MySQL tutorials:

Database Design

Design standard, minimize redundancy, don't loose any data, **Change structure easily**

با طراحی صحیح بانک اطلاعاتی بدون از دست دادن اطلاعات حجم آنها را کاهش دهید و تغییرات ساختاری در آنرا برای خود به یک کابوس تبدیل نکنید.

Resource:

MySQL Tutorial
[A concise introduction to the fundamentals of working with MySQL]
Luke Welling – Laura Thomson – Sams Publishing – ISBN: 0-672-32548-5

Special thanks to:

Fedora Core 3 – GNOME – OpenOffice.org 1.9.104 (2.0 Beta) - Tahoma [TTF Font] - Nimbus Roman No9 L [Font]

iranamp.com
امیر محمد سعید
خرداد ۱۳۸۴

موجودیتها و روابط

در ابتدا بهتره که با دو اصطلاح آشنا بشیم:

Entity: مواردی در دنیای واقعی هستند که می خواهیم اطلاعاتی در باره آنها در بانک اطلاعاتی ذخیره کنیم.

Relationship: روابط بین Entities.

مثلا می خواهیم اطلاعاتی درباره کارکنان و دپارتمانهایی که برای آنها کار می کنند را در یک بانک اطلاعاتی ذخیره کنیم، در این مثال کارکنان و دپارتمانها هرکدام یک entity و این اصطلاح که فلان کاربر برای فلان دپارتمان کار می کند را به عنوان realationship می شناسیم.

از این به بعد entity را به عنوان موجودیت و relationship را به عنوان روابط به کار می برم.

روابط می توانند به ۴ صورت متفاوت تعریف شوند: رابطه یک به یک، رابطه چند به یک، رابطه یک به چند (بستگی به جهت نگاه شما به مسائل دارد، یعنی اگر از طرف دیگر به مسئله نگاه کنید همان رابطه چند به یک است) و در نهایت رابطه چند موجودیت به چند موجودیت دیگر.

خوشبختانه MySQL یک DBMS از نوع Relational است و می توانیم relationship را در آن پیاده سازی کنیم. در واقع می توان relation را همان جدول یا table در نظر گرفت.

اگر با جداول آشنا نیستید! یک جدول را در زیر ببینید:

employee

employeeID	name	job	departmentID
4242	Abbas Gholipur	DBA	128
4157	Reza Abbasi	Programmer	42
3659	Behzad Vatany	System Admin.	128

همانطور که می بینید این جدول اطلاعات مربوط به کارکنان یک شرکت را داراست.

Columns or Attributes:

در هر جدول بانک اطلاعاتی هر ستون تشریح کننده قسمتی از اطلاعات است که هر رکورد در جدول آنرا داراست. عبارات column به معنای ستون و attribute به معنی صفت و خصوصیت بسیار زیاد به جای هم استفاده می شوند اما column چیزی است که در جداول بانک طالاعاتی وجود دارد و attribute چیزی است که در دنیای واقعی به عنوان صفت برای یک موجودیت استفاده دارد.

Rows or Records or Tuples:

در جدول بالا در هر ردیف اطلاعات مخصوص به یک موجودیت ذخیره شده است که اصطلاحا به آن ردیف، رکورد و یا چندتایی خصوصیتی گفته می شود.

Keys:

یک Superkey در واقع یک ستون (یا مجموعه ای از چند ستون) است که برای شناسایی یک ردیف خاص مورد استفاده قرار می گیرد. اما key یک Superkey کوچک است. مثلا در جدول بالا شما می توانید از مجموع employeeID و name برای شناسایی یک ردیف خاص استفاده کنید و همچنین می توانید از مجموع همه ستونها برای شناسایی یک ردیف خاص استفاده کنید که در همه این موارد شما از Superkeys استفاده کردید اما در این مثال شما نیاز به استفاده از همه این موارد را ندارید و به سادگی با استفاده از employeeID به عنوان یک Key می توانید هر ردیف دلخواه را شناسایی کنید، می بینید که Key در واقع یک Superkey کوچک شده است که برای شناسایی یک ردیف خاص مورد استفاده قرار می گیرد.

یک بار دیگر به جدول نگاه کنید، ما برای شناسایی یک ردیف خاص می توانیم هم از employeeID و هم از name استفاده کنیم که به این دو ستون Candidate Keys می گوییم چون اینها Key های نامزد برای Primary Key هستند. یک primary key یک ستون است که ما برای شناسایی هر ردیف دلخواه از آن استفاده می کنیم. در این مثال ما از employeeID به عنوان primary key استفاده می کنیم چون وجود دو شخص با نام یکسان چندان دور از عقل نیست!

حالا Foreign Keys بیانگر ارتباط بین جدوال مختلف است مثلا در جدول بالا می بینید که یک ستون به نام departmentID داریم که دارای یک عدد است و اطلاعات مربوط به هر Department در یک جدول دیگر ذخیره شده است که رابط بین این دو جدول همین departmentID به عنوان یک Foreign Key است.

Functional Dependency:

عبارت Functional Dependency بسیار کمتر از عباراتی که تا به حال بحث شد کاربرد دارد اما به هر حال آشنایی با آن لازم است. اگر بین ستون A و ستون B یک functional dependency وجود داشته باشد آنگاه مقدار ستون A مقدار ستون B را مشخص می کند مثلا در مثال قبل مقدار ستون employeeID مقدار ستون name را مشخص میکند چون برای موجودیت با employeeID خاص تنها یک name وجود دارد (همچنین تنها یک job و departmentID).

Schemas:

عبارت Schema یا Database Schema به سادگی یعنی ساختار طراحی یک بانک اطلاعاتی که به معنی شکل بانک اطلاعاتی بدون هیچ گونه اطلاعات ذخیره شده در آن است.

نحوه توضیح Schema به این گونه است:

employee(employeeID, name, job, departmentID)

یک ستون که primary key باشد (در این مثال employeeID) با یک خط ممتد در زیر آن و یک ستون که foreign key باشد با یک خط چین نمایش می دهیم، ستونی که هم primary key و هم foreign key باشد را با هر دو خط ممتد و خط چین در زیر آن نشان می دهیم.

اصول طراحی بانک اطلاعاتی

هنگامی که می خواهید یک بانک اطلاعاتی را طراحی کنید دو مورد بسیار مهم را باید بررسی کنید.

اولا که چه اطلاعاتی را می خواهید در بانک اطلاعاتی ذخیره کنید و دوما قراره چه سؤالاتی از بانک اطلاعاتی بپرسید (query). ساختار بانک طالاعاتی باید به گونه ای طراحی شود که redundancy و data anomaly نداشته باشد.

زوائد در برابر از دست دادن اطلاعات یا Redundancy versus Loss of Data

هنگام طراحی یک بانک اطلاعاتی ما می خواهیم به نحوی که هیچگونه اطلاعاتی را از دست ندهیم حشو و زوائد را نیز به حداقل ممکن برسانیم. زوائد یا Redundancy به معنای تکرار یکسری اطلاعات خاص در یک جدول و یا جداول یک بانک اطلاعاتی است.

فرض کنید که به جای اینکه در مثال بالا دو جدول employee و department داشتیم تنها یک جدول employeeDepartment داشتیم، در این حالت می بایست یک ستون دیگر به نام departmentName به ساختار جدول اضافه میکردیم تا نام دپارتمان مشخص شود، یعنی ساختار این چنین میشد:

employeeDepartment(employeeID, name, job, departmentID, departmentName)

که یعنی مثلا برای هر رکورد با مقدار ستون departmentID برابر با ۱۲۸ باید در قسمت

departmentName مقدار Research and Development را وارد کنیم. شکل کلی جدول را در زیر می بینید:

employeeDepartment

employeeID	name	job	departmentID	departmentName
4242	Abbas Gholipur	DBA	128	Research and Development
4157	Reza Abbasi	Programmer	42	Finance
3659	Behzad Vatany	System Admin.	128	Research and Development

همونطور که میبینید این نوع طراحی یعنی تکرار departmentName بارها و بارها.
باید بانک اطلاعاتی را اینطور طراحی کنیم:

employee(employeeID, name, job, departmentID)
department(departmentID, name)

در این حالت نام هر دپارتمان تنها یک بار در بانک اطلاعاتی نوشته می شود و این هم باعث اشغال فضای کمتر و هم جلوگیری از مشکلات آتی است.

Anomalies:
این بحث تا حدودی پیشرفته تر است و هنگامی که در بانک اطلاعاتی یک اشتباه طراحی داشته باشیم بروز می کند، ۳ نوع آنومالی وجود دارد:

آنومالیهای ورود اطلاعات:
فرض کنید به جدولی که در بالا به اشتباه طراحی کردیم که هم نام و هم ID هر دپارتمان را در یک جدول گذاشتیم قرار باشد یک شخص جدید وارد شود، در این حالت اگر شما departmentID را ۴۲ و departmentName را Developing وارد کنید! در این ساختار مشخص نیست که کدام یک از این موارد درست است.

آنومالیهای حذف اطلاعات:
فرض کنید که همه اشخاصی که برای دپارتمان Research and Development با departmentID شماره ۱۲۸ کار میکنند در یک روز شرکت را ترک کنند، و شما بخواهید اطلاعات آنها را از بانک اطلاعاتی پاک کنید، بعد از پاک کردن نام آنها شرکت شما دیگر دپارتمان Research and Development نخواهد داشت(میشود یک شرکت ایرانی)!

آنومالیهای ویرایش اطلاعات:
حالا فرض کنید که دپارتمان شماره ۱۲۸ تصمیم بگیره که نام خودشو عوض کنه و بذاره Emerging Technology، حالا شما باید برای همه رکوردها این مورد را تغییر دهید و کافی است یکی را فراموش کنید!

متغیرهای خالی یا Null values:

قانون آخر در طراحی یک بانک اطلاعاتی این است که از تعداد زیاد ستونهای خالی در یک جدول باید پرهیز شود، مثلا اگر در هر صد نفر کارمند شرکت تنها یکی دکترا دارد نباید در جدول یک ستون جدید باز کنیم و در ۹۹ مورد آنرا خالی بگذاریم، به جای ان باید یک جدول دیگر طراحی کنیم که employeeID کارکنان مدرک دکترا را در خود نگه دارد.

نرمال سازی یا Normalization:

نرمال سازی فرآیندی است که در آن مشکلات طراحی یک بانک اطلاعاتی را برطرف می کنیم. در این فرآیند یک سری قوانین وجود دارد که با رعایت آنها طراحی ما در یک سطح از Normalization قرار می گیرد، مثلا با یک سری قوانین ساده طراحی ما در سطح یک از Normalization قرار می گیرد، برای اینکه هر طراحی در سطح بعدی از Normalization قرار بگیره باید در سطوح زیرین خودش هم وجود داشته باشد. مثلا طراحی ای که در سطح دوم قرار دارد حتما در سطح اول هم قرار دارد و طراحی ای که در سطح سوم قرار دارد حتما در سطوح یک و دو نیز قرار دارد.

اولین شکل نرمال:

در اولین شکل نرمال که گاها 1NF هم اطلاق می شود مقدار هر ستون باید atomic باشد. که یعنی در هر ستون تنها باید یک مقدار ذخیره شود نه مجموعه ای از مقادیر و یا اطلاعات یک ستون از بانک اطلاعاتی، جدول زیر را در نظر بگیرید:

employee

employeeID	name	job	departmentID	skills
4242	Abbas Gholipur	DBA	128	DB2
4157	Reza Abbasi	Programmer	42	C, Perl, Java
3659	Behzad Vatany	System Admin.	128	NT, Linux

این طراحی در 1NF قرار ندارد چون مقدار ستون skills در بعضی از ردیفها atomic نیست، برای قرار دادن این جدول در 1NF راههای متفاوتی وجود دارد:

employee

employeeID	name	job	departmentID	skills
4242	Abbas Gholipur	DBA	128	DB2
4157	Reza Abbasi	Programmer	42	C
4157	Reza Abbasi	Programmer	42	Perl
4157	Reza Abbasi	Programmer	42	Java
3659	Behzad Vatany	System Admin.	128	NT
3659	Behzad Vatany	System Admin.	128	Linux

خوب مثل این که جدول ما حالا در حالت 1NF قرار داره، خیالم راحت شد.

اما صبر کن این طراحی مشکل داره! هر کی گفت مشکلش چیه؟ بله Data Redundancy اونهم از نوع خفنش!

پس باید جور دیگری اونو طراحی کرد:

employee

employeeID	name	job	departmentID
4242	Abbas Gholipur	DBA	128
4157	Reza Abbasi	Programmer	42
3659	Behzad Vatany	System Admin.	128

employeeSkills

employeeID	skill
4242	DB2
4157	C
4157	Perl
4157	Java
3659	NT
3659	Linux

خوب حالا جدول ما هم در 1NF قرار دارد و هم حداقل redundancy رو داره. طراحی صحیح بانکهای اطلاعاتی به شدت وابسته به تجربه شما است.

دومین شکل نرمال:
بعد از اینکه طراحی ما آزمون استاندارد 1NF رو پشت سر گذاشت با آزمون سخت تر 2NF روبرو می شود که تا حدودی فهم آن سخت تر است، قانون این مرحله می گوید که در هر ردیف همه ستونها به جز ستونهای متعلق به primary key باید کاملا نسبت به primary key ها Functional Dependent باشند در حالی که باید در حالت 1NF نیز باشد، سخت شد، نه؟
برای درک بهتر به مثال قبل بر میگردیم، در طراحی اولین حالت برای اینکه جدول ما در 1NF قرار بگیره.
در اون جدول به ازای هر کاربر چند ردیف داشتیم که هر ردیف در ستون skill تفاوت داشت. اون جدول در 1NF بود ولی آیا در 2NF نیز هست، مسلما نه. گفتیم که primary key باید هر ردیف خاص را در جدول شناسایی کند ولی در اون طراحی employeeID به تنهایی نمی تواند کمک کند و باید مجموع آن با skill به عنوان primary key در نظر گرفته شود تا هر ستون خاص مشخص شود، پس ساختار این چنین می شود:

employee(employeeID, name, job, departmentID, skill)

خوب حالا باید از خودمون بپرسیم که functional dependency اینجا چه جوریه:
employeeID, skill ---> name, job, departmentID
اما ما همچنین داریم:
employeeID ---> name, job, departmentID

خوب این یعنی که بقیه ستونها با employeeID به تنها که در اینجا یک primary key نیست functional dependency دارند در حالی که همین ستونها با کل مجموعه employeeID و skill هم که

نقش primary key رو داره بازهم functional dependency دارن پس این جدول در 2NF نیست (سخته نه؟!).

خوب چه جوری این مشکل رو برطرف کنیم و طراحی رو در 2NF قرار بدیم، درست حدس زدید، راه حل دوم برای مثال قبل طراحی ما رو در 2NF قرار میده، یعنی:

employee(employeeID, name, job, departmentID)
employeeSkills(employeeID, skill)

سومین حالت نرمال:

برای تشریح ساده این حالت بسیار سخت این موارد را بدانید:

طرف چپ یک عبارت functional dependency باید حتما یک superkey باشد (نیازی نیست حالت minimal آن یا همان key باشد.)

طرف راست یک عبارت functional dependency هم جزئی از هر key در جدول باشد.

در اکثر موارد رعایت قانون اول کافی است.

سطوح بالاتری نیز داریم (چهارم، پنجم و ...) اما تنها اهداف آکادمیک دارند و نیازی با آشنایی و استفاده از انها در طراحی بانکهای اطلاعاتی نیست.