درس طرحی پایگاه داده

تمرین تئوری سری چهارم ۱۴۰۴/۰۳/۱۲

متین باقری ۴۰۲۱۰۵۷۲۷

---

۱) آ) نادرست، هدف normalization کاهش anomaly و data redundancy است که معمولا با افزایش تعداد جداول همراه است.

ب) نادرست، با افزایش تعداد جداول، نیاز به انجام join های بیشتری داریم.

ج) اگر یک جدول فقط یک ستون داشته باشد و مقادیر آن غیرتکراری باشند، یعنی آن ستون کلید اصلی است و هیچ وابستگی تابعی یا وابستگی چندمقداری یا وابستگی join وجود ندارد. درست

اما اگر جدول چندین ستون داشته باشد و تنها یکی از آنها دارای مقادیر غیر تکراری باشد، اطلاعات کافی نیست. نادرست

د) نادرست. ممکن است نیاز به بروزرسانی secondary index ایجاد شده و این کار زمان بر است.

ه) درست. bitmap index برای ستون هایی با مقادیر محدود استفاده می‌شود، در حالی که PK باید unique باشد و به همین علت شامل تعداد زیادی مقادیر مختلف است.

و) نادرست. معمولا با افزایش تعداد جداول (رابطه ها) همراه است و رابطه ها شامل کلید ها خارجی از دیگر رابطه ها شده و به طور کلی سادگی خاصی برای رابطه ها به همراه ندارد.

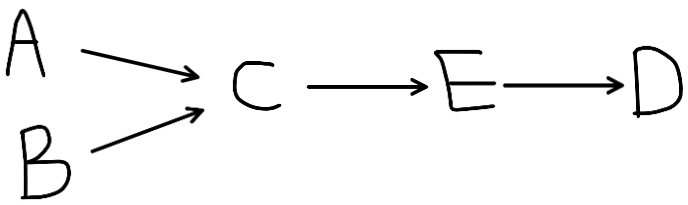
۲) آ)

$C \rightarrow D, CD \rightarrow E$ ::: $C \rightarrow D, C \rightarrow E$

$A \rightarrow C, A \rightarrow CE, C \rightarrow E$ ::: $A \rightarrow C, C \rightarrow E$

$C \rightarrow D, C \rightarrow E, E \rightarrow D$ ::: $C \rightarrow E, E \rightarrow D$

$\{B \rightarrow C, C \rightarrow E, E \rightarrow D, A \rightarrow C\}$



ب)

The only candidate key: AB

number of super keys: $8 = 2^3$ (including A & B , may or may not include C & D & E)

ج) با توجه به دیاگرام بخش آ، dependency preserving را بررسی می‌کنیم. باید تمام FD های رسم شده را بتوان در یک جدول یافت. برای بررسی lossless بودن، در صورتی که مشخصه ای که دو جدول بر روی آن join میخورند R2 PK باشد، lossless است، ولی در غیر این صورت پس از join، سطر جدید ایجاد شده و lossless نیست.

- lossless **نیست**. dependency preserving **دارد**.
- lossless **هست**. dependency preserving **ندارد**.
- lossless **هست**. dependency preserving **دارد**.

د) بله، بسیار شبیه R1(A, B, C) ،R2(C, D, E) است و همه FD ها را حفظ کرده و lossless نیز هست. توضیحات نحوه بررسی آن مانند بخش قبل است.

۳) آ) درج: نمیتوان موارد زیر را بدون اینکه قرض گرفتنی اتفاق افتاده باشد به جدول اضافه کرد: عضو، آیتم، نویسنده، ...

به‌روزرسانی: برای به‌روزرسانی یک عضو یا یک کتاب یا ... باید تمام سطر هایی که مربوط به آن است را به‌روزرسانی کنیم.

حذف: اگر یک قرض گرفتن، تنها سطر موجود مربوط به یک کتاب یا عضو یا ... باشد، با حذف آن سطر، آن عضو یا کتاب یا ... به طور کامل از جدول حذف می‌شود.

ب) ابتدا با توجه به FD های داده شده، یک طراحی جدید ایجاد کرده و سپس BCNF بودن آن را بررسی می‌کنیم.

Member (ID, name, address, type)

Item (ID, title, type, format, AuthorID)

Author (ID, name)

Fee (MemberType, ItemType, LateFee)

Loan (ID, MemberID, ItemID, LoanDate, DueDate, ReturnDate)

1NF: no multi-valued attribute or composite attribute

2NF: single attribute PK

3NF: no transitive dependency

BCNF: every determinant of Non Trivial irreducible FD is a candidate key

همه شروط بالا برقرار اند. شرط آخر برقرار است چون سمت چپ همه FD ها، PK جداول ما اند.

ج) برای تغییر امانت سیاست، مقدار LateFee را در جدول Fee آپدیت می‌کنیم. برای افزودن عضو جدید به جدول member و برای افزودن کتاب جدید به جدول item سطر اضافه می‌کنیم.

۴) آ)

R (A, B, C, D, E)

F = {AB→C, CD→E, DE→B}

A و D در سمت راست هیچ FD وجود ندارند، پس همه SKها شامل A و D هستند.

Core of SuperKeys: {A, B, D}, {A, C, D}, {A, D, E}

هر ترکیبی از attributeها که شامل یکی از سه مورد بالا باشد، SK است. در کل ۷ SK داریم:

3 Attribute SuperKeys: {A, B, D}, {A, C, D}, {A, D, E}

4 Attribute SuperKeys: {A, B , C, D }, {A, B, D, E}, {A, C, D, E}

5 Attribute SuperKeys: {A, B, C, D, E}

ب)

R (A, B, C, D, E, F, G)

FD = {~~BCD→E, BCD→B, BCD→F~~, BD→A, F →G, E→F, C→E}

۳ FD سمت چپ از دیگر FD ها به دست می‌آیند و در واقع اطلاعات جدیدی به ما نمی‌دهند، پس می‌توان آنها را در نظر نگرفت. به کمک ۴ FD دیگر، جداول زیر را می‌سازیم:

R1 (C, E)

R2 (E, F)

R3 (F, G)

R4 (B, D, A)

R5 (B, C, D)

۴ جدول بالا، تمام FD ها را با رعایت 3NF پوشش می‌دهند. جدول ۵ ام به ما امکان واکشی همه attribute های یک tupple از R (A, B, C, D, E, F, G) را می‌دهد. پس در کل به ۵ جدول نیاز داریم.

---

۵) آ) Clustered Indexing: ردیف‌های جدول به‌طور فیزیکی بر اساس مقدار ستون ایندکس‌شده مرتب می‌شوند. هر جدول فقط می‌تواند یک ایندکس خوشه‌ای داشته باشد، زیرا داده‌ها به همین ترتیب ذخیره می‌شوند. این نوع ایندکس باعث افزایش سرعت جستجوهای شامل مرتب‌سازی یا محدوده‌های خاص می‌شود.

Non-Clustered Indexing: ایندکس به‌طور جداگانه از داده‌های اصلی ذخیره می‌شود و ترتیب منطقی ایجاد می‌کند، نه ترتیب فیزیکی. یک جدول می‌تواند چندین ایندکس غیرخوشه‌ای داشته باشد، که باعث افزایش سرعت جستجوهای خاص می‌شود بدون اینکه ترتیب ذخیره داده‌ها تغییر کند.

ایندکس خوشه‌ای ترتیب فیزیکی داده‌ها را تعیین می‌کند، در حالی که ایندکس غیرخوشه‌ای ساختاری جداگانه برای جستجوهای سریع فراهم می‌نماید.

ب) i. non cluster indexing, hash on username: می‌توان به سرعت hash یوزرنیم خواسته شده را محاسبه کرد و به جستجوی آن پرداخت.

ii. cluster indexing, b+ tree on age: چون به دنبال جست‌وجوی یک محدوده زمانی خاص هستیم، بهتر است tuple های مورد هدف ما کنار هم ذخیره شده باشند و کافی است اولین و آخرین tuple خواسته شده را پیدا کنیم.

iii. composite non cluster indexing, b+ on city and age: بر اساس شهر فیلتر کرده و بر اساس سن مرتب میکند تا پس از آن نیاز به order by نداشته باشیم.

۶) آ)

- list بر روی نوع محصول: محصولات تعداد محدودی نوع دارند و اینگونه query هایی که روی یک دسته خاص اجرا می‌شوند، سرعت بیشتری خواهند داشت.
- اگر منبع لاگ ها اهمیت زیادی دارد: hash روی منبع دریافت لاگ: می‌توان با سرعت بیشتری لاگ های مربوط به یک منبع خاص را بررسی کرد.

در غیر این صورت: round robin، درخواست‌ها را به طور یکنواخت بین گره‌ها توزیع می‌کند و باعث متعادل شدن بار پردازشی و جلوگیری از تراکم بیش از حد در یک گره می‌شود.

- hash بر روی شناسه کاربر: توزیع متوازن کاربران
- range بر روی تاریخ فروش: می‌توان با دانستن بازه ی فروش یک کالا، سریع تر آن را جست‌وجو کرد و یا در یک بازه زمانی query خواسته شده را اجرا کرد.

ب) درج: با توجه به نوع افراز و مقدار attribute مربوطه در tuple جدید، جدول مربوطه پیدا شده و در آن درج صورت می‌گیرد.

به‌روزرسانی: اگر با انجام به‌روزرسانی، جدولی که tuple مورد نظر در آن قرار دارد تغییری نکند، با توجه به مقدار attribute مربوطه در tuple مورد نظر، آن tuple در جدول مربوطه پیدا شده و به‌روزرسانی انجام می‌شود.

حذف: با توجه به نوع افراز و مقدار attribute مربوطه در tuple مورد نظر، جدول مربوطه پیدا شده و آن tuple از آن حذف می‌شود.

اما اگر با انجام به‌روزرسانی، جدولی که tuple مورد نظر در آن قرار دارد تغییری کند، ابتدا tuple فعلی حذف شده و سپس مقدار به‌روزرسانی شده آن در جدول جدید درج می‌شود.

ج)

- **اسکن کل جدول:** با توجه به اینکه عملیات ها به طور موازی انجام می‌شوند، هرچه افراز متوازن تر باشد و کاردینالیتی جداول به هم نزدیک تر باشند، بهتر است.

range: بسته به نوع index attribute شده می‌تواند خوب یا بد باشد، تضمینی بر خوب بودن آن وجود ندارد

hash: نسبتا خوب و متوازن

round bin: بسیار متوازن، بسیار عالی

- **پرس‌وجوی نقطه‌ای:**

range: نسبتا خوب، به سراغ بررسی جدولی که range آن شامل نقطه مورد نظر است می‌رویم.

hash: خیلی خوب، مستقیما به سراغ جدول متناظر با hash مورد نظر می‌رویم.

round bin: خوب نیست. چون افراز بر پایه منطق خاصی نبوده، باید تمام جداول جست‌وجو شوند.

- **پرس‌وجوی بازه‌ای:**

range: بهترین حالت، ساخته شده دقیقا برای همین منظور

hash: نسبتا بد، باید برای هر مقدار در بازه مشخص شده، hash محاسبه شده و به سراغ جدول مربوطه برویم.

round bin: خوب نیست. چون افراز بر پایه منطق خاصی نبوده، باید تمام جداول جست‌وجو شوند.