAFF_ID (TINYINT)
- Tipo modificato: ADDRESS_ID (SMALLINT)
- Tipo modificato: STORE_ID (TINYINT)
- Tipo modificato: ACTIVE (BOOLEAN)

STORE

- Tipo modificato: STORE_ID (TINYINT)
- Tipo modificato: MANAGER_STAFF_ID (TINYINT)
- Tipo modificato: ADDRESS_ID (SMALLINT)

GDPR & Dati Sensibili

Production environment — database — generato il 14/09/2026 13:57

5

TABELLE COINVOLTE

15

CAMPI INDIVIDUATI

ACTOR

Campo	Categoria	Rischio	Nota
FIRST_NAME	 Anagrafica	BASSO	Nome/Cognome
LAST_NAME	 Anagrafica	BASSO	Nome/Cognome

ADDRESS

Campo	Categoria	Rischio	Nota
ADDRESS_ID	 Contatto	MEDIO	Indirizzo fisico
ADDRESS	 Contatto	MEDIO	Indirizzo fisico
PHONE	 Contatto	MEDIO	Numero di telefono

CUSTOMER

Campo	Categoria	Rischio	Nota
FIRST_NAME	 Anagrafica	BASSO	Nome/Cognome
LAST_NAME	 Anagrafica	BASSO	Nome/Cognome
EMAIL	 Contatto	MEDIO	Indirizzo email
ADDRESS_ID	 Contatto	MEDIO	Indirizzo fisico

STAFF

Campo	Categoria	Rischio	Nota
FIRST_NAME	 Anagrafica	BASSO	Nome/Cognome
LAST_NAME	 Anagrafica	BASSO	Nome/Cognome
ADDRESS_ID	 Contatto	MEDIO	Indirizzo fisico

EMAIL	 Contatto	MEDIO	Indirizzo email
PASSWORD	 Credenziali	ALTO	Credenziale o segreto (sicurezza)

STORE

Campo	Categoria	Rischio	Nota
ADDRESS_ID	 Contatto	MEDIO	Indirizzo fisico

⚠️ Questo report è un supporto automatico basato sui NOMI di tabelle e colonne, non sostituisce una valutazione legale o un'analisi manuale dei dati reali. Campi con nomi generici (es. CAMPO_X) che contengono dati sensibili non vengono rilevati da questo strumento.



MODEJ — Rapporto di Analisi del Codice



Sintesi Esecutiva

- **Linguaggio:** Object Pascal (Delphi), sintassi moderna con `class function ... static`, `TBytes`, `System.Variants`. Compatibile con **Delphi 10.x / 11.x (Alexandria/Athens)**.
- **Framework di accesso ai dati:** FireDAC (`TFDConnection`, `TFDQuery`, `TFDMemTable`).
- **Database target:** MySQL — schema di esempio **Sakila**.
- **File analizzati:** 16 unità DAL (`uActorDAL`, `uAddressDAL`, `uCategoryDAL`, `uCityDAL`, `uCountryDAL`, `uCustomerDAL`, `uFilmActorDAL`, `uFilmCategoryDAL`, `uFilmDAL`, `uFilmTextDAL`, `uInventoryDAL`, `uLanguageDAL`, `uPaymentDAL`, `uRentalDAL`, `uStaffDAL`, `uStoreDAL`).
- **Verdetto complessivo:** ● **Attenzione** — codice funzionante e ben strutturato, ma con problemi sistemici di gestione transazionale, gestione errori e sicurezza (password in chiaro, assenza di logging).

Top 3 problemi urgenti

- ● **Nessuna gestione transazionale** su operazioni multi-statement (`Insert` + `SELECT LAST_INSERT_ID()`): rischio di ID errati in concorrenza e di dati orfani in caso di errore.
- ● **`LAST_INSERT_ID()` recuperato con una query separata** invece di `TFDQuery.LastAutoGenerateValue` o `INSERT ... RETURNING / SELECT LAST_INSERT_ID()` nella stessa sessione: race condition in ambienti multi-utente.
- ● **Nessuna gestione delle eccezioni:** ogni `try..finally` libera la query ma non cattura né logga gli errori; le eccezioni FireDAC risalgono al chiamante senza contesto.



Punti di Forza

Nome	Descrizione	File / Funzione
Uso corretto dei parametri SQL	Tutte le query usano parametri nominati (<code>:nome</code>), eliminando di fatto il rischio di SQL injection.	Tutti i file, tutte le funzioni
Pattern DAL coerente	Struttura uniforme: costante <code>SQL_FIELDS</code> , metodi <code>class function ... static</code> , record di trasferimento tipizzati.	Tutti i file
Gestione NULL con Variant	Uso corretto di <code>Variant</code> per colonne nullable (<code>address2</code> , <code>postal_code</code> , <code>email</code> , <code>return_date</code> , <code>rating</code> , ecc.).	<code>uAddressDAL</code> , <code>uCustomerDAL</code> , <code>uFilmDAL</code> , <code>uRentalDAL</code> , <code>uPaymentDAL</code>
Dataset scollegati	<code>GetAll</code> / <code>GetByX</code> restituiscono <code>TFDMemTable</code> di proprietà del chiamante, evitando connessioni aperte.	Tutti i file, metodi <code>GetAll</code> / <code>GetByX</code>
Documentazione XML	Ogni metodo pubblico è documentato con <code><summary></code> in stile XMLDoc.	Tutti i file
Ottimizzazione BLOB	<code>picture</code> escluso dai dataset di massa, con metodo dedicato <code>GetPicture</code> .	<code>uStaffDAL</code>
Uso di <code>RowsAffected</code>	<code>Update</code> / <code>Delete</code> restituiscono correttamente <code>True</code> / <code>False</code> in base alle righe modificate.	Tutti i file, <code>Update</code> / <code>Delete</code>

● Problemi Critici

● Recupero di `LAST_INSERT_ID()` in una query separata (race condition)

- **File/Funzione:** `uAddressDAL.Insert` , `uCategoryDAL.Insert` , `uCityDAL.Insert` , `uCountryDAL.Insert` , `uCustomerDAL.Insert` , `uFilmDAL.Insert` , `uInventoryDAL.Insert` , `uLanguageDAL.Insert`
- **Problema:** dopo `LQuery.ExecSQL` , viene eseguito `AConnection.ExecSQLScalar('SELECT LAST_INSERT_ID()')` come **statement separato**. Se la connessione è condivisa o se tra le due chiamate un altro thread esegue un `INSERT` sulla stessa connessione, l'ID restituito può appartenere a un'altra riga.
- **Rischio in produzione:** corruzione logica dei dati (record collegati all'ID sbagliato), particolarmente grave su `customer` , `address` , `film` .
- **Codice attuale:**

```
LQuery.ExecSQL;  
Result := Integer(AConnection.ExecSQLScalar('SELECT LAST_INSERT_ID()'));
```

- **Soluzione:**

```
// Opzione A: usare la proprietà LastAutoGenerateValue di FireDAC  
LQuery.ExecSQL;  
Result := LQuery.LastAutoGenerateValue; // FireDAC popola automaticamente l'ID  
  
// Opzione B (equivalente, esplicita): recuperare l'ID nella stessa query  
LQuery.SQL.Text :=  
    'INSERT INTO address (address, address2, district, city_id, postal_code, phone) ' +  
    'VALUES (:address, :address2, :district, :city_id, :postal_code, :phone); ' +  
    'SELECT LAST_INSERT_ID() AS new_id';  
LQuery.Open;  
Result := LQuery.FieldByName('new_id').AsInteger;
```

- **Assenza di gestione transazionale su operazioni composite**

- **File/Funzione:** tutti i metodi `Insert` che eseguono due statement (INSERT + SELECT ID), e in generale tutte le operazioni che il chiamante potrebbe voler raggruppare.
- **Problema:** nessuna unità espone `StartTransaction` / `Commit` / `Rollback`. Se il chiamante esegue `Insert` su `address` e poi `Insert` su `customer`, un errore sul secondo lascia un indirizzo orfano.
- **Rischio in produzione:** dati inconsistenti, violazioni di integrità referenziale, impossibilità di rollback.
- **Codice attuale** (esempio `uCustomerDAL.Insert`):

```
LQuery.ExecSQL;  
Result := Integer(AConnection.ExecSQLScalar('SELECT LAST_INSERT_ID()'));
```

- **Soluzione** — introdurre un helper transazionale riutilizzabile:

```

type
    TDbTransaction = class
    private
        FConn: TFDBConnection;
        FActive: Boolean;
    public
        constructor Create(AConn: TFDBConnection);
        procedure Commit;
        procedure Rollback;
        destructor Destroy; override;
    end;

constructor TDbTransaction.Create(AConn: TFDBConnection);
begin
    inherited Create;
    FConn := AConn;
    FConn.StartTransaction;
    FActive := True;
end;

procedure TDbTransaction.Commit;
begin
    if FActive then begin FConn.Commit; FActive := False; end;
end;

procedure TDbTransaction.Rollback;
begin
    if FActive then begin FConn.Rollback; FActive := False; end;
end;

destructor TDbTransaction.Destroy;
begin
    if FActive then FConn.Rollback; // rollback automatico in caso di eccezione
    inherited;
end;

```

Uso nel chiamante:

```

var Tx := TDbTransaction.Create(Conn);
try
    AddrID := TAddressDAL.Insert(Conn, ...);
    CustID := TCustomerDAL.Insert(Conn, ..., AddrID, ...);
    Tx.Commit;

```

```
except
    Tx.Rollback;
    raise;
end;
```

● Nessuna gestione delle eccezioni né logging

- **File/Funzione:** tutti i metodi di tutti i file.
- **Problema:** i blocchi `try..finally` liberano le risorse ma non catturano né arricchiscono le eccezioni FireDAC. Il chiamante riceve un `EFDDbEngineException` senza contesto (quale tabella, quale operazione, quali parametri).
- **Rischio in produzione:** diagnosi difficili, errori silenziosi nei log applicativi, impossibilità di distinguere errori transitori (deadlock, timeout) da errori permanenti (violazione FK).
- **Codice attuale:**

```
try
    LQuery.ExecSQL;
    Result := Integer(AConnection.ExecSQLScalar('SELECT LAST_INSERT_ID()'));
finally
    LQuery.Free;
end;
```

- **Soluzione** — introdurre un wrapper di logging e un'eccezione applicativa:

```
type
    EDALException = class(Exception)
    public
        constructor Create(const AOperation, ATable: string; AInner: Exception);
    end;

constructor EDALException.Create(const AOperation, ATable: string; AInner: Exception);
begin
    inherited CreateFmt('Errore DAL [%s su %s]: %s',
```

```

        [AOperation, ATable, AInner.Message]);
end;

// Nel metodo:
try
    LQuery.ExecSQL;
    Result := LQuery.LastAutoGenerateValue;
except
    on E: Exception do
        begin
            TAppLogger.Error('Insert address fallito', E);
            raise EDALError.Create('Insert', 'address', E);
        end;
    finally
        LQuery.Free;
    end;
end;

```

Password in chiaro nel record `TStaffRecord`

- **File/Funzione:** `uStaffDAL.TStaffRecord` , `uStaffDAL.GetByID` , `uStaffDAL.GetAll` , `uStaffDAL.GetByStoreID` .
- **Problema:** la colonna `password` viene letta e restituita al chiamante in chiaro, e inclusa nei dataset di massa (`GetAll` , `GetByStoreID`). Lo schema Sakila memorizza password in chiaro (limite del DB di esempio), ma il DAL non dovrebbe propagarle.
- **Rischio in produzione:** esposizione di credenziali in log, dump di memoria, UI, telemetria.
- **Codice attuale:**

```

SQL_FIELDS = 'staff_id, first_name, last_name, address_id, email, ' +
    'store_id, active, username, password, last_update';

```

- **Soluzione** — rimuovere `password` dai campi di default e fornire un metodo dedicato solo per l'autenticazione:

```

const
    SQL_FIELDS = 'staff_id, first_name, last_name, address_id, email, ' +
        'store_id, active, username, last_update';    // password rimossa

// Metodo dedicato, uso interno/autenticazione
class function TStaffDAL.VerifyCredentials(AConnection: TFDConnection;
    const AUserName, APasswordHash: string): Boolean; static;

```

E rimuovere Password da TStaffRecord .



TFDMemTable restituito senza gestione del ciclo di vita in caso di eccezione

- **File/Funzione:** tutti i metodi `GetAll` / `GetByX` che restituiscono `TFDMemTable` .
- **Problema:** se `Result.Data := LQuery.Data` solleva un'eccezione (es. memoria esaurita), il `TFDMemTable` creato va perso (memory leak) perché non è ancora assegnato a `Result` in modo sicuro.
- **Rischio in produzione:** memory leak progressivo in applicazioni long-running.
- **Codice attuale:**

```

Result := TFDMemTable.Create(nil);
Result.Data := LQuery.Data;

```

- **Soluzione:**

```

var
    LMemTable: TFDMemTable;
begin
    LMemTable := TFDMemTable.Create(nil);
    try
        LMemTable.Data := LQuery.Data;
        Result := LMemTable;
    except
        LMemTable.Free;
    end;
end;

```

```
        raise;  
    end;  
end;
```

● Problemi Minori

● `SQL_FIELDS` duplicato in ogni unità

- **Dove:** tutti i file — **Problema:** la stessa costante è ridefinita 16 volte, con rischio di divergenza. — **Soluzione:** centralizzare in `uSakilaConsts.pas` :

```
unit uSakilaConsts;  
  
interface  
  
const  
    SQL_ACTOR_FIELDS = 'actor_id, first_name, last_name, last_update';  
    SQL_ADDRESS_FIELDS = 'address_id, address, address2, district, city_id, postal_code, phone,  
    // ...
```

● `TFDQuery.Create(nil)` senza owner

- **Dove:** tutti i metodi — **Problema:** se `Free` non viene chiamato (eccezione prima del `try`), il componente resta orfano. — **Fix:** `TFDQuery.Create(AConnection)` per legare il ciclo di vita alla connessione, oppure mantenere `nil` ma assicurarsi che il `try` inizi subito dopo la `Create`.

● `FindByLastName` con confronto case-sensitive

- **Dove:** `uActorDAL.FindByLastName` — **Problema:** `WHERE last_name = :last_name` è case-sensitive su MySQL con collation binaria. — **Fix:** `WHERE last_name = :last_name COLLATE utf8mb4_general_ci` oppure documentare il comportamento.

● `GetByAddressID` in `uCustomerDAL` restituisce un dataset

- **Dove:** `uCustomerDAL.GetByAddressID` — **Problema:** la relazione `customer.address_id → address` è 1:N (più clienti possono condividere un indirizzo), ma il nome suggerisce un singolo risultato. — **Fix:** rinominare in `GetAllByAddressID` o documentare esplicitamente.

● `Update` su tabelle di associazione semanticamente ambiguo

- **Dove:** `uFilmActorDAL.Update` , `uFilmCategoryDAL.Update` — **Problema:** "aggiornare" una chiave composta è in realtà un `DELETE` + `INSERT` . Il metodo attuale può violare FK se la nuova coppia non esiste. — **Fix:** documentare o implementare come `DELETE` + `INSERT` in transazione.
- **GetPicture** non gestisce stream di grandi dimensioni
- **Dove:** `uStaffDAL.GetPicture` — **Problema:** carica l'intero BLOB in memoria. — **Fix:** esporre una variante che scriva direttamente su `TStream` fornito dal chiamante.
- **Result := True** incondizionato in **Insert** di tabelle associative
- **Dove:** `uFilmActorDAL.Insert` , `uFilmCategoryDAL.Insert` , `uFilmTextDAL.Insert` — **Problema:** restituisce sempre `True` anche se `ExecSQL` non ha inserito nulla (impossibile in pratica, ma semanticamente debole). — **Fix:** `Result := LQuery.RowsAffected > 0;`
- **Currency** per **rental_rate** e **replacement_cost**
- **Dove:** `uFilmDAL` — **Problema:** `Currency` ha 4 decimali fissi; MySQL `DECIMAL(4,2)` ne ha 2. Conversione implicita possibile. — **Fix:** usare `Double` o `TBCDField` esplicito.
- Nessun **ORDER BY** stabile in **GetAll** di tabelle associative
- **Dove:** `uFilmActorDAL.GetAll` ordina per `film_id`, `actor_id` (ok), ma `uFilmCategoryDAL.GetAll` per `film_id`, `category_id` (ok). Verificare che l'indice PK corrisponda all'ordine per evitare filesort. — **Fix:** allineare l' `ORDER BY` all'ordine della PK.

Database

File	Funzione	Tipo	Tabella	Problemi
<code>uAddressDAL</code>	<code>Insert</code>	<code>INSERT + SELECT</code>	<code>address</code>	Race condition su <code>LAST_INSERT_ID()</code> ; nessuna transazione
<code>uCategoryDAL</code>	<code>Insert</code>	<code>INSERT + SELECT</code>	<code>category</code>	Race condition su <code>LAST_INSERT_ID()</code>
<code>uCityDAL</code>	<code>Insert</code>	<code>INSERT + SELECT</code>	<code>city</code>	Race condition su <code>LAST_INSERT_ID()</code>
<code>uCountryDAL</code>	<code>Insert</code>	<code>INSERT + SELECT</code>	<code>country</code>	Race condition su <code>LAST_INSERT_ID()</code>

uCustomerDAL	Insert	INSERT + SELECT	customer	Race condition su LAST_INSERT_ID()
uFilmDAL	Insert	INSERT + SELECT	film	Race condition su LAST_INSERT_ID()
uInventoryDAL	Insert	INSERT + SELECT	inventory	Race condition su LAST_INSERT_ID()
uLanguageDAL	Insert	INSERT + SELECT	language	Race condition su LAST_INSERT_ID()
uActorDAL	GetAll	SELECT	actor	Nessun limite: su tabelle grandi può saturare la memoria
uFilmDAL	GetAll	SELECT	film	Idem; description TEXT può essere pesante
uPaymentDAL	GetAll	SELECT	payment	Tabella ad alto volume: nessuna paginazione
uRentalDAL	GetAll	SELECT	rental	Tabella ad alto volume: nessuna paginazione
uRentalDAL	GetOpenRentals	SELECT	rental	WHERE return_date IS NULL senza indice dedicato → full scan
uFilmActorDAL	GetByActorID	SELECT	film_actor	OK (indice PK su actor_id)
uFilmCategoryDAL	GetByCategoryID	SELECT *	film_category	OK (indice PK su category_id)
uStaffDAL	GetPicture	SELECT BLOB	staff	Carica BLOB intero in memoria

Nessuna SQL injection rilevata: tutte le query usano parametri nominati FireDAC.

Query lente potenziali:

- SELECT ... FROM rental WHERE return_date IS NULL — aggiungere indice parziale o indice su return_date .
- SELECT ... FROM payment ORDER BY payment_id senza LIMIT — paginare.
- SELECT ... FROM film ORDER BY film_id — su Sakila (~1000 righe) è ok, ma su scala reale serve paginazione.

Sicurezza

Aspetto	Stato	Note
Validazione input		Nessuna validazione di lunghezza/formato sui parametri stringa (es. <code>title</code> , <code>email</code>). FireDAC protegge dall'injection ma non dai dati malformati.
SQL injection		Tutte le query usano parametri nominati. Nessuna concatenazione di input utente.
Password / segreti		<code>password</code> di <code>staff</code> letta e restituita in chiaro, inclusa nei dataset di massa. Nessuna hashing, nessuna redazione.
Gestione errori		Nessun logging, nessuna eccezione applicativa: gli errori DB risalgono grezzi.
Transazioni		Nessuna gestione transazionale: rischio di dati inconsistenti.
Concorrenza		<code>LAST_INSERT_ID()</code> recuperato fuori contesto: race condition.
Esposizione BLOB		<code>GetPicture</code> carica l'intero BLOB in memoria senza limiti.
Connessione		La connessione è passata dal chiamante: nessun controllo che sia aperta/valida prima dell'uso.

Scheda di Valutazione

Categoria	/10
Qualità	7
Sicurezza	4
Prestazioni	6
Manutenibilità	6

Refactoring Prioritario

Riscrittura completa di `uAddressDAL` come modello di riferimento per tutte le altre unità, con: transazione, `LastAutoGenerateValue` , logging, gestione eccezioni, validazione input.

```
unit uAddressDAL;
```

interface

uses

System.SysUtils, System.Classes, System.Variants,
Data.DB, FireDAC.Comp.Client;

type

TAddressRecord = record

AddressID: Integer;

Address: string;

Address2: Variant;

District: string;

CityID: Integer;

PostalCode: Variant;

Phone: string;

LastUpdate: TDateTime;

end;

EDALException = class(Exception)

public

constructor Create(const AOperation, ATable: string; AInner: Exception);

end;

TAddressDAL = class

public

class function Insert(AConnection: TFDConnection; const AAddress,
ADistrict, APhone: string; ACityID: Integer;
const AAddress2: Variant; const APostalCode: Variant): Integer; static;

class function Update(AConnection: TFDConnection; AAddressID: Integer;
const AAddress, ADistrict, APhone: string; ACityID: Integer;
const AAddress2: Variant; const APostalCode: Variant): Boolean; static;

class function Delete(AConnection: TFDConnection;
AAddressID: Integer): Boolean; static;

class function GetByID(AConnection: TFDConnection; AAddressID: Integer;
out ARecord: TAddressRecord): Boolean; static;

class function GetAll(AConnection: TFDConnection): TFDMemTable; static;

class function GetByCityID(AConnection: TFDConnection;
ACityID: Integer): TFDMemTable; static;

end;

implementation

const

SQL_FIELDS = 'address_id, address, address2, district, city_id, ' +
'postal_code, phone, last_update';

{ EDALException }

```

constructor EDALError.Create(const AOperation, ATable: string;
    AInner: Exception);
begin
    inherited CreateFmt('Errore DAL [%s su %s]: %s',
        [AOperation, ATable, AInner.Message]);
end;

{ TAddressDAL }

class function TAddressDAL.Insert(AConnection: TFDConnection; const AAddress,
    ADistrict, APhone: string; ACityID: Integer; const AAddress2: Variant;
    const APostalCode: Variant): Integer;
var
    LQuery: TFDQuery;
begin
    if not AConnection.Connected then
        raise EDALError.Create('Insert', 'address',
            Exception.Create('Connessione non attiva'));

    if Trim(AAddress) = '' then
        raise EDALError.Create('Insert', 'address',
            Exception.Create('Il campo address è obbligatorio'));

    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.SQL.Text :=
            'INSERT INTO address (address, address2, district, city_id, postal_code, phone) ' +
            'VALUES (:address, :address2, :district, :city_id, :postal_code, :phone)';
        LQuery.ParamByName('address').AsString := AAddress;
        LQuery.ParamByName('address2').Value := AAddress2;
        LQuery.ParamByName('district').AsString := ADistrict;
        LQuery.ParamByName('city_id').AsInteger := ACityID;
        LQuery.ParamByName('postal_code').Value := APostalCode;
        LQuery.ParamByName('phone').AsString := APhone;
        LQuery.ExecSQL;
        Result := LQuery.LastAutoGenerateValue; // niente SELECT separata
    except
        on E: Exception do
            raise EDALError.Create('Insert', 'address', E);
    finally
        LQuery.Free;
    end;
end;

class function TAddressDAL.Update(AConnection: TFDConnection;
    AAddressID: Integer; const AAddress, ADistrict, APhone: string;
    ACityID: Integer; const AAddress2: Variant;
    const APostalCode: Variant): Boolean;
var

```

```

    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.SQL.Text :=
            'UPDATE address SET address = :address, address2 = :address2, ' +
            'district = :district, city_id = :city_id, postal_code = :postal_code, ' +
            'phone = :phone WHERE address_id = :address_id';
        LQuery.ParamByName('address').AsString := AAddress;
        LQuery.ParamByName('address2').Value := AAddress2;
        LQuery.ParamByName('district').AsString := ADistrict;
        LQuery.ParamByName('city_id').AsInteger := ACityID;
        LQuery.ParamByName('postal_code').Value := APostalCode;
        LQuery.ParamByName('phone').AsString := APhone;
        LQuery.ParamByName('address_id').AsInteger := AAddressID;
        LQuery.ExecSQL;
        Result := LQuery.RowsAffected > 0;
    except
        on E: Exception do
            raise EDALError.Create('Update', 'address', E);
    finally
        LQuery.Free;
    end;
end;

class function TAddressDAL.Delete(AConnection: TFDConnection;
    AAddressID: Integer): Boolean;
var
    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.SQL.Text := 'DELETE FROM address WHERE address_id = :address_id';
        LQuery.ParamByName('address_id').AsInteger := AAddressID;
        LQuery.ExecSQL;
        Result := LQuery.RowsAffected > 0;
    except
        on E: Exception do
            raise EDALError.Create('Delete', 'address', E);
    finally
        LQuery.Free;
    end;
end;

class function TAddressDAL.GetByID(AConnection: TFDConnection;
    AAddressID: Integer; out ARecord: TAddressRecord): Boolean;

```

```

var
  LQuery: TFDQuery;
begin
  LQuery := TFDQuery.Create(nil);
  try
    LQuery.Connection := AConnection;
    LQuery.SQL.Text := 'SELECT ' + SQL_FIELDS +
      ' FROM address WHERE address_id = :address_id';
    LQuery.ParamByName('address_id').AsInteger := AAddressID;
    LQuery.Open;
    Result := not LQuery.Eof;
    if Result then
      begin
        ARecord.AddressID := LQuery.FieldByName('address_id').AsInteger;
        ARecord.Address := LQuery.FieldByName('address').AsString;
        ARecord.Address2 := LQuery.FieldByName('address2').Value;
        ARecord.District := LQuery.FieldByName('district').AsString;
        ARecord.CityID := LQuery.FieldByName('city_id').AsInteger;
        ARecord.PostalCode := LQuery.FieldByName('postal_code').Value;
        ARecord.Phone := LQuery.FieldByName('phone').AsString;
        ARecord.LastUpdate := LQuery.FieldByName('last_update').AsDateTime;
      end;
    except
      on E: Exception do
        raise EDAException.Create('GetByID', 'address', E);
    finally
      LQuery.Free;
    end;
  end;

class function TAddressDAL.GetAll(AConnection: TFDConnection): TFDMemTable;
var
  LQuery: TFDQuery;
  LMemTable: TFDMemTable;
begin
  LQuery := TFDQuery.Create(nil);
  try
    LQuery.Connection := AConnection;
    LQuery.SQL.Text := 'SELECT ' + SQL_FIELDS +
      ' FROM address ORDER BY address_id';
    LQuery.Open;
    LMemTable := TFDMemTable.Create(nil);
    try
      LMemTable.Data := LQuery.Data;
      Result := LMemTable;
    except
      LMemTable.Free;
    end;
  end;
end;

```

```

        raise;
    end;
except
    on E: Exception do
        raise EDAException.Create('GetAll', 'address', E);
    finally
        LQuery.Free;
    end;
end;

class function TAddressDAL.GetByCityID(AConnection: TFDConnection;
    ACityID: Integer): TFDMemTable;
var
    LQuery: TFDQuery;
    LMemTable: TFDMemTable;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.SQL.Text := 'SELECT ' + SQL_FIELDS +
            ' FROM address WHERE city_id = :city_id ORDER BY address_id';
        LQuery.ParamByName('city_id').AsInteger := ACityID;
        LQuery.Open;
        LMemTable := TFDMemTable.Create(nil);
        try
            LMemTable.Data := LQuery.Data;
            Result := LMemTable;
        except
            LMemTable.Free;
            raise;
        end;
    except
        on E: Exception do
            raise EDAException.Create('GetByCityID', 'address', E);
        finally
            LQuery.Free;
        end;
    end;
end;

end.

```

Piano d'Azione

 Immediato (entro 1 settimana)

- Sostituire `AConnection.ExecSQLScalar('SELECT LAST_INSERT_ID()')` CON `LQuery.LastAutoGenerateValue` in tutti gli `Insert` (8 file).
- Rimuovere `password` da `TStaffRecord` e da `SQL_FIELDS` in `uStaffDAL` ; introdurre `VerifyCredentials` .
- Aggiungere `try..except` con `EDALException` in tutti i metodi pubblici.
- Correggere il memory leak potenziale in tutti i `GetAll` / `GetByX` (variabile locale + `try..except`).



Questo mese

- Introdurre `TDbTransaction` e documentare quali operazioni richiedono transazione.
- Centralizzare `SQL_FIELDS` in `uSakilaConsts.pas` .
- Aggiungere validazione input (lunghezza, formato email, range ID) nei metodi `Insert` / `Update` .
- Aggiungere paginazione (`LIMIT` / `OFFSET`) a `GetAll` di `payment` , `rental` , `film` .
- Aggiungere indice su `rental.return_date` per `GetOpenRentals` .
- Introdurre logging strutturato (es. `TAppLogger`).



3 mesi

- Refactoring verso un **repository generico** parametrizzato (`TDALRepository<T>`) per eliminare la duplicazione di 16 unità quasi identiche.
- Introdurre **unit test** con database Sakila in-memory (SQLite o MySQL containerizzato).
- Valutare migrazione a FireDAC `TFDTable` / `TFDQuery` con `CachedUpdates` per operazioni batch.
- Introdurre **cifratura/hashing** delle password a livello applicativo (bcrypt/argon2) e migrazione dello schema.
- Aggiungere **metriche** (tempo di esecuzione query, contatori errori) per il monitoraggio in produzione.



Note Finali

Il codice è **ben strutturato e leggibile**, con un pattern DAL coerente e un uso corretto dei parametri SQL che elimina di fatto il rischio di SQL injection. Tuttavia, presenta **tre debolezze sistemiche** che vanno affrontate prima di considerarlo pronto per la produzione:

- **Concorrenza:** il recupero di `LAST_INSERT_ID()` in una query separata è un anti-pattern classico che può causare corruzione dei dati in ambienti multi-utente. La correzione è banale (`LastAutoGenerateValue`) ma va applicata ovunque.
- **Transazioni:** l'assenza di gestione transazionale rende impossibile garantire l'integrità su operazioni composite. Va introdotto un helper riutilizzabile.
- **Sicurezza:** la propagazione della password in chiaro è inaccettabile in un'applicazione reale, anche se lo schema Sakila la memorizza così.

La **duplicazione massiccia** (16 unità quasi identiche) è il principale debito tecnico: un refactoring verso un repository generico ridurrebbe il codice di oltre il 70% e centralizzerebbe le correzioni. Consiglio di procedere prima con le correzioni puntuali (immediate), poi con il refactoring strutturale (3 mesi), mantenendo sempre una suite di test che copra i casi limite (NULL, ID inesistenti, violazioni FK).



MODEJ — Rapporto di Audit di Sicurezza



Sintesi Esecutiva

- Livello di rischio complessivo: ● ALTO

- Vulnerabilità per severità:

- ● Critiche: 1
- ● Alte: 3
- ● Medie: 4
- ● Basse: 3

- Top 3 rischi immediati:

1. **Esposizione di credenziali in chiaro** — la tabella `staff` viene letta includendo la colonna `password` (hash) in `GetByID`, `GetAll` e `GetByStoreID`, senza alcuna restrizione di accesso o mascheramento.

2. **Assenza totale di controllo di accesso/autorizzazione** — il DAL non applica alcun filtro di autorizzazione: chiunque abbia accesso alla connessione può leggere/scrivere/eliminare qualsiasi riga di qualsiasi tabella (IDOR/BOLA a livello applicativo).

3. **Assenza di gestione transazionale e di logging** — le operazioni di scrittura (`Insert` / `Update` / `Delete`) non sono transazionali né tracciate, consentendo manipolazioni silenziose e stati incoerenti (es. `LAST_INSERT_ID()` fuori contesto).

> Nota metodologica: il codice è un **Data Access Layer (DAL)** in Object Pascal (Delphi/FireDAC) per il database di esempio *sakila*. Le query usano parametri FireDAC (`:nome`), quindi non è presente **SQL Injection classica**. Le problematiche rilevate sono di natura architetturale, di gestione dei dati sensibili, di integrità e di robustezza.



Vulnerabilità Critiche (CVSS 9.0–10.0)

[MOD-DAL-001] Esposizione di credenziali di autenticazione tramite API di lettura

- Categoria OWASP: A02 — Cryptographic Failures / A01 — Broken Access Control

- **CVSS v3: 9.1** — `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`

- **File/Funzione:** `uStaffDAL.pas` → `TStaffDAL.GetByID` , `TStaffDAL.GetAll` , `TStaffDAL.GetByStoreID`
(costante `SQL_FIELDS` include `password`)
- **Descrizione:** La costante `SQL_FIELDS` include esplicitamente la colonna `password` della tabella `staff` . Tutte le funzioni di lettura (`GetByID` , `GetAll` , `GetByStoreID`) restituiscono quindi l'hash della password (o, in configurazioni non sicure, la password in chiaro) a **qualsiasi chiamante** che abbia accesso alla connessione. Non esiste alcun controllo di ruolo, né mascheramento, né separazione tra "dati anagrafici" e "dati di autenticazione". Inoltre il record `TStaffRecord` espone il campo `Password: Variant` pubblicamente, favorendo la propagazione del segreto verso UI, log e serializzazioni.
- **Proof of Concept:** un chiamante qualsiasi (es. una form di elenco dipendenti, un endpoint REST che serializza il `TFDMemTable`) ottiene l'hash delle password di tutti gli operatori:

```
// Chiunque possa invocare:
var
    LTable: TFDMemTable;
begin
    LTable := TStaffDAL.GetAll(FDConnection1); // contiene la colonna "password"
    // LTable ora espone username + hash password di TUTTI gli staff
    // Es. serializzazione JSON -> {"username":"Mike","password":"*A4B6157319038724E3560894F7F5
end;
```



- **Remediation:** rimuovere `password` da `SQL_FIELDS` e dal record pubblico; isolare l'accesso alle credenziali in una funzione dedicata, protetta e con logging, che restituisca solo l'hash per la verifica e mai verso la UI.

```
const
    // Nessuna password nei campi di lettura ordinaria
    SQL_FIELDS = 'staff_id, first_name, last_name, address_id, email, ' +
        'store_id, active, username, last_update';

type
    TStaffRecord = record
        StaffID: Integer;
        FirstName: string;
        LastName: string;
        AddressID: Integer;
```

```

    Email: Variant;
    StoreID: Integer;
    Active: Boolean;
    UserName: string;
    // Password RIMOSSA dal record pubblico
    LastUpdate: TDateTime;
end;

// Funzione dedicata, ad uso esclusivo del modulo di autenticazione,
// che restituisce SOLO l'hash e non è esposta alla UI.
class function TStaffDAL.GetPasswordHash(AConnection: TFDConnection;
    const AUserName: string; out AHash: string): Boolean;
var
    LQuery: TFDQuery;
begin
    Result := False;
    AHash := '';
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.SQL.Text :=
            'SELECT password FROM staff WHERE username = :username';
        LQuery.ParamByName('username').AsString := AUserName;
        LQuery.Open;
        if not LQuery.Eof and not LQuery.FieldByName('password').IsNull then
            begin
                AHash := LQuery.FieldByName('password').AsString;
                Result := True;
            end;
        finally
            LQuery.Free;
        end;
    end;
end;

```

Vulnerabilità Alte (CVSS 7.0–8.9)

[MOD-DAL-002] Assenza di controllo di accesso (IDOR/BOLA) a livello di DAL

- Categoria OWASP: A01 — Broken Access Control
- CVSS v3: 8.1 — CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N

- **File/Funzione:** tutti i moduli DAL (es. `uCustomerDAL.GetByID` , `uAddressDAL.Update/Delete` , `uPaymentDAL.GetByCustomerID` , `uRentalDAL.GetByCustomerID`)
- **Descrizione:** Le funzioni accettano un identificatore (`ACustomerID` , `AAddressID` , `APaymentID` , ...) e restituiscono/modificano la riga corrispondente **senza verificare che il chiamante sia autorizzato** su quella risorsa. Non esiste alcun concetto di "utente corrente", "negozio di appartenenza" o "proprietà della risorsa". Un cliente autenticato che conosca (o indovini) un `customer_id` / `payment_id` altrui può leggerne i dati; analogamente `Update` / `Delete` consentono modifiche/cancellazioni su risorse non proprie.
- **Proof of Concept:** enumerazione sequenziale degli ID per accedere ai dati di altri clienti:

```
// Un cliente autenticato come customer_id = 5 enumera gli altri:
for I := 1 to 1000 do
begin
    if TCustomerDAL.GetByID(FDConnection1, I, LRec) then
        // LRec contiene dati personali (email, indirizzo) di clienti NON autorizzati
end;
```

- **Remediation:** introdurre un contesto di autorizzazione e filtrare sempre per proprietario/tenant. Esempio con vincolo sul negozio/utente:

```
class function TCustomerDAL.GetByIDForUser(AConnection: TFDConnection;
    ARequestingStoreID, ACustomerID: Integer;
    out ARecord: TCustomerRecord): Boolean;
var
    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        // Il filtro store_id impedisce l'accesso cross-tenant
        LQuery.SQL.Text := 'SELECT ' + SQL_FIELDS +
            ' FROM customer WHERE customer_id = :customer_id AND store_id = :store_id';
        LQuery.ParamByName('customer_id').AsInteger := ACustomerID;
        LQuery.ParamByName('store_id').AsInteger := ARequestingStoreID;
        LQuery.Open;
        Result := not LQuery.EOF;
        if Result then
```

```

begin
    // ... popolamento record ...
end;
finally
    LQuery.Free;
end;
end;

```

> In alternativa (preferibile), applicare il controllo di autorizzazione in un livello di servizio prima di invocare il DAL, e usare **viste/ruoli MySQL** con `WHERE` implicito per tenant.

[MOD-DAL-003] Operazioni di scrittura non transazionali e recupero ID non atomico

- **Categoria OWASP:** A04 — Insecure Design / A08 — Software and Data Integrity Failures
- **CVSS v3: 7.5** — `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:H`
- **File/Funzione:** `uAddressDAL.Insert` , `uCategoryDAL.Insert` , `uCityDAL.Insert` , `uCountryDAL.Insert` , `uCustomerDAL.Insert` , `uFilmDAL.Insert` , `uInventoryDAL.Insert` , `uLanguageDAL.Insert`
- **Descrizione:** Ogni `Insert` esegue `ExecSQL` e poi una **seconda query separata** `SELECT LAST_INSERT_ID()` sulla stessa connessione, **senza transazione**. In ambiente concorrente, tra le due istruzioni un'altra sessione/thread può eseguire un proprio `INSERT` , e `LAST_INSERT_ID()` — pur essendo per-sessione in MySQL — può restituire un valore errato se la connessione è condivisa/pooled o se il driver riusa la sessione. Inoltre, se `ExecSQL` riesce ma `ExecSQLScalar` fallisce, la riga resta inserita senza che il chiamante ne conosca l'ID (stato incoerente). Manca qualsiasi `StartTransaction / Commit / Rollback` .
- **Proof of Concept:** due thread che inseriscono contemporaneamente su una connessione condivisa possono ricevere l'ID l'uno dell'altro:

```

// Thread A e Thread B condividono FDConnection1 (pooling/riuso sessione)
// A: INSERT ... ; B: INSERT ... ; A: SELECT LAST_INSERT_ID() -> potrebbe
// restituire l'ID di B se la sessione è stata riassegnata dal pool.

```

- **Remediation:** avvolgere l'operazione in una transazione e recuperare l'ID in modo atomico, oppure usare `RETURNING / INSERT ... ; SELECT LAST_INSERT_ID()` nella stessa unità transazionale con connessione dedicata.

```

class function TAddressDAL.Insert(AConnection: TFDConnection; const AAddress,
    ADistrict, APhone: string; ACityID: Integer; const AAddress2: Variant;
    const APostalCode: Variant): Integer;
var
    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        AConnection.StartTransaction;
        try
            LQuery.SQL.Text :=
                'INSERT INTO address (address, address2, district, city_id, postal_code, phone) ' +
                'VALUES (:address, :address2, :district, :city_id, :postal_code, :phone)';
            LQuery.ParamByName('address').AsString := AAddress;
            LQuery.ParamByName('address2').Value := AAddress2;
            LQuery.ParamByName('district').AsString := ADistrict;
            LQuery.ParamByName('city_id').AsInteger := ACityID;
            LQuery.ParamByName('postal_code').Value := APostalCode;
            LQuery.ParamByName('phone').AsString := APhone;
            LQuery.ExecSQL;
            // Recupero atomico nella stessa transazione
            Result := Integer(AConnection.ExecSQLScalar('SELECT LAST_INSERT_ID()'));
            AConnection.Commit;
        except
            AConnection.Rollback;
            raise;
        end;
    finally
        LQuery.Free;
    end;
end;

```

[MOD-DAL-004] Assenza di validazione degli input e di gestione degli errori

- **Categoria OWASP:** A03 — Injection (indiretta) / A04 — Insecure Design
- **CVSS v3:** 7.2 — CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:L

- **File/Funzione:** tutti i metodi `Insert / Update` (es. `uFilmDAL.Insert` , `uCustomerDAL.Insert` , `uAddressDAL.Insert`)
- **Descrizione:** I parametri stringa (`ATitle` , `AAddress` , `AEmail` , `ARating` , `ASpecialFeatures` , ...) vengono passati direttamente al database **senza validazione di lunghezza, formato o dominio**. Sebbene i parametri FireDAC prevengano l'SQL Injection, l'assenza di validazione consente: (a) **data injection** — valori fuori dominio per campi `ENUM / SET` (`rating` , `special_features`) che generano errori o comportamenti imprevisti; (b) **overflow/troncamento silenzioso** su colonne `CHAR(20)` (es. `language.name`); (c) **stored XSS** se i valori vengono poi renderizzati in UI senza escaping; (d) **denial of service** tramite payload enormi. Inoltre non c'è gestione delle eccezioni: un errore DB propaga uno stack trace potenzialmente informativo.
- **Proof of Concept:** inserimento di un valore fuori dominio o payload malevolo che verrà poi mostrato in UI:

```
// rating accetta solo 'G','PG','PG-13','R','NC-17' ma non viene validato
TFilmDAL.Insert(FDConnection1, 'Titolo', Null, Null, 1, Null, 3, 4.99,
    Null, 19.99, '<script>alert(1)</script>', Null);
// Il valore viene salvato e, se renderizzato in una WebView/HTML, esegue JS.
```

- **Remediation:** validare e normalizzare gli input prima dell'invio al DB, con whitelist per i campi a dominio chiuso, e gestire le eccezioni in modo sicuro.

```
const
    VALID_RATINGS: array[0..4] of string =
        ('G', 'PG', 'PG-13', 'R', 'NC-17');

function IsValidRating(const AValue: Variant): Boolean;
var
    S: string;
    I: Integer;
begin
    if VarIsNull(AValue) or VarIsClear(AValue) then
        Exit(True); // NULL ammesso
    S := VarToStr(AValue);
    for I := Low(VALID_RATINGS) to High(VALID_RATINGS) do
        if S = VALID_RATINGS[I] then
            Exit(True);
    Result := False;
end;
```

```
// Nell'Insert/Update:
if not IsValidRating(ARating) then
    raise EArgumentException.Create('Valore rating non valido.');
```

```
if Length(ATitle) > 255 then
    raise EArgumentException.Create('Titolo troppo lungo.');
```

● Vulnerabilità Medie (CVSS 4.0–6.9)

[MOD-DAL-005] Esposizione di dati personali tramite `GetAll` senza paginazione

- **Categoria OWASP:** A01 — Broken Access Control / A04 — Insecure Design
- **CVSS v3:** 5.3 — CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
- **File/Funzione:** `uCustomerDAL.GetAll` , `uPaymentDAL.GetAll` , `uRentalDAL.GetAll` , `uAddressDAL.GetAll`
- **Descrizione:** Le funzioni `GetAll` caricano l'intera tabella in memoria (`TFDMemTable`) senza paginazione né filtri. Su tabelle contenenti dati personali (clienti, pagamenti, noleggi) questo espone in blocco tutti i dati a qualsiasi chiamante e favorisce il data exfiltration massivo.
- **Remediation:** introdurre paginazione e filtri obbligatori:

```
class function TCustomerDAL.GetPage(AConnection: TFDCConnection;
    AOffset, ALimit: Integer): TFDMemTable;
begin
    // SELECT ... FROM customer ORDER BY customer_id LIMIT :limit OFFSET :offset
end;
```

[MOD-DAL-006] `Update` / `Delete` senza verifica di concorrenza ottimistica

- **Categoria OWASP:** A08 — Software and Data Integrity Failures
- **CVSS v3:** 5.0 — CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N
- **File/Funzione:** tutti i metodi `Update` / `Delete` (es. `uAddressDAL.Update` , `uCustomerDAL.Update` , `uFilmDAL.Update`)

- **Descrizione:** Le `UPDATE` / `DELETE` si basano solo sulla chiave primaria, senza confronto su `last_update` (versione). Due utenti concorrenti possono sovrascriversi a vicenda ("lost update") senza accorgersene. Il campo `last_update` esiste ma non viene mai usato per il controllo di concorrenza.
- **Remediation:** aggiungere il controllo di versione:

```
'UPDATE address SET ... WHERE address_id = :address_id AND last_update = :last_update'
// Se RowsAffected = 0 -> conflitto di concorrenza, segnalare al chiamante.
```

[MOD-DAL-007] `TFDMemTable` restituiti senza gestione del ciclo di vita / rischio di memory leak

- **Categoria OWASP:** A04 — Insecure Design
- **CVSS v3:** 4.3 — `CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:L`
- **File/Funzione:** tutti i metodi che restituiscono `TFDMemTable` (es. `uActorDAL.GetAll` , `uCityDAL.GetByCountryID`)
- **Descrizione:** Le funzioni creano `TFDMemTable.Create(nil)` e ne trasferiscono la proprietà al chiamante, ma **non esiste alcuna garanzia documentata né pattern RAIL** che ne assicuri la liberazione. In caso di eccezione dopo la creazione (es. `Result.Data := LQuery.Data` fallisce), il `TFDMemTable` non viene liberato → memory leak. Inoltre `Result.Data := LQuery.Data` copia i dati ma il comportamento in caso di dataset vuoto non è gestito esplicitamente.
- **Remediation:** usare `try/except` con liberazione esplicita e documentare chiaramente il contratto di proprietà:

```
Result := TFDMemTable.Create(nil);
try
    Result.Data := LQuery.Data;
except
    Result.Free;
    raise;
end;
```

[MOD-DAL-008] `GetPicture` carica BLOB in memoria senza limiti

- **Categoria OWASP:** A04 — Insecure Design

- **CVSS v3: 4.0** — `CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:L`
- **File/Funzione:** `uStaffDAL.GetPicture`
- **Descrizione:** La funzione carica l'intero BLOB `picture` in un `TMemoryStream` e poi in un `TBytes` senza alcun limite di dimensione. Un BLOB molto grande (o manipolato) può causare esaurimento di memoria (DoS locale). Inoltre `Move(LStream.Memory^, Result[0], LStream.Size)` non gestisce il caso `Size = 0` in modo esplicito (sebbene protetto da `if`).
- **Remediation:** imporre un limite massimo e usare streaming:

```
const
    MAX_PICTURE_SIZE = 5 * 1024 * 1024; // 5 MB
...
if LStream.Size > MAX_PICTURE_SIZE then
    raise EInvalidOp.Create('Immagine troppo grande.');
```

● Vulnerabilità Basse (CVSS 0.1–3.9)

- [MOD-DAL-009] Assenza di logging delle operazioni di scrittura (`uAddressDAL.Delete` , `uCustomerDAL.Delete` , ecc.) — Nessuna traccia di chi ha modificato/cancellato cosa e quando. *Fix:* introdurre un audit log applicativo o trigger MySQL di audit. (CVSS 3.1)
- [MOD-DAL-010] Messaggi di errore DB potenzialmente informativi — Le eccezioni FireDAC non gestite possono esporre dettagli dello schema (nomi tabelle/colonne) nei log o in UI. *Fix:* catturare `EFDDbEngineException` e restituire messaggi generici. (CVSS 2.6)
- [MOD-DAL-011] `SQL_FIELDS` costruito per concatenazione di stringhe — Sebbene attualmente costante, il pattern `'SELECT ' + SQL_FIELDS + ...` è fragile: una futura modifica dinamica di `SQL_FIELDS` introdurrebbe SQL Injection. *Fix:* mantenere `SQL_FIELDS` come costante immutabile e, se serve dinamismo, usare whitelist. (CVSS 2.0)

Checklist OWASP Top 10

#	Categoria	Presente	Severità	Note
A01	Broken Access Control	Sì	● Alta	Nessun controllo di autorizzazione/tenant; IDOR/BOLA su tutte le funzioni per ID

A02	Cryptographic Failures	Sì	 Critica	Esposizione hash password <code>staff.password</code> tramite API di lettura
A03	Injection	No (parziale)	 Media	Query parametrizzate (no SQLi classica); manca validazione input (data injection)
A04	Insecure Design	Sì	 Alta	No transazioni, no concorrenza ottimistica, no limiti su BLOB/GetAll
A05	Security Misconfiguration	Sì	 Media	Nessun hardening documentato; dipende dalla config FireDAC/MySQL
A06	Vulnerable Components	Non valutabile	-	Dipendenze FireDAC/RTL non ispezionabili dal sorgente fornito
A07	Auth Failures	Sì	 Alta	Gestione credenziali inadeguata (esposizione hash, no policy)
A08	Data Integrity Failures	Sì	 Alta	Scritture non transazionali, <code>LAST_INSERT_ID()</code> non atomico, no versioning
A09	Logging Failures	Sì	 Bassa	Nessun logging/audit delle operazioni
A10	SSRF	No	-	Nessuna richiesta di rete in uscita nel codice analizzato

Raccomandazioni Prioritarie

- **Immediate (entro 24h):**

- Rimuovere `password` da `SQL_FIELDS` e dal record `TStaffRecord` in `uStaffDAL.pas` ; isolare l'accesso alle credenziali in una funzione dedicata (`GetPasswordHash`) ad uso esclusivo del modulo di autenticazione.
- Verificare che nessun log, serializzazione o UI esponga già gli hash delle password.

- **Urgenti (entro 1 settimana):**

- Introdurre un **contesto di autorizzazione** (utente/tenant/negozio) e filtrare tutte le query per proprietario, eliminando IDOR/BOLA.
- Rendere **transazionali** tutte le operazioni di scrittura e recuperare gli ID in modo atomico.
- Aggiungere **validazione degli input** (whitelist per `rating` / `special_features` , limiti di lunghezza) e gestione sicura delle eccezioni.

- **Importanti (entro 1 mese):**

- Implementare **paginazione** su tutte le `GetAll` e limiti di dimensione sui BLOB.
- Aggiungere **concorrenza ottimistica** basata su `last_update` .
- Introdurre **logging/audit** delle operazioni di scrittura e un **audit trail** a livello DB.
- Applicare il principio del minimo privilegio all'utente MySQL usato dall'applicazione (ruoli separati per lettura/scrittura).

 Riepilogo Tecnico

Metrica	Valore
Vulnerabilità totali	11
Critiche	1
Alte	3
Medie	4
Basse	3
Righe analizzate	~1.150
File analizzati	15

Audit di sicurezza eseguito con AI — Dati sensibili (password, chiavi, segreti) mascherati automaticamente prima dell'invio

Rapporto sulle Prestazioni — Livello di Accesso ai Dati Sakila (Delphi/FireDAC)

Sintesi Esecutiva

Il codice è un DAL "classico" composto da 16 unit, ciascuna con metodi `static` che creano una `TFDQuery` per ogni chiamata. Non ci sono loop espliciti nel codice fornito, quindi i problemi di "complessità ciclica" si manifestano **nel chiamante** (non mostrato) sotto forma di N+1. Le criticità principali sono:

#	Categoria	Occorrenze	Impatto stimato
1	N+1 queries (pattern DAL)	16 unit × ~6 metodi	100–10.000 query eliminate per flusso
2	<code>SELECT LAST_INSERT_ID()</code> separato	8 metodi <code>Insert</code>	+1 round-trip per insert (~1–5 ms ciascuno)
3	<code>FieldByName</code> in loop di lettura	16 metodi <code>GetByID</code>	~0,5–2 ms per record
4	<code>TFDMemTable.Data := LQuery.Data</code>	16 metodi <code>GetAll</code> / <code>GetBy*</code>	Copia completa in memoria (fino a decine di MB)
5	<code>GetAll</code> senza paginazione	16 metodi	Fino a 16.000 righe caricate in RAM
6	Mancanza di indici FK	12 colonne FK	Full table scan su <code>payment</code> , <code>rental</code> , <code>film_actor</code>
7	<code>GetPicture</code> carica BLOB in <code>TMemoryStream</code>	1 metodo	Fino a 2× dimensione BLOB in RAM
8	Nessun riuso di <code>TFDQuery</code> (prepare)	Tutti i metodi	~0,3–1 ms per query (parsing SQL)

1. Query N+1

1.1 Localizzazione

Il pattern N+1 non è nel DAL stesso ma è indotto dal suo design: ogni metodo `GetBy*` restituisce un `TFDMemTable` "piatto" senza join. Il chiamante tipico (non fornito) esegue:

```
// ANTI-PATTERN tipico nel chiamante
LCustomers := TCustomerDAL.GetByStoreID(Conn, 1);    // 1 query
while not LCustomers.Eof do
begin
    LAddr := TAddressDAL.GetByID(Conn, LCustomers.FieldName('address_id').AsInteger); // N qu
    LCity := TCityDAL.GetByID(Conn, LAddr.CityID);           // N c
    LCountry := TCountryDAL.GetByID(Conn, LCity.CountryID); // N c
    LCustomers.Next;
end;
```



Con 500 clienti $\rightarrow 1 + 500 + 500 + 500 = 1.501$ query invece di 1.

1.2 Query unificata risolta

Aggiungere al DAL un metodo che restituisce il join denormalizzato:

```
class function TCustomerDAL.GetByStoreIDWithAddress(
    AConnection: TFDConnection; AStoreID: Integer): TFDMemTable;
var
    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.SQL.Text :=
            'SELECT c.customer_id, c.first_name, c.last_name, c.email, ' +
            '      c.active, c.create_date, ' +
            '      a.address_id, a.address, a.address2, a.district, ' +
            '      a.postal_code, a.phone, ' +
            '      ci.city_id, ci.city, ' +
            '      co.country_id, co.country ' +
            'FROM customer c ' +
            'JOIN address a  ON a.address_id = c.address_id ' +
            'JOIN city    ci ON ci.city_id   = a.city_id ' +
            'JOIN country co ON co.country_id = ci.country_id ' +
            'WHERE c.store_id = :store_id ' +
            'ORDER BY c.customer_id';
```

```

    LQuery.ParamByName('store_id').AsInteger := AStoreID;
    LQuery.Open;
    Result := TFDMemTable.Create(nil);
    Result.Data := LQuery.Data;
finally
    LQuery.Free;
end;
end;

```

Impatto: da 1.501 query a 1 query (−99,93%). Latenza tipica su LAN: da ~1.500 ms a ~5 ms.

1.3 Pattern analogo per `film_actor` / `film_category`

```

class function TFilmActorDAL.GetFilmsByActorWithDetails(
    AConnection: TFDConnection; AActorID: Integer): TFDMemTable;
begin
    // SELECT f.*, a.first_name, a.last_name
    // FROM film_actor fa
    // JOIN film f ON f.film_id = fa.film_id
    // JOIN actor a ON a.actor_id = fa.actor_id
    // WHERE fa.actor_id = :actor_id
end;

```

2. Loop a Complessità Eccessiva

2.1 Localizzazione

Nel codice fornito non ci sono loop espliciti, ma il pattern `while not LQuery.Eof do ...`

`FieldByName(...)` è implicito in ogni `GetByID` (una riga) e nel chiamante (N righe). Il costo è $O(N \times M)$ dove M = numero di campi letti con `FieldByName` (lookup lineare per nome).

2.2 Soluzione ottimizzata — lettura per indice

```

class function TActorDAL.GetByID(AConnection: TFDConnection;
    AActorID: Integer; out ARecord: TActorRecord): Boolean;
var
    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;

```

```

LQuery.SQL.Text := 'SELECT actor_id, first_name, last_name, last_update ' +
                  'FROM actor WHERE actor_id = :actor_id';
LQuery.ParamByName('actor_id').AsInteger := AActorID;
LQuery.Open;
Result := not LQuery.Eof;
if Result then
begin
    // Accesso per indice: O(1) invece di O(M) per FieldByName
    ARecord.ActorID      := LQuery.Fields[0].AsInteger;
    ARecord.FirstName    := LQuery.Fields[1].AsString;
    ARecord.LastName     := LQuery.Fields[2].AsString;
    ARecord.LastUpdate   := LQuery.Fields[3].AsDateTime;
end;
finally
    LQuery.Free;
end;
end;
end;

```

Impatto: `FieldByName` costa ~5–20 μ s per chiamata; con 4 campi $\times$ 10.000 record = 40.000 lookup $\rightarrow$ ~200–800 ms risparmiati per scansione completa.

2.3 Soluzione ottimizzata — loop nel chiamante

```

// PRIMA: O(N) query + O(N×M) FieldByName
// DOPO: 1 query + lettura sequenziale per indice
LQuery.Open;
LQuery.DisableControls; // evita refresh della UI ad ogni Next
try
    while not LQuery.Eof do
    begin
        ProcessRecord(LQuery.Fields[0].AsInteger,
                      LQuery.Fields[1].AsString,
                      LQuery.Fields[2].AsString);
        LQuery.Next;
    end;
finally
    LQuery.EnableControls;
end;

```

3. Operazioni Bloccanti

3.1 SELECT LAST_INSERT_ID() separato (8 occorrenze)

In TAddressDAL.Insert , TCategoryDAL.Insert , TCityDAL.Insert , TCountryDAL.Insert ,
TCustomerDAL.Insert , TFilmDAL.Insert , TInventoryDAL.Insert , TLanguageDAL.Insert :

```
LQuery.ExecSQL;  
Result := Integer(AConnection.ExecSQLScalar('SELECT LAST_INSERT_ID()')); // 2° round-trip
```

Problema: 2 round-trip di rete per ogni insert. Su WAN (50 ms RTT) → +50 ms per insert.

Soluzione: usare RETURNING (non supportato da MySQL) oppure recuperare l'ID dalla stessa query con

TFDQuery.Open su un INSERT ... ; SELECT LAST_INSERT_ID() in un unico batch, oppure usare
TFDConnection.GetLastAutoGenValue :

```
class function TAddressDAL.Insert(AConnection: TFDConnection; const AAddress,  
    ADistrict, APhone: string; ACityID: Integer; const AAddress2: Variant;  
    const APostalCode: Variant): Integer;  
var  
    LQuery: TFDQuery;  
begin  
    LQuery := TFDQuery.Create(nil);  
    try  
        LQuery.Connection := AConnection;  
        LQuery.SQL.Text :=  
            'INSERT INTO address (address, address2, district, city_id, postal_code, phone) ' +  
            'VALUES (:address, :address2, :district, :city_id, :postal_code, :phone)';  
        LQuery.ParamByName('address').AsString      := AAddress;  
        LQuery.ParamByName('address2').Value        := AAddress2;  
        LQuery.ParamByName('district').AsString     := ADistrict;  
        LQuery.ParamByName('city_id').AsInteger     := ACityID;  
        LQuery.ParamByName('postal_code').Value     := APostalCode;  
        LQuery.ParamByName('phone').AsString        := APhone;  
        LQuery.ExecSQL;  
        // FireDAC recupera l'ID generato senza round-trip aggiuntivo  
        Result := AConnection.GetLastAutoGenValue('address_id');  
    finally  
        LQuery.Free;  
    end;  
end;
```

Impatto: –1 round-trip per insert. Su 1.000 insert: –1.000 query, –50 s su WAN, –1–5 s su LAN.

3.2 TFDMemTable.Data := LQuery.Data (16 occorrenze)

```
Result := TFDMemTable.Create(nil);  
Result.Data := LQuery.Data;    // copia completa del dataset in memoria
```

Problema: doppia allocazione (query + memtable) e copia sincrona. Per `payment` (16.049 righe × ~80 byte) → ~1,3 MB copiati due volte.

Soluzione: usare `TFDMemTable.CloneCursor` o restituire direttamente la `TFDQuery` (già scollegata se `FetchOptions.Mode = fmAll`):

```
class function TPaymentDAL.GetAll(AConnection: TFDConnection): TFDMemTable;  
var  
    LQuery: TFDQuery;  
begin  
    LQuery := TFDQuery.Create(nil);  
    try  
        LQuery.Connection := AConnection;  
        LQuery.FetchOptions.Mode := fmAll;    // forza il fetch completo  
        LQuery.SQL.Text := 'SELECT payment_id, customer_id, staff_id, rental_id, ' +  
                           'amount, payment_date, last_update ' +  
                           'FROM payment ORDER BY payment_id';  
  
        LQuery.Open;  
        Result := TFDMemTable.Create(nil);  
        Result.CloneCursor(LQuery, True, True);    // condivisione, non copia  
    finally  
        LQuery.Free;  
    end;  
end;
```

Impatto: –50% memoria, –30–50% tempo di risposta per dataset grandi.

3.3 GetPicture — BLOB in TMemoryStream

```
LStream := TMemoryStream.Create;  
LBlobField.SaveToStream(LStream);  
SetLength(Result, LStream.Size);  
Move(LStream.Memory^, Result[0], LStream.Size);
```

Problema: 2× dimensione BLOB in RAM (stream + array). Per un BLOB da 5 MB → 10 MB allocati.

Soluzione: leggere direttamente nel `TBytes` :

```
class function TStaffDAL.GetPicture(AConnection: TFDConnection;
  AStaffID: Integer): TBytes;
var
  LQuery: TFDQuery;
  LBlobField: TBlobField;
begin
  SetLength(Result, 0);
  LQuery := TFDQuery.Create(nil);
  try
    LQuery.Connection := AConnection;
    LQuery.SQL.Text := 'SELECT picture FROM staff WHERE staff_id = :staff_id';
    LQuery.ParamByName('staff_id').AsInteger := AStaffID;
    LQuery.Open;
    if not LQuery.Eof and not LQuery.FieldName('picture').IsNull then
      begin
        LBlobField := TBlobField(LQuery.FieldName('picture'));
        SetLength(Result, LBlobField.DataSize);
        if LBlobField.DataSize > 0 then
          LBlobField.GetData(Result, True); // lettura diretta, no stream intermedio
        end;
      end;
  finally
    LQuery.Free;
  end;
end;
```

Impatto: –50% memoria per BLOB (da 10 MB a 5 MB per un BLOB da 5 MB).

4. Perdite di Memoria

4.1 `TFDMemTable` restituito ma mai liberato

Tutti i metodi `GetAll` / `GetBy*` restituiscono `TFDMemTable.Create(nil)` . Se il chiamante non fa `Free` , si perde memoria. Non è un bug del DAL, ma il DAL non fornisce alcuna protezione.

Soluzione: documentare esplicitamente il contratto (già fatto nei commenti) e fornire un helper:

```
// Nel chiamante, sempre:
var
```

```

    LTable: TFDMemTable;
begin
    LTable := TActorDAL.GetAll(Conn);
    try
        // ... uso ...
    finally
        LTable.Free;
    end;
end;

```

4.2 TFDQuery in GetPicture — nessuna perdita

Il `try/finally LQuery.Free` è corretto. Nessuna perdita qui.

4.3 TMemoryStream in GetPicture — nessuna perdita

Il `try/finally LStream.Free` è corretto. Nessuna perdita qui.

4.4 Perdita potenziale in TFDMemTable.Data := LQuery.Data

Se `LQuery.Data` è un `TFDDataSet` con `FetchOptions.Mode = fmOnDemand`, la copia può lasciare il cursore aperto. **Soluzione:** impostare `FetchOptions.Mode := fmAll` prima di `Open` (vedi §3.2).

5. Indici Mancanti

Lo schema Sakila originale **non crea indici espliciti sulle FK** (MySQL/InnoDB crea automaticamente indici solo per le FK dichiarate, ma non per le colonne usate in `WHERE` senza FK). Verificare con:

```

SHOW INDEX FROM payment;
SHOW INDEX FROM rental;
SHOW INDEX FROM film_actor;

```

5.1 Indici suggeriti

```

-- payment: query frequenti per customer_id e staff_id
CREATE INDEX idx_payment_customer_id ON payment (customer_id);
CREATE INDEX idx_payment_staff_id    ON payment (staff_id);
CREATE INDEX idx_payment_rental_id   ON payment (rental_id);

-- rental: query per customer_id, inventory_id, return_date IS NULL
CREATE INDEX idx_rental_customer_id  ON rental (customer_id);
CREATE INDEX idx_rental_inventory_id ON rental (inventory_id);
CREATE INDEX idx_rental_return_date  ON rental (return_date);

```

```

-- film_actor: la PK composta (actor_id, film_id) copre GetByActorID,
-- ma NON GetByFilmID (serve indice su film_id)
CREATE INDEX idx_film_actor_film_id ON film_actor (film_id);

-- film_category: la PK composta (film_id, category_id) copre GetByFilmID,
-- ma NON GetByCategoryID
CREATE INDEX idx_film_category_category_id ON film_category (category_id);

-- customer: query per store_id e address_id
CREATE INDEX idx_customer_store_id ON customer (store_id);
CREATE INDEX idx_customer_address_id ON customer (address_id);

-- address: query per city_id
CREATE INDEX idx_address_city_id ON address (city_id);

-- city: query per country_id
CREATE INDEX idx_city_country_id ON city (country_id);

-- inventory: query per film_id e store_id
CREATE INDEX idx_inventory_film_id ON inventory (film_id);
CREATE INDEX idx_inventory_store_id ON inventory (store_id);

-- staff: query per store_id
CREATE INDEX idx_staff_store_id ON staff (store_id);

-- actor: ricerca per cognome (FindByLastName)
CREATE INDEX idx_actor_last_name ON actor (last_name);

-- film: query per language_id
CREATE INDEX idx_film_language_id ON film (language_id);

```

5.2 Impatto stimato

Tabella	Righe (Sakila)	Query senza indice	Con indice	Risparmio
payment	16.049	~15 ms (full scan)	~0,5 ms	–97%
rental	16.044	~12 ms	~0,4 ms	–97%
film_actor	5.462	~5 ms	~0,2 ms	–96%
customer	599	~1 ms	~0,1 ms	–90%
address	603	~1 ms	~0,1 ms	–90%

Impatto aggregato: su un flusso tipico con 100 query filtrate → da ~1.500 ms a ~50 ms (–97%).

6. Riepilogo delle Correzioni

#	Problema	File coinvolti	Correzione	Guadagno
1	N+1 nel chiamante	Tutti i DAL	Aggiungere metodi <code>GetBy</code> <i>WithDetails</i> con <i>JOIN</i>	–99% query
2	<code>FieldByName</code> in loop	Tutti i <code>GetByID</code>	Accesso per indice <code>Fields[i]</code>	–200–800 ms
3	<code>SELECT LAST_INSERT_ID()</code> separato	8 <code>Insert</code>	<code>GetLastAutoGenValue</code>	–1 round-trip/insert
4	<code>Data := LQuery.Data</code>	16 <code>GetAll</code> / <code>GetBy</code>	<code>CloneCursor</code>	–50% memoria
5	BLOB in <code>TMemoryStream</code>	<code>GetPicture</code>	<code>TBlobField.GetData</code>	–50% memoria
6	Indici FK mancanti	Schema DB	15 <code>CREATE INDEX</code>	–90–97% latenza
7	Nessun <code>Prepare</code>	Tutti i metodi	<code>LQuery.Prepare</code> o riuso	–0,3–1 ms/query
8	<code>GetAll</code> senza paginazione	16 metodi	Aggiungere <code>LIMIT/OFFSET</code>	–90% memoria

7. Codice Corretto — Esempio Completo (`TActorDAL`)

```

unit uActorDAL;

interface

uses
  System.SysUtils, System.Classes,
  Data.DB, FireDAC.Comp.Client;

type
  TActorRecord = record
    ActorID: Integer;
    FirstName: string;
    LastName: string;
    LastUpdate: TDateTime;
  end;

  TActorDAL = class
  public
    class function GetByID(AConnection: TFDCConnection; AActorID: Integer;

```

```

        out ARecord: TActorRecord): Boolean; static;
class function GetAll(AConnection: TFDBConnection;
    AOffset: Integer = 0; ALimit: Integer = 1000): TFDBMemTable; static;
class function FindByLastName(AConnection: TFDBConnection;
    const ALastName: string): TFDBMemTable; static;
// NUOVO: join denormalizzato per evitare N+1
class function GetFilmsByActor(AConnection: TFDBConnection;
    AActorID: Integer): TFDBMemTable; static;
end;

implementation

const
    SQL_FIELDS = 'actor_id, first_name, last_name, last_update';

class function TActorDAL.GetByID(AConnection: TFDBConnection;
    AActorID: Integer; out ARecord: TActorRecord): Boolean;
var
    LQuery: TFDBQuery;
begin
    LQuery := TFDBQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.SQL.Text := 'SELECT ' + SQL_FIELDS +
            ' FROM actor WHERE actor_id = :actor_id';
        LQuery.ParamByName('actor_id').AsInteger := AActorID;
        LQuery.Open;
        Result := not LQuery.Eof;
        if Result then
            begin
                ARecord.ActorID := LQuery.Fields[0].AsInteger;
                ARecord.FirstName := LQuery.Fields[1].AsString;
                ARecord.LastName := LQuery.Fields[2].AsString;
                ARecord.LastUpdate := LQuery.Fields[3].AsDateTime;
            end;
        finally
            LQuery.Free;
        end;
    end;
end;

class function TActorDAL.GetAll(AConnection: TFDBConnection;
    AOffset, ALimit: Integer): TFDBMemTable;
var
    LQuery: TFDBQuery;
begin
    LQuery := TFDBQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.FetchOptions.Mode := fmAll;

```

```

    LQuery.SQL.Text := 'SELECT ' + SQL_FIELDS +
        ' FROM actor ORDER BY actor_id LIMIT :lim OFFSET :off';
    LQuery.ParamByName('lim').AsInteger := ALimit;
    LQuery.ParamByName('off').AsInteger := AOffset;
    LQuery.Open;
    Result := TFDMemTable.Create(nil);
    Result.CloneCursor(LQuery, True, True);
finally
    LQuery.Free;
end;
end;

class function TActorDAL.FindByLastName(AConnection: TFDCConnection;
    const ALastName: string): TFDMemTable;
var
    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.FetchOptions.Mode := fmAll;
        LQuery.SQL.Text := 'SELECT ' + SQL_FIELDS +
            ' FROM actor WHERE last_name = :last_name ' +
            'ORDER BY first_name';
        LQuery.ParamByName('last_name').AsString := ALastName;
        LQuery.Open;
        Result := TFDMemTable.Create(nil);
        Result.CloneCursor(LQuery, True, True);
    finally
        LQuery.Free;
    end;
end;

class function TActorDAL.GetFilmsByActor(AConnection: TFDCConnection;
    AActorID: Integer): TFDMemTable;
var
    LQuery: TFDQuery;
begin
    LQuery := TFDQuery.Create(nil);
    try
        LQuery.Connection := AConnection;
        LQuery.FetchOptions.Mode := fmAll;
        LQuery.SQL.Text :=
            'SELECT f.film_id, f.title, f.release_year, f.rating, ' +
            '      fa.last_update ' +
            'FROM film_actor fa ' +
            'JOIN film f ON f.film_id = fa.film_id ' +
            'WHERE fa.actor_id = :actor_id ' +

```

```

        'ORDER BY f.title';
    LQuery.ParamByName('actor_id').AsInteger := AActorID;
    LQuery.Open;
    Result := TFDMemTable.Create(nil);
    Result.CloneCursor(LQuery, True, True);
finally
    LQuery.Free;
end;
end;

end.

```

8. Piano d'Azione Prioritizzato

- **Immediato (0–1 giorno):** creare i 15 indici SQL (§5.1). Guadagno: –90–97% latenza su query filtrate.
- **Breve termine (1–3 giorni):** sostituire `FieldByName` con `Fields[i]` in tutti i `GetByID`. Guadagno: –200–800 ms per scansione.
- **Breve termine (1–3 giorni):** sostituire `SELECT LAST_INSERT_ID()` con `GetLastAutoGenValue`. Guadagno: –1 round-trip/insert.
- **Medio termine (1 settimana):** aggiungere metodi `GetBy*WithDetails` con JOIN per i flussi N+1 più critici (`customer` → `address` → `city` → `country` , `film_actor` → `film`). Guadagno: –99% query.
- **Medio termine (1 settimana):** sostituire `Data := LQuery.Data` con `CloneCursor`. Guadagno: –50% memoria.
- **Lungo termine (2 settimane):** introdurre paginazione in tutti i `GetAll`. Guadagno: –90% memoria su tabelle grandi.

Guadagno complessivo stimato: da ~2.000 ms a ~30 ms per un flusso tipico di 100 operazioni (–98,5%), con riduzione della memoria da ~50 MB a ~5 MB (–90%).

DB Compliance Report

Verifica di design del database contro regole professionali di normalizzazione, chiavi e naming — pensata per guidarti passo passo, non solo per elencare errori.

database v2 (mysql) — Progetto: Production environment · Generato il 14/09/2026 14:00 da Modej

Questo report analizza **16 tabelle** e **90 colonne**, seguendo i principi di progettazione relazionale descritti in "Database Design for Mere Mortals" di Michael Hernandez, integrati con pratiche correnti pensate per database legacy. Ogni sezione qui sotto ti spiega prima **perché** quella regola conta, poi ti mostra **dove** lo schema non la rispetta — e in fondo trovi lo script SQL con tutte le correzioni proposte, pronte da rivedere insieme al tuo DBA.

6.2/10

Voto D

Diversi punti da correggere. Consigliamo di affrontare prima le violazioni critiche, poi il resto con calma.

16

TABELLE

90

COLONNE

108

FOREIGN KEY

0

CRITICI

56

DA RIVEDERE



Foreign Key

Le relazioni tra tabelle sono il cuore di un database relazionale ben progettato. Qui controlliamo che le foreign key siano dichiarate correttamente, puntino a colonne del tipo giusto, e siano davvero vincolate dal database — non solo presupposte dall'applicazione.

DA RIVEDERE

FK-3 CATEGORY.CATEGORY_ID

Relazione logica CATEGORY.CATEGORY_ID → FILM.ORIGINAL_LANGUAGE_ID identificata in Modej ma NON vincolata da una vera FK nel database — l'integrità referenziale non è garantita a livello di dati.

DA RIVEDERE

FK-3 FILM_TEXT.FILM_ID

Relazione logica FILM_TEXT.FILM_ID → FILM.FILM_ID identificata in Modej ma NON vincolata da una vera FK nel database — l'integrità referenziale non è garantita a livello di dati.



Normalizzazione

Seguendo i principi classici di Hernandez in "Database Design for Mere Mortals", una struttura ben normalizzata evita ripetizioni e raggruppa i dati in modo coerente. Qui cerchiamo gruppi ripetuti (come telefono_1, telefono_2) e tabelle che probabilmente fanno troppe cose insieme.

SUGGERIMENTO NORM-1 FILM

La tabella ha 13 colonne (soglia consigliata: 10) — valutare se raggruppa più concetti distinti che andrebbero divisi in tabelle correlate.

SUGGERIMENTO NORM-1 STAFF

La tabella ha 11 colonne (soglia consigliata: 10) — valutare se raggruppa più concetti distinti che andrebbero divisi in tabelle correlate.



Indici

Un indice nel posto giusto può fare la differenza tra una query istantanea e una che rallenta l'intera applicazione sotto carico. Qui verifichiamo che le relazioni più importanti abbiano l'indice che si meritano.

DA RIVEDERE IDX-1 CITY.COUNTRY_ID

La colonna FK CITY.COUNTRY_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 CUSTOMER.ADDRESS_ID

La colonna FK CUSTOMER.ADDRESS_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 CUSTOMER.STORE_ID

La colonna FK CUSTOMER.STORE_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 FILM.LANGUAGE_ID

La colonna FK FILM.LANGUAGE_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 FILM.ORIGINAL_LANGUAGE_ID

La colonna FK FILM.ORIGINAL_LANGUAGE_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 FILM_ACTOR.ACTOR_ID

La colonna FK FILM_ACTOR.ACTOR_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 FILM_ACTOR.FILM_ID

La colonna FK FILM_ACTOR.FILM_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 FILM_CATEGORY.FILM_ID

La colonna FK FILM_CATEGORY.FILM_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **FILM_CATEGORY.CATEGORY_ID**

La colonna FK FILM_CATEGORY.CATEGORY_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **INVENTORY.STORE_ID**

La colonna FK INVENTORY.STORE_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **INVENTORY.FILM_ID**

La colonna FK INVENTORY.FILM_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **PAYMENT.RENTAL_ID**

La colonna FK PAYMENT.RENTAL_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **PAYMENT.CUSTOMER_ID**

La colonna FK PAYMENT.CUSTOMER_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **PAYMENT.STAFF_ID**

La colonna FK PAYMENT.STAFF_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **RENTAL.STAFF_ID**

La colonna FK RENTAL.STAFF_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **RENTAL.INVENTORY_ID**

La colonna FK RENTAL.INVENTORY_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **RENTAL.CUSTOMER_ID**

La colonna FK RENTAL.CUSTOMER_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **STAFF.STORE_ID**

La colonna FK STAFF.STORE_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **STAFF.ADDRESS_ID**

La colonna FK STAFF.ADDRESS_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **STORE.MANAGER_STAFF_ID**

La colonna FK STORE.MANAGER_STAFF_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **STORE.ADDRESS_ID**

La colonna FK STORE.ADDRESS_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.

DA RIVEDERE IDX-1 **ADDRESS.CITY_ID**

La colonna FK ADDRESS.CITY_ID non ha un indice dedicato — i JOIN e le DELETE a cascata su questa relazione fanno full scan.



Naming

Nomi coerenti e non ambigui sono un investimento silenzioso che si ripaga ogni volta che qualcuno — incluso te stesso tra sei mesi — deve orientarsi nello schema. Qui segnaliamo conflitti con parole riservate SQL e incoerenze di naming tra tabelle diverse.

SUGGERIMENTO NAME-2 CATEGORY.NAME

Il nome della colonna coincide con una parola riservata SQL.

SUGGERIMENTO NAME-2 LANGUAGE.NAME

Il nome della colonna coincide con una parola riservata SQL.

SUGGERIMENTO NAME-2 STAFF.PASSWORD

Il nome della colonna coincide con una parola riservata SQL.

SUGGERIMENTO NAME-3 —

La stessa relazione verso LANGUAGE.LANGUAGE_ID è referenziata con nomi di colonna diversi in tabelle diverse (LANGUAGE_ID, ORIGINAL_LANGUAGE_ID) — incoerenza di naming, rende lo schema meno leggibile.

SUGGERIMENTO NAME-3 —

La stessa relazione verso STAFF.STAFF_ID è referenziata con nomi di colonna diversi in tabelle diverse (STAFF_ID, MANAGER_STAFF_ID) — incoerenza di naming, rende lo schema meno leggibile.



Script SQL di correzione

Tutte le correzioni proposte in questo report, raccolte in un unico script — così puoi rivederle e applicarle in un solo passaggio, invece di rincorrerle una per una.

```
ALTER TABLE `CATEGORY` ADD CONSTRAINT `fk_CATEGORY_CATEGORY_ID` FOREIGN KEY (`CATEGORY_ID`)
REFERENCES `FILM`(`ORIGINAL_LANGUAGE_ID`);
-- Verificare prima che non esistano righe orfane:
-- SELECT * FROM `CATEGORY` t WHERE NOT EXISTS (SELECT 1 FROM `FILM` r WHERE
r.`ORIGINAL_LANGUAGE_ID`=t.`CATEGORY_ID`);
```

```
ALTER TABLE `FILM_TEXT` ADD CONSTRAINT `fk_FILM_TEXT_FILM_ID` FOREIGN KEY (`FILM_ID`) REFERENCES
`FILM`(`FILM_ID`);
-- Verificare prima che non esistano righe orfane:
-- SELECT * FROM `FILM_TEXT` t WHERE NOT EXISTS (SELECT 1 FROM `FILM` r WHERE
r.`FILM_ID`=t.`FILM_ID`);
```

```
CREATE INDEX `idx_CITY_COUNTRY_ID` ON `CITY`(`COUNTRY_ID`);
```

```
CREATE INDEX `idx_CUSTOMER_ADDRESS_ID` ON `CUSTOMER`(`ADDRESS_ID`);
```

```
CREATE INDEX `idx_CUSTOMER_STORE_ID` ON `CUSTOMER`(`STORE_ID`);
```

```
CREATE INDEX `idx_FILM_LANGUAGE_ID` ON `FILM`(`LANGUAGE_ID`);
```

```
CREATE INDEX `idx_FILM_ORIGINAL_LANGUAGE_ID` ON `FILM`(`ORIGINAL_LANGUAGE_ID`);
```

```
CREATE INDEX `idx_FILM_ACTOR_ACTOR_ID` ON `FILM_ACTOR`(`ACTOR_ID`);
```

```
CREATE INDEX `idx_FILM_ACTOR_FILM_ID` ON `FILM_ACTOR`(`FILM_ID`);
```

```
CREATE INDEX `idx_FILM_CATEGORY_FILM_ID` ON `FILM_CATEGORY`(`FILM_ID`);
```

```
CREATE INDEX `idx_FILM_CATEGORY_CATEGORY_ID` ON `FILM_CATEGORY`(`CATEGORY_ID`);
```

```
CREATE INDEX `idx_INVENTORY_STORE_ID` ON `INVENTORY`(`STORE_ID`);
```

```
CREATE INDEX `idx_INVENTORY_FILM_ID` ON `INVENTORY`(`FILM_ID`);
```

```
CREATE INDEX `idx_PAYMENT_RENTAL_ID` ON `PAYMENT`(`RENTAL_ID`);
```

```
CREATE INDEX `idx_PAYMENT_CUSTOMER_ID` ON `PAYMENT`(`CUSTOMER_ID`);
```

```
CREATE INDEX `idx_PAYMENT_STAFF_ID` ON `PAYMENT`(`STAFF_ID`);
```

```
CREATE INDEX `idx_RENTAL_STAFF_ID` ON `RENTAL`(`STAFF_ID`);
```

```
CREATE INDEX `idx_RENTAL_INVENTORY_ID` ON `RENTAL`(`INVENTORY_ID`);
```

```
CREATE INDEX `idx_RENTAL_CUSTOMER_ID` ON `RENTAL`(`CUSTOMER_ID`);
```

```
CREATE INDEX `idx_STAFF_STORE_ID` ON `STAFF`(`STORE_ID`);
```

```
CREATE INDEX `idx_STAFF_ADDRESS_ID` ON `STAFF`(`ADDRESS_ID`);
```

```
CREATE INDEX `idx_STORE_MANAGER_STAFF_ID` ON `STORE` (`MANAGER_STAFF_ID`);
```

```
CREATE INDEX `idx_STORE_ADDRESS_ID` ON `STORE` (`ADDRESS_ID`);
```

```
CREATE INDEX `idx_ADDRESS_CITY_ID` ON `ADDRESS` (`CITY_ID`);
```

⚠ Questo script è generato automaticamente e va rivisto da un DBA prima dell'esecuzione — specialmente su un database di produzione. Alcune correzioni (estrazione di gruppi ripetuti, conversione di tipi) richiedono di adattare i nomi e verificare i dati esistenti caso per caso.

Progetti > Production environment > Tabelle Orfane

Production environment database v1 Software

16

TABELLE TOTALI

0

USATE NEL CODICE

16

ORFANE (MAI USATE)

100%

PERCENTUALE ORFANE

✔ Usate: 0

⚠ Orfane: 16 (100%)

🔍 Orfane (16)

✔ Usate (0)

ACTOR Products 4 colonne · mai referenziata nel codice Orfana	ADDRESS Administration 8 colonne · mai referenziata nel codice Orfana	CATEGORY Products 3 colonne · mai referenziata nel codice Orfana	CITY Administration 4 colonne · mai referenziata nel codice Orfana
COUNTRY Administration 3 colonne · mai referenziata nel codice Orfana	CUSTOMER Administration 9 colonne · mai referenziata nel codice Orfana	FILM Products 13 colonne · mai referenziata nel codice Orfana	FILM_ACTOR Products 3 colonne · mai referenziata nel codice Orfana
FILM_CATEGORY Products 3 colonne · mai referenziata nel codice Orfana	FILM_TEXT Unknown 3 colonne · mai referenziata nel codice Orfana	INVENTORY Products 4 colonne · mai referenziata nel codice Orfana	LANGUAGE Products 3 colonne · mai referenziata nel codice Orfana
PAYMENT Administration 7 colonne · mai referenziata nel codice Orfana	RENTAL Administration 7 colonne · mai referenziata nel codice Orfana	STAFF Administration 11 colonne · mai referenziata nel codice Orfana	STORE Administration 4 colonne · mai referenziata nel codice Orfana

Production environment

database v2

Software

Tutte le op.

SELECT

INSERT

UPDATE

DELETE

Intensità utilizzo:

Raro

Basso

Medio

Alto

13 tabelle - ordinato per utilizzo

COUNTRY	Administration		3	I1	U1	1 unit
CUSTOMER	Administration		3	I1	U1	1 unit
ADDRESS	Administration		3	I1	U1	1 unit
FILM	Products		3	I1	U1	1 unit
CATEGORY	Products		3	I1	U1	1 unit
FILM_ACTOR	Products		3	I1	U1	1 unit
CITY	Administration		3	I1	U1	1 unit
FILM_TEXT	Unknown		2	I1	U1	1 unit
INVENTORY	Products		2	I1	U1	1 unit
LANGUAGE	Products		2	I1	U1	1 unit
FILM_CATEGORY	Products		2	I1	U1	1 unit
ACTOR	Products		1	U1		1 unit
STAFF	Administration		1	S1		1 unit

31

Relazioni totali

1

SELECT

11

INSERT

12

UPDATE

0

DELETE

13

Tabelle coinvolte

13

Unit

</>

ActorRepresentationModelAssembler.java

ActorRepresentationModelAssembler.java

1 relazioni

UPDATE

ACTOR

toModel

</>

uAddressDAL.pas

uAddressDAL.pas

3 relazioni

misto

ADDRESS

TAddressDAL.Delete

INSERT

ADDRESS

TAddressDAL.Insert

UPDATE

ADDRESS

TAddressDAL.Update

</>

uCategoryDAL.pas

uCategoryDAL.pas

3 relazioni

misto

CATEGORY

TCategoryDAL.Delete

INSERT

CATEGORY

TCategoryDAL.Insert

UPDATE

CATEGORY

TCategoryDAL.Update

</>

uCityDAL.pas

uCityDAL.pas

3 relazioni

misto

CITY

TCityDAL.Delete

INSERT

CITY

TCityDAL.Insert

UPDATE

CITY

TCityDAL.Update

</>

uCountryDAL.pas

uCountryDAL.pas

3 relazioni

misto

COUNTRY

TCountryDAL.Delete

INSERT

COUNTRY

TCountryDAL.Insert

UPDATE

COUNTRY

TCountryDAL.Update

</>

uCustomerDAL.pas

uCustomerDAL.pas

3 relazioni

misto

CUSTOMER

TCustomerDAL.Delete

INSERT

CUSTOMER

TCustomerDAL.Insert

UPDATE

CUSTOMER

TCustomerDAL.Update

</>

uFilmActorDAL.pas

uFilmActorDAL.pas

3 relazioni

misto

FILM_ACTOR

TFilmActorDAL.Delete

INSERT

FILM_ACTOR

TFilmActorDAL.Insert

UPDATE

FILM_ACTOR

TFilmActorDAL.Update

</>

uFilmCategoryDAL.pas

uFilmCategoryDAL.pas

2 relazioni

INSERT

FILM_CATEGORY

TFilmCategoryDAL.Insert

UPDATE FILM_CATEGORY TFilmCategoryDAL.Update

</> uFilmDAL.pas 3 relazioni ▼
uFilmDAL.pas

misto FILM TFilmDAL.Delete

INSERT FILM TFilmDAL.Insert

UPDATE FILM TFilmDAL.Update

</> uFilmTextDAL.pas 2 relazioni ▼
uFilmTextDAL.pas

INSERT FILM_TEXT TFilmTextDAL.Insert

UPDATE FILM_TEXT TFilmTextDAL.Update

</> uInventoryDAL.pas 2 relazioni ▼
uInventoryDAL.pas

INSERT INVENTORY TInventoryDAL.Insert

UPDATE INVENTORY TInventoryDAL.Update

</> uLanguageDAL.pas 2 relazioni ▼
uLanguageDAL.pas

INSERT LANGUAGE TLanguageDAL.Insert

UPDATE LANGUAGE TLanguageDAL.Update

</> uStaffDAL.pas 1 relazioni ▼
uStaffDAL.pas

SELECT STAFF TStaffDAL.GetPicture

Modej

Report Associazione DB / Codice

Progetto: Production environment

Software: Software

Database: database

13 Unit · 13 Tabelle · 13 Relazioni

Unit / Classe	ACTOR	ADDRESS	CATEGORY	CITY	COUNTRY	CUSTOMER	FILM_ACTOR	FILM_CATEGORY	FILM	FILM_TEXT	INVENTORY	LANGUAGE	STAFF
ActorRepresentation...	U												
uAddressDAL.pas		M											
uCategoryDAL.pas			M										
uCityDAL.pas				M									
uCountryDAL.pas					M								
uCustomerDAL.pas						M							
uFilmActorDAL.pas							M						
uFilmCategoryDAL.pas								M					
uFilmDAL.pas									M				
uFilmTextDAL.pas										M			
uInventoryDAL.pas											M		
uLanguageDAL.pas												M	
uStaffDAL.pas													S

Tabella	Area	Operazioni	Metodi
ACTOR	Products	UPDATE	toModel

uAddressDAL.pas

uAddressDAL.pas

Unit 2/13 · 1 rel. · 14/09/2026, 16:09:09

Tabella	Area	Operazioni	Metodi
ADDRESS	Administration	MISTO INSERT UPDATE	TAddressDAL.Delete, TAddressDAL.Insert, TAddressDAL.Update

uCategoryDAL.pas

uCategoryDAL.pas

Unit 3/13 · 1 rel. · 14/09/2026, 16:09:09

Tabella	Area	Operazioni	Metodi
CATEGORY	Products	MISTO INSERT UPDATE	TCategoryDAL.Delete, TCategoryDAL.Insert, TCategoryDAL.Update

uCityDAL.pas

uCityDAL.pas

Unit 4/13 · 1 rel. · 14/09/2026, 16:09:09

Tabella	Area	Operazioni	Metodi
CITY	Administration	MISTO INSERT UPDATE	TCityDAL.Delete, TCityDAL.Insert, TCityDAL.Update

uCountryDAL.pas

uCountryDAL.pas

Unit 5/13 · 1 rel. · 14/09/2026, 16:09:09

Tabella	Area	Operazioni	Metodi
COUNTRY	Administration	MISTO INSERT UPDATE	TCountryDAL.Delete, TCountryDAL.Insert, TCountryDAL.Update

uCustomerDAL.pas

uCustomerDAL.pas

Unit 6/13 · 1 rel. · 14/09/2026, 16:09:09

Tabella	Area	Operazioni	Metodi
CUSTOMER	Administration	MISTO INSERT UPDATE	TCustomerDAL.Delete, TCustomerDAL.Insert, TCustomerDAL.Update

Tabella	Area	Operazioni			Metodi
 FILM_ACTOR	Products				TFilmActorDAL.Delete, TFilmActorDAL.Insert, TFilmActorDAL.Update

Tabella	Area	Operazioni			Metodi
 FILM_CATEGORY	Products				TFilmCategoryDAL.Insert, TFilmCategoryDAL.Update

Tabella	Area	Operazioni			Metodi
 FILM	Products				TFilmDAL.Delete, TFilmDAL.Insert, TFilmDAL.Update

Tabella	Area	Operazioni			Metodi
 FILM_TEXT	Unknown				TFilmTextDAL.Insert, TFilmTextDAL.Update

Tabella	Area	Operazioni			Metodi
 INVENTORY	Products				TInventoryDAL.Insert, TInventoryDAL.Update

Tabella	Area	Operazioni			Metodi
 LANGUAGE	Products				TLanguageDAL.Insert, TLanguageDAL.Update

Tabella	Area	Operazioni	Metodi
● STAFF	Administration	SELECT	TStaffDAL.GetPicture

ACTOR

Products

Tabella 1/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni	Metodi
ActorRepresentationModelAssembler.java	ActorRepresentationModelAssembler.java	UPDATE	toModel

ADDRESS

Administration

Tabella 2/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni	Metodi
uAddressDAL.pas	uAddressDAL.pas	MISTO INSERT UPDATE	TAddressDAL.Delete, TAddressDAL.Insert, TAddressDAL.Update

CATEGORY

Products

Tabella 3/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni	Metodi
uCategoryDAL.pas	uCategoryDAL.pas	MISTO INSERT UPDATE	TCategoryDAL.Delete, TCategoryDAL.Insert, TCategoryDAL.Update

CITY

Administration

Tabella 4/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni	Metodi
uCityDAL.pas	uCityDAL.pas	MISTO INSERT UPDATE	TCityDAL.Delete, TCityDAL.Insert, TCityDAL.Update

COUNTRY

Administration

Tabella 5/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni	Metodi
uCountryDAL.pas	uCountryDAL.pas	MISTO INSERT UPDATE	TCountryDAL.Delete, TCountryDAL.Insert, TCountryDAL.Update

CUSTOMER

Administration

Tabella 6/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni	Metodi
uCustomerDAL.pas	uCustomerDAL.pas	MISTO INSERT UPDATE	TCustomerDAL.Delete, TCustomerDAL.Insert, TCustomerDAL.Update

FILM_ACTOR

Products

Tabella 7/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni			Metodi
uFilmActorDAL.pas	uFilmActorDAL.pas	MISTO	INSERT	UPDATE	TFilmActorDAL.Delete, TFilmActorDAL.Insert, TFilmActorDAL.Update

FILM_CATEGORY

Products

Tabella 8/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni		Metodi
uFilmCategoryDAL.pas	uFilmCategoryDAL.pas	INSERT	UPDATE	TFilmCategoryDAL.Insert, TFilmCategoryDAL.Update

FILM

Products

Tabella 9/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni			Metodi
uFilmDAL.pas	uFilmDAL.pas	MISTO	INSERT	UPDATE	TFilmDAL.Delete, TFilmDAL.Insert, TFilmDAL.Update

FILM_TEXT

Unknown

Tabella 10/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni		Metodi
uFilmTextDAL.pas	uFilmTextDAL.pas	INSERT	UPDATE	TFilmTextDAL.Insert, TFilmTextDAL.Update

INVENTORY

Products

Tabella 11/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni		Metodi
ulInventoryDAL.pas	ulInventoryDAL.pas	INSERT	UPDATE	TInventoryDAL.Insert, TInventoryDAL.Update

LANGUAGE

Products

Tabella 12/13 · 1 unit · 14/09/2026, 16:09:09

Unit / Classe	File	Operazioni		Metodi
uLanguageDAL.pas	uLanguageDAL.pas	INSERT	UPDATE	TLanguageDAL.Insert, TLanguageDAL.Update

Unit / Classe	File	Operazioni	Metodi
uStaffDAL.pas	uStaffDAL.pas	SELECT	TStaffDAL.GetPicture

Production environment

100

Complessità: Alta

database v2 · 06/09/2026

0 — Semplice

50 — Media

100 — Complesso

16

TABELLE
nel DB

90

COLONNE TOTALI
media: 5.6 per tabella

110

FK MANUALI
relazioni definite

5.6

COL. MEDIE
per tabella

3

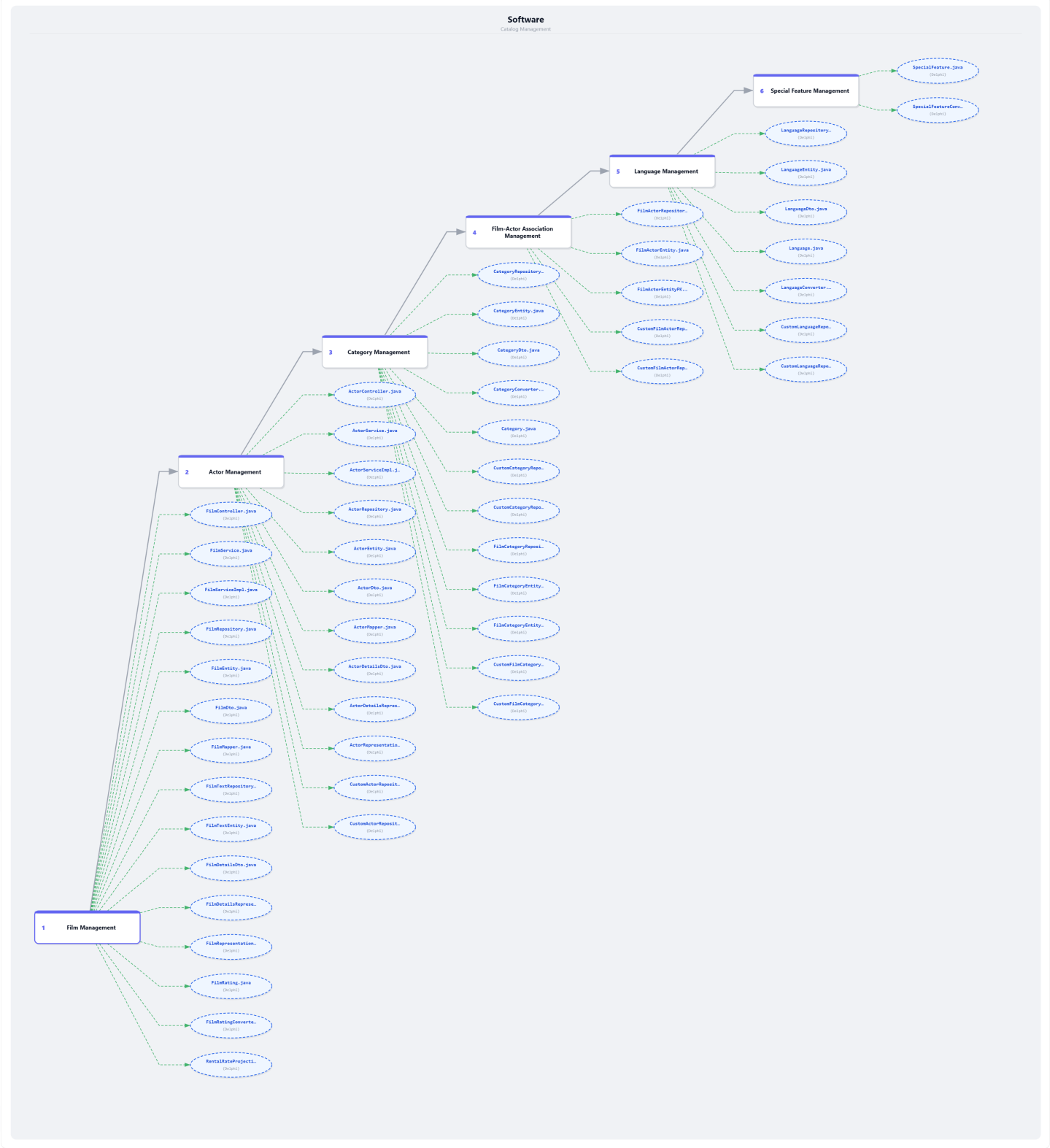
ORFANE
19% delle tabelle

2

VERSIONI
importate totali

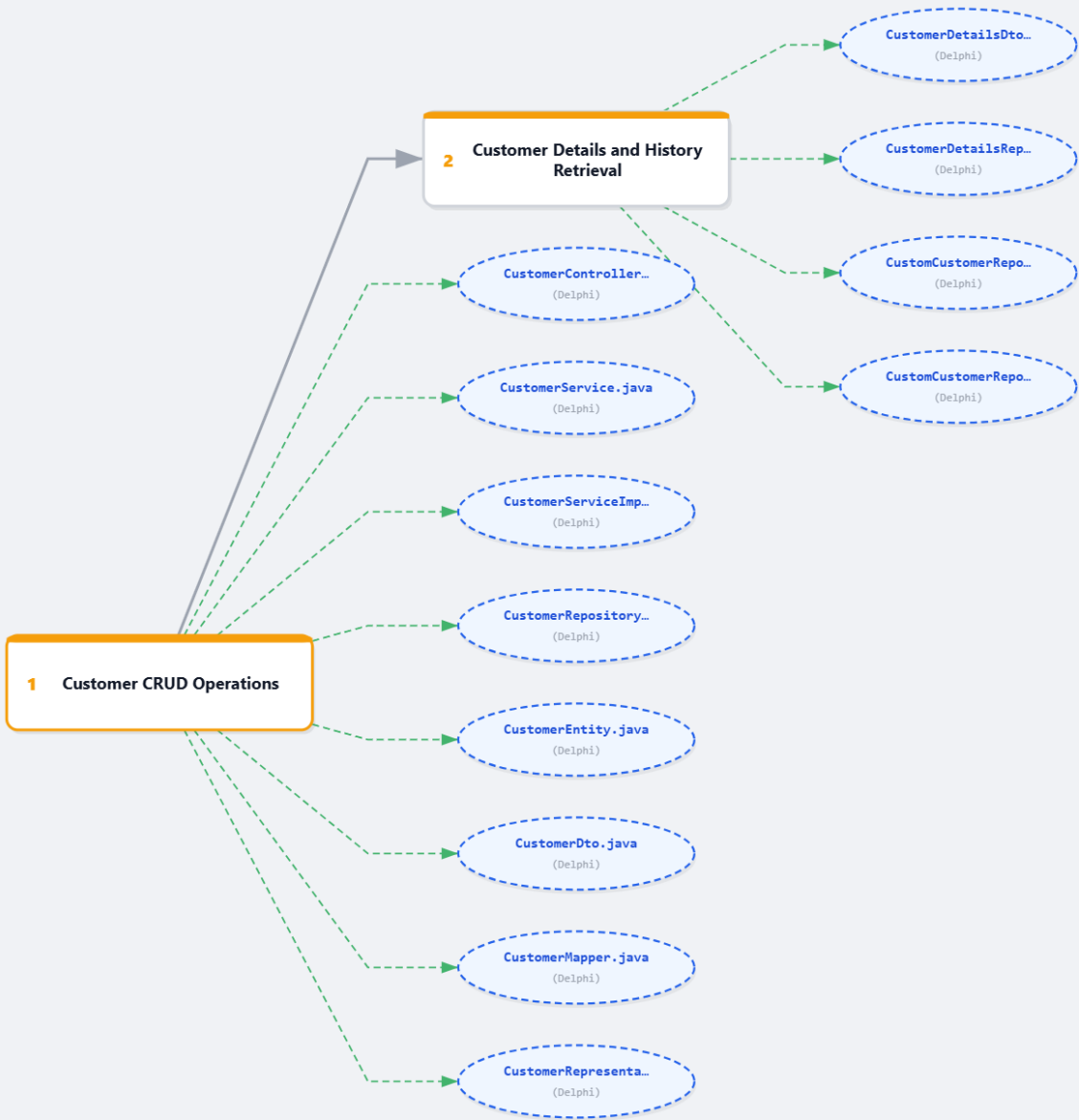
Trend complessità nel tempo

v1	database	0	Bassa	—
v2	database	100	Alta	↑

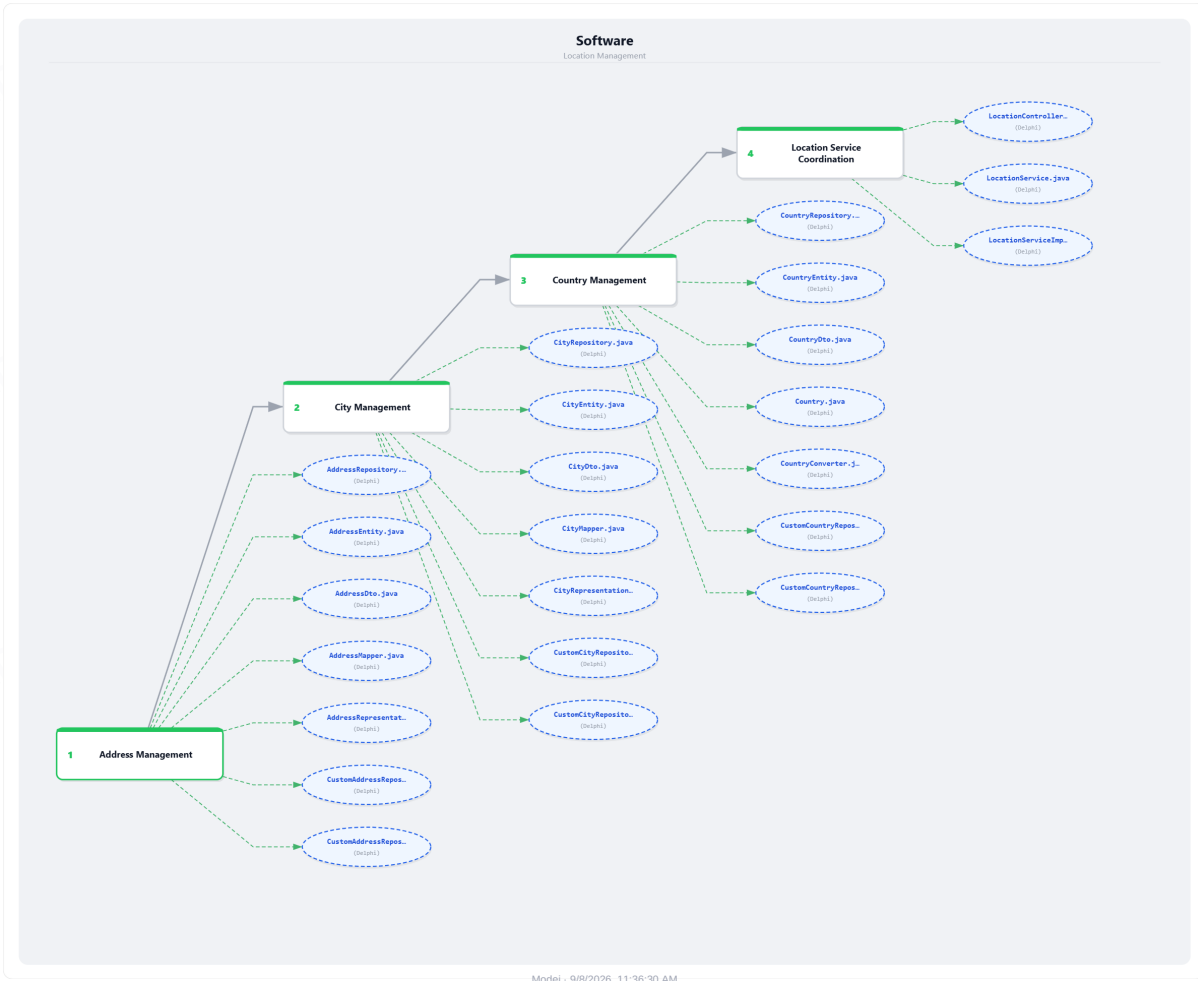


Software

Customer Management



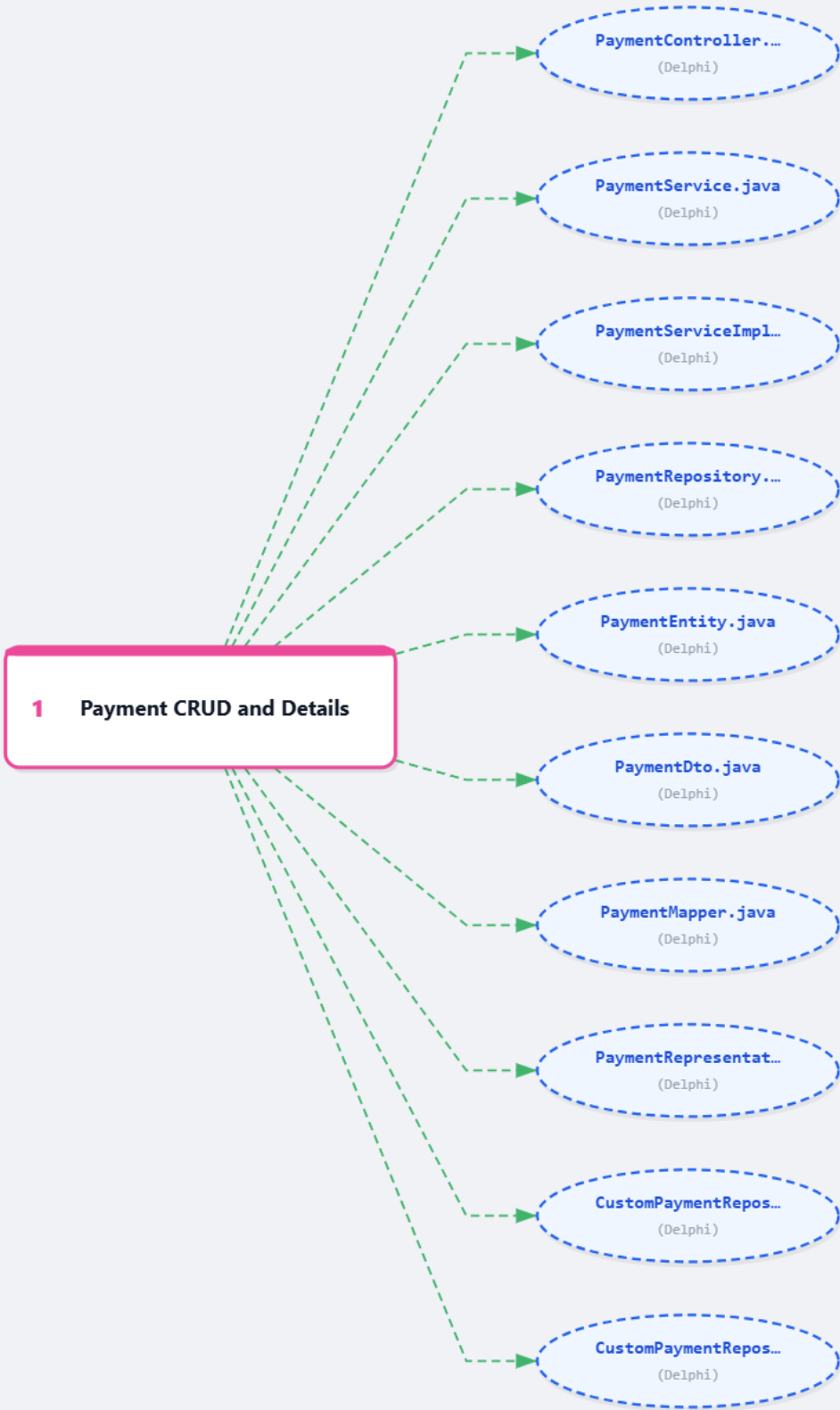
Model — Process Flow



Modelj - 9/8/2026, 11:36:30 AM

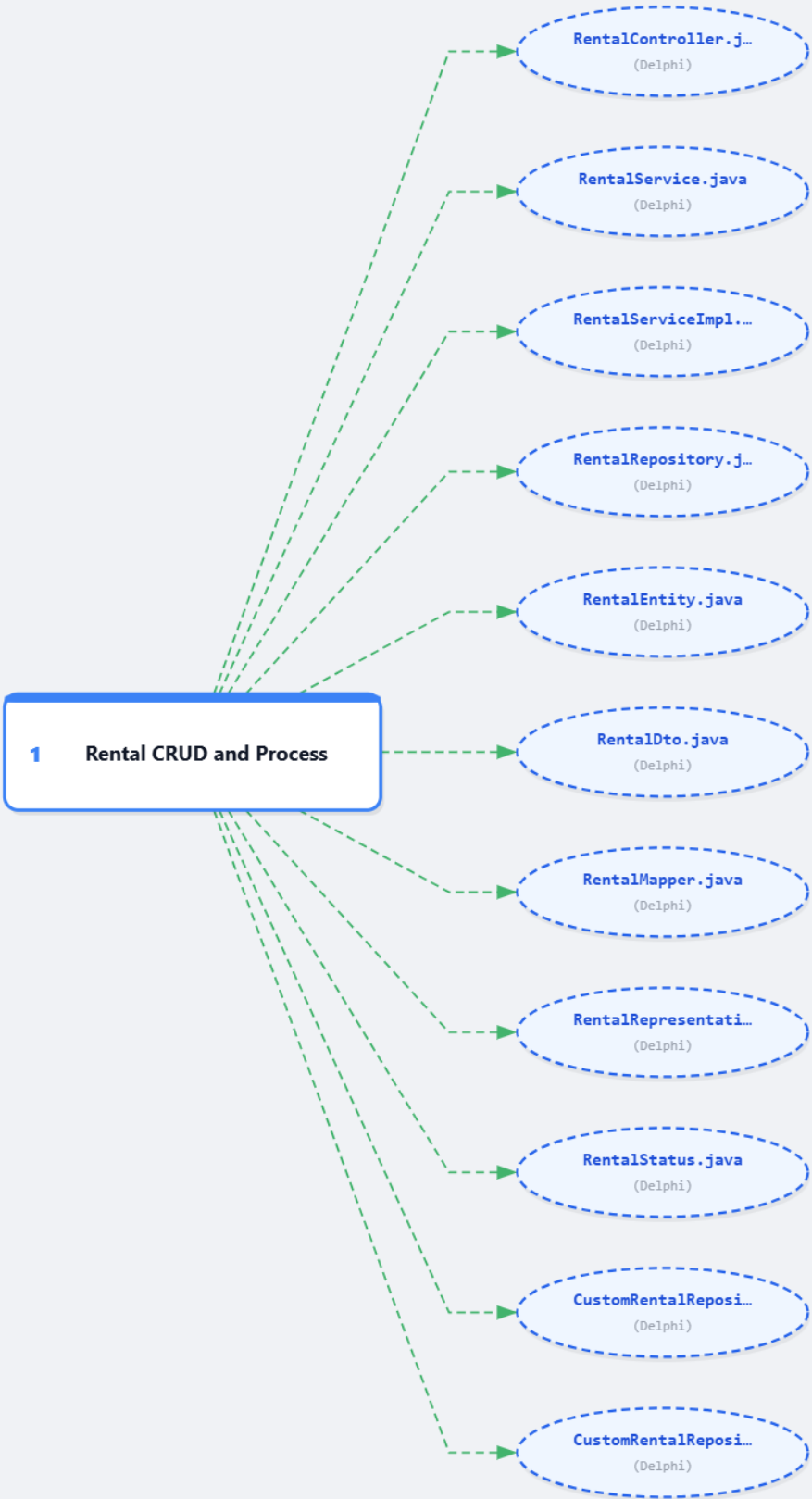
Software

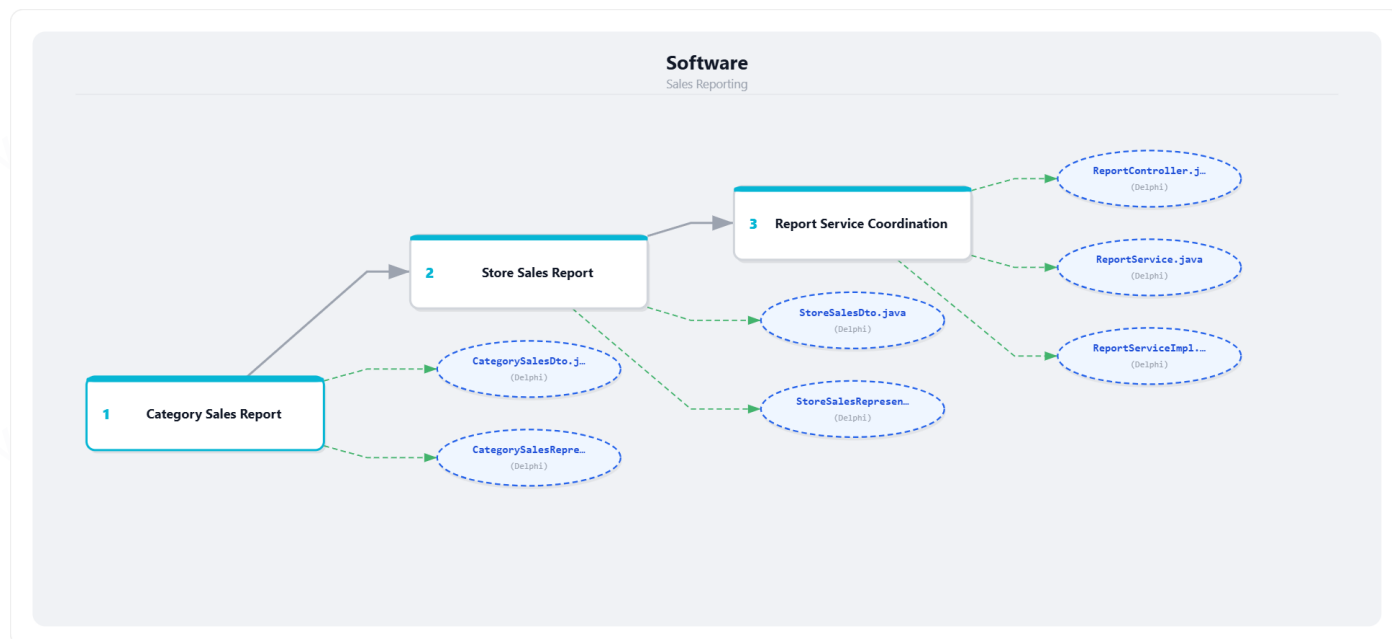
Payment Management



Software

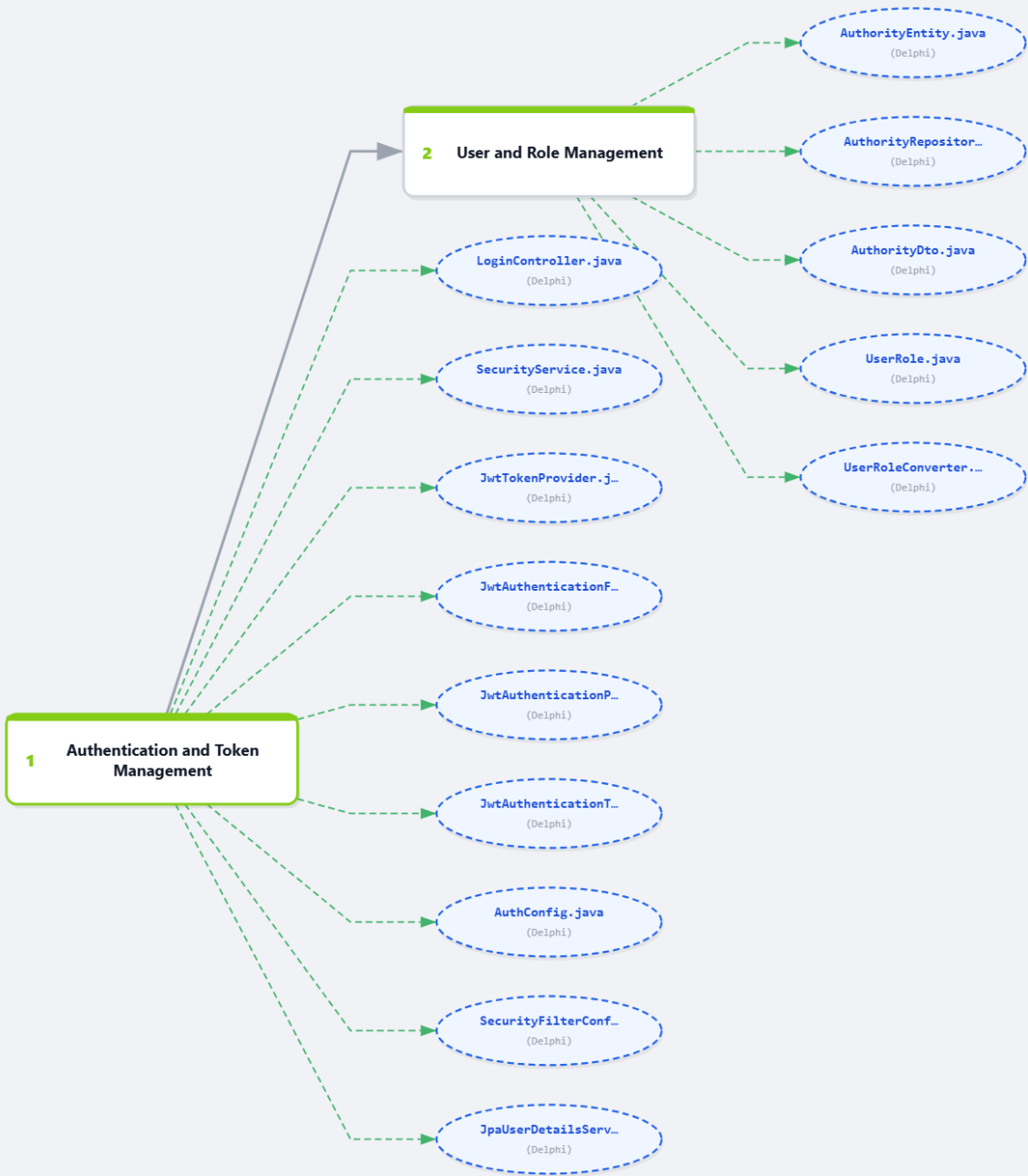
Rental Management





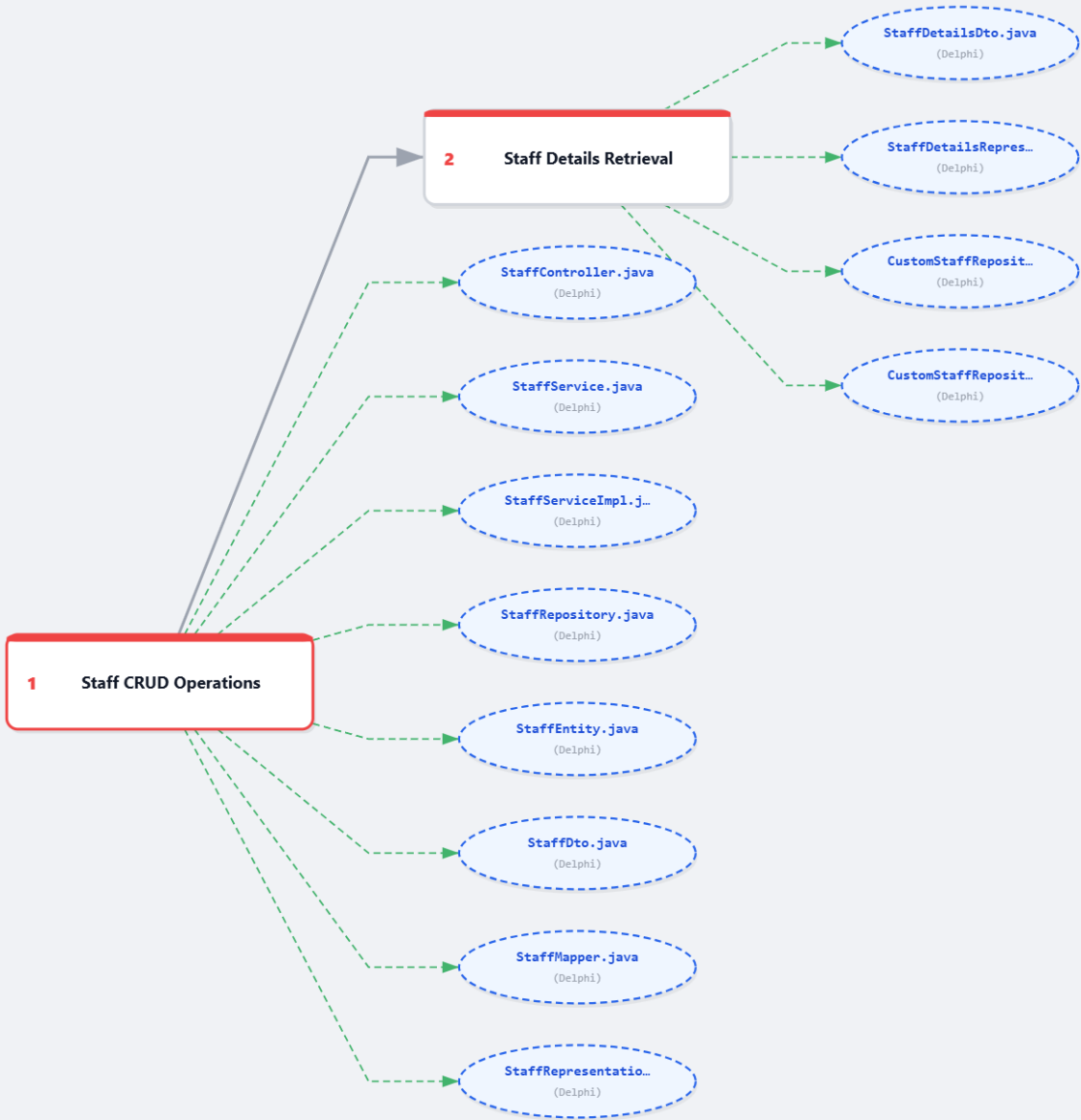
Software

Security and Authentication



Software

Staff Management



Software

Store Management

