

بنام خدا

پایگاه داده ۲

Design of Data Warehouse

بصیری

دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

مراجع

- Han, Jiawei, Micheline Kamber, and Data Mining. "Concepts and techniques." *Morgan Kaufmann* 340 (2006): 94104-3205.
- Kimball, Raiph. *The data warehouse toolkit*. John Wiley & Sons, 2006.
- Inmon, William H. *Building the data warehouse*. John wiley & sons, 2005.

Data Warehouse Design Process

- Top-down, bottom-up approaches or a combination of both
 - Top-down: Starts with overall design and planning (mature)
 - Bottom-up: Starts with experiments and prototypes (rapid)
- Typical data warehouse design process
 - Choose a **business process** to model, e.g., orders, invoices, etc.
 - Choose the **grain** (atomic level of data) of the business process
 - Choose the **dimensions** that will apply to each fact table record
 - Choose the **measure** that will populate each fact table record

-
دو نگاه وجود داره توی بحث انبار داده:

بالا به پایین: ما اگر میخوایم بیایم طراحی انبار داده رو برای حوزه ای انجام بدیم باید اون حوزه رو کامل بدونیم ینی از بالا همه چی رو مسلط باشیم و بعد می تونیم شروع کنیم به طراحی انبار داده و دیتا مارت ها - این نگاه سختگیرانه است چون طراحی انبار داده برای ورود یک حوزه جدید خیلی سخته که تمام بیزینس رو بدونیم و با علم این که کل بیزینس رو بدونیم حالا بیایم طراحی بکنیم ولی اگر همچین نگاهی وجود داشت طراحی انبار داده خیلی کیفیت بالایی داشت - این نگاه بالغ تر است پایین به بالا: نگاه تجربی تری است - ینی از یک دیتامارتی شروع میکنیم و می ریم جلو و اگر لازم بود تغییراتی توی طراحی انبار داده انجام بشه به مرور اتفاق می افته و کامل تر میشه - نگاه سریع تری است و در عمل هم نگاه دوم به کار میاد مگر اینکه ما تجربه خوبی داشته باشیم توی اون حوزه که میخوایم فعالیت بکنیم

پس اول باید نگاه کنیم انبار داده برای چی می خوایم ایجاد بکنیم و هدف چیه و بعد ببینیم نسبت به کجاها مسلط هستیم و از همون جا شروع بکنیم و می ریم جلو

دیتامارت : برای بخش مالی و حسابداری سازمان می تونیم دیتامارت متفاوتی ببینیم ینی یکسری فکت و dimensionهایی داریم که مرتبط با حوزه حسابداری و مالی اون سازمان هست برای اینکه بخوایم این کارو انجام بدیم و قبل از ورودی ممکنه دانش کمی داشته باشیم پس اگر شناخت کمی داشتیم باید به اون فعالیت ها و روال هایی که توی اون سازمان اتفاق می افته توجه کنیم

grain : ریزدانگی داده

- ریزدانگی به این معنا است که تا چه سطحی لازم است ریز اطلاعات نمایش داده شود.
- با توجه به نیاز سازمان ریزدانگی باید تعیین شود.
- ریزدانگی در فکت و به تبع آن در بعدها نمایان می شود. مثلاً می خواهیم اطلاعات را به تفکیک روز ذخیره نماییم یا به تفکیک ماه.
- طبیعتاً ریزدانگی تاثیر مستقیمی روی حجم داده ذخیره شده خواهد داشت.

اینکه ما وقتی میخوایم داده ها رو توی انبار داده ذخیره بکنیم توی چه سطحی از ریزدانگی میخوایم نگه داریم و بررسیشون بکنیم ینی توی اون حوزه سازمان تا چه حد می خواد ریز اطلاعاتش رو نگه داره مثلا اطلاعات در حد ماه براش کفایت میکنه یا نه براش مهم نیست ماه و می خواد به تفکیک روزها مشاهده بکنه --> این ریزدانگی است

این ریزدانگی هم به dimension مرتبط است و هم به فکت

اینجا دوتا بحث است یکی بحث نیاز اون سازمان است و یک بحث هم فنی است --> این سطح که می خوایم اینقدر ریز اطلاعات رو مشاهده بکنیم و داشته باشیم و ذخیره بکنیم چقدر هزینه و حافظه و... باید براش در نظر گرفته بشه مثلا می خوایم بکاپ بگیریم یک بحث نیاز و یک بحث هم بحث حافظه ای است که اشغال میشه مثلا ما داریم Snapshot هر روز بکاپ می گیریم از سرور حالا چقدر این بکاپ رو نگه داریم اگر بحث فنی مطرح نشده ینی مثلا مهم نباشه چقدر حجم می گیره یا .. ممکنه یک مدیر بگه همه رو نگه داره که این کمترین ریسک است (که این نیاز واقعی نیست) پس این نیاز باید مبتنی بر واقعیت هم باشه و حتی میشه به جای هر روز، هر ساعت بکاپ بگیریم پس باید بهش بگیم میخوای این قدر هزینه بکنی؟! --> پس یک بحث میشه بحث نیاز و یک بحث هم میشه هزینه و نیاز واقعی باید از این وسط در بیاد که برسیم به اون چیزی که مد نظر است توی اون فرایندهایی که شناختیم و بیزینس رو یکمی درک کردیم باید برسیم به این که چی برای اون سازمان مهم است --> شاخص ها یا KPIها عملا باید در بیاد و این شاخص ها که در بیاد یه جوری مژرها هم در میاد (مژرها عملا توی دل KPIها وجود دارن) حالا اینکه این مژرها از چه منظری بررسی بشه باعث میشه این dimensionها کم کم شفاف بشه

مهم!

انواع جداول واقعیت (Fact Table)

توی پروژه از هر فکتی حداقل باید
یکی داشته باشیم

Transaction Fact Table-

Periodic Snapshot Fact Table-

- وضعیت یک موجودیت اصلی را در دوره های زمانی مشخص ذخیره می کند.
- معمولاً لازم است فکتهای مختلفی با ریزدانگی های متفاوت زمانی از این نوع وجود داشته باشد، مثلاً روزانه و ماهانه

Accumulating Snapshot Fact Table-

Fact less Fact Table-

- برای ارتباط بین دو بعد استفاده می شود

به صورت عمده با نگاه کلی ما باید 4 نوع فکت طراحی بکنیم:

توی اسلاید قبلی می‌گه که باید بیزینس رو بشناسیم و بدونیم چیه و dimension ها و مژرهاشو و فکت هاشو بشناسیم و در بیاریم

حالا می‌خوایم طراحی بکنیم:

1- فکت Transaction:

خیلی شبیه به تراکنش های جدول های سورس است مثلاً یک فروشگاه داره تراکنش انجام میده و فاکتورهای که داره صادر میشه معلوم میشه چه ادمی چی ایتمی رو فروخته و... این جزئیاتی است که داره ثبت میشه اونجا خیلی شبیه اینو ما توی انبار داده میایم ذخیره میکنیم --> اگر اون جدول

سورس است برای تراکنش های خریدی که انجام شده 1 میلیون رکورد برای هر روز داره ثبت میشه ما توی فکت Transaction یک میلیون رکورد داریم --> خب چرا از همون جدول سورس استفاده نمی‌کنیم؟ 1- جدول سورسی که در اونجا در اختیارمون است با ادبیات فکت و dimension نوشته نشده و برای اینکه بخواد اطلاعاتی که می‌خواد رو بررسی کنه باید کلی جدول رو جوین بکنه ولی اینجا اینا خیلی شسته و رفته شده اومدن و dimension هایی هم که برای بررسی این تراکنش ها لازم است دم دستش آوردیم

2- ممکنه مژرهایی رو اضافه کنیم که اینا داخل جدول سورس به علت افزونگی و... وجود نداشته باشه مثلاً مانده بعد از تراکنش خیلی منطقی نیست توی جدول های سورس وجود داشته باشه ولی ما اینجا می‌ایم

نکته: فکت Transaction از لحاظ تعداد رکوردها و ریزدانگی خیلی شبیه جدول های سورس است ولی به علت بالایی هایی که گفتیم فکت های Transaction توی سازمان ها خیلی پرطرفدار هستن --> خیلی شبیه جدول های سورس هستن و خیلی دم دست تر است و خیلی از دغدغه هایی که توی جدول های سورس داشتیم الان اینجا توی این فکته هست و دغدغه نداریم براش چون سرعت پاسخ دهی رو می‌خوایم خیلی خیلی بیشتر بکنیم پس دغدغه که داره اینجا سرعت است و افزونگی نیست

مثال: تراکنش هایی که توی بیمارستان داریم عبارتند از: پرداخت - نوبت دهی - نسخه ها و..
نکته: وقتی فکت های تراکنشی رو سعی کردیم متوجه بشیم فکت های دیگه هم شفاف خواهند شد
2- فکت Periodic Snapshot:

فکت هایی که یک اسنپ شات زمانی رو دارن داخل خودشون نگه میدارن --> یکسری موجودیت های مهم اون سازمان رو که براش داره تراکنش هایی ثبت میشه باید درست شناسایی بکنیم و برای اون لایه بالای سازمان مهمه این موجودیتی که مهم است و تراکنش هایی داره براش ثبت میشه رو در مقاطع زمانی مختلف بتونه داده هاشو ردیابی بکنه --> حالا این مقاطع زمانی که می خواد داده هاشو ردیابی بکنه بسیار مهمه چه مقطعی است مثلا میخواد هر روز بررسی بکنه هر ماه بررسی بکنه یا سال...

بحث ریزدانگی اینجا خیلی واضح نشون داده میشه

پس چه موجودیتی رو توی چه بازه های زمانی میخوایم بیایم بررسی بکنیم

مثلا سپرده یک موجودیت مهم داخل حوزه بانکی است حالا میخوایم مانده سپرده رو بررسی بکنیم:

یک فکت تراکنشی داریم برای سپرده ینی واریز و برداشت سپرده رو داریم ثبت میکنیم

حالا برای فکت اسنپ شاتی --> مقادیری که توی سپرده داریم توی چه مقاطع زمانی می خوایم

بررسیش بکنیم ینی گزارش گیری توی چه رنج زمانی است --> می خوایم مانده های سپرده رو هر

روز مشاهده بکنیم اینجا اسنپ شات زمانی میشه روز --> چجوری میخوایم روزانه ببینیم؟ وقتی که

این ریزدانگی و این بعد زمانی مشخص شد که توی چه بازه زمانی میخواد نگاه بکنه معنیش این

است که اون موجودیت اصلی رو در اون مقاطع زمانی همیشه باید وضعیتش رو داشته باشیم برای

تمام نمونه ها مثل یک اسنپ شات بکاپ فول است ینی ما داریم هر ساعت فول بکاپ می گیریم -->

وضعیت سپرده ها رو توی اسنپ شاتی که انتخاب کردیم که اینجا روز است برای هر روز میخوایم

همشو داشته باشیم پس فول می گیریم ازش و میذاریم کنار --> برای بانک روز بهتر است چون

براساس این روزا بعدا میاد تصمیم گیری میکنه --> حالا اگر برای ماه می خواست براساس ماه

میگیره و فرق بین ماه و روز توی ریزدانگیش است و تفاوتش 30 برابر حجمش است

نکته: اگر n تا سپرده داشته باشیم و بخوایم اسنپ شات رو روزانه ببینیم در مقایسه با ماهانه 30 برابر فرقت است و باید تصمیم گیری بکنیم که کدام رو میخوایم

نکته: بیشترین حجم انبار داده رو فکت های اسنپ شاتی می گیرن --> که این هم اون زمان اسنپ شاته تعیین میکنه که مدیر سازمان تعیین میکنه که چجوری داده ها رو فول بکاپ بگیریم --> که تصمیم گیری در مورد این ینی اینکه حجم انبار داده داره تخمین میخوره که چقدرش گرفته میشه نکته: ما می تونیم فکت تراکنشی سپرده داشته باشیم در کنارش فکت اسنپ شاتی سپرده هم داریم مثلا برای فکت اسنپ شات --> dimension های تایم، سپرده، شعبه، مشتری، نوع سپرده

و مژرهایی که داریم: موجودی سپرده، تعداد تراکنش هایی که مشتری در اون روز انجام داده، تعداد روز بدون تراکنش، جمع گردش بدهکار، جمع گردش بستانکار، میانگین مانده در اون روز، کمترین موجودی اون روز (میشه برای به دست آوردن سود)، حداقل مانده در روز، حداکثر مانده در روز اینجا ریزدانگی روز است برای این فکت الان

حالا برای فکت تراکنشی سپرده این dimension ها رو داریم: شعبه، سپرده، نوع سپرده، زمان، مشتری و مژرهانش عبارتند از: مبلغ تراکنش، موجودی بعد از تراکنش، گردش بدهکار، گردش بستانکار، مانده بعد از تراکنش، مانده بدهکار، مانده بستانکار

نکته: تعداد تراکنش توی ماه مربوط به دیبی 1 میشه ینی operational و مربوط به انبار داده نمیشه --> مثلا اگر 10 تا تراکنش توی روز داشته باشیم 11 رو دیگه نمی تونیم انجام بدیم این ربطی به انبار داده نداره چون ما توی انبار داده هر لحظه دیتا داخلش اضافه نمی کنیم و انبار داده آنلاین نیست بلکه افلاین است --> چون فکت تراکنش رو مثلا روزانه داریم به روز می کنیم و هر لحظه به روز نمیشه

3- فکت ACC:

این فکت ها از جنس اسنپ شاتی ACC هستن

مثلا توی یک مثال ممکنه یک فکت اسنپ شات 70 میلیون رکورد برای یک روز داشته باشه ولی این فکت میگه چه موجودیتی رو داشتی ذخیره میکردی و برای هر موجودیت کلا یک رکورد باید داشته باشی ینی تجمیع شده کل هیستوری ینی آخرین باری که انبار داده به روز شده کی بوده مثلا 2 شب بوده و چه دیتایی رو به روز کرده دیتای تا 12 شب قبل رو --> پس این فکت برای هر سپرده، مانده اش میشه آخرین مانده که 12 شب دیشب داشته --> پس آخرین وضعیت سپرده رو میخواد و میگه کی به روز شده انبار داده تا 12 شب دیشب به روز شده پس میگه تا 12 دیشب هر سپرده چه وضعیتی داشته

مثال: اگر فرض کنیم تعداد سپرده های بانک ملی در گذر زمان تغییری نمیکنه و انبار داده هم از تاریخ 1402/1/1 ایجاد کردیم و الان هم 1402/12/29 است که این میشه 365 روز از روزی که انبار داده رو ایجاد کردیم که توی همه این 365 روز بانک ملی 70 میلیون سپرده داشته: حالا تعداد رکوردهای فکت اسنپ شات میشه $70 * 365$ و تعداد رکوردهای ACC میشه 70 میلیون فقط

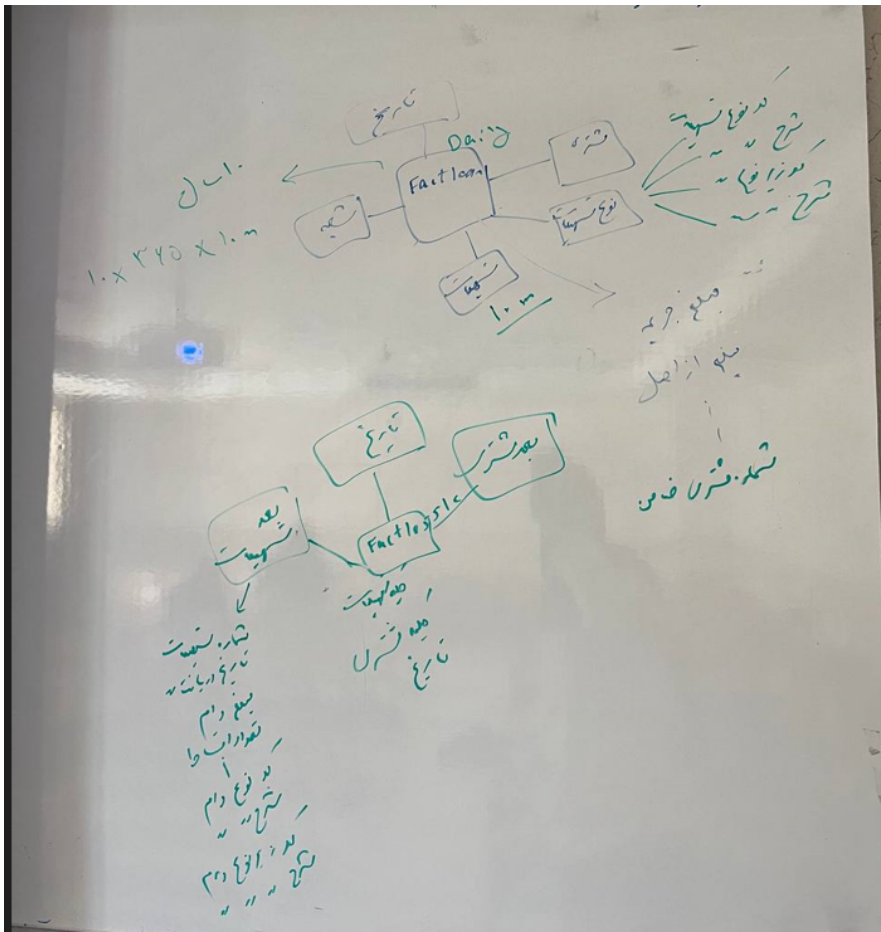
حالا dimensionهایی که داریم عبارتند از: نوع سپرده، شعبه، مشتری و مژره‌اش عبارتند از: مانده سپرده، تعداد کل تراکنش هایی که اون سپرده انجام داده، کل گردش بدهکار سپرده، کل گردش بستانکار سپرده، تعداد روز بدون فعالیت

نکته: هر چی توی اسنپ شات هست اینجا هم هست فقط اینجا dimension زمان نداریم چون آخرین وضعیت رو می‌خوایم و نمی‌خوایم به تاریخچه بریم نگاه کنیم که چی بوده --> پس اگر ما فکت اسنپ شات رو فهمیدیم چیه dimension زمان رو از این فکت حذف میکنیم و می‌رسیم به فکت ACC

4- فکت less Fact:

نیاز داریم بین دوتا یا چند dimension بخوایم ارتباط برقرار کنیم ولی بین فکت و dimension داشتیم اینو قبلا ولی بین چندتا dimension نداشتیم
مژر نداریم اینجا
مثلا

dimension تسهیلات بانک داریم
dimension مشتری هم داریم
ارتباط ضامن با تسهیلات



انواع جداول واقعیت (Fact Table)

Temporal Snapshot Fact Table-

[https://sqlbits.com/\(X\(1\)S\(xkq3bqj54ulh1rmknteod-x3p\)\)/Sessions/Event10/Temporal_Snapshot_Fact_Table](https://sqlbits.com/(X(1)S(xkq3bqj54ulh1rmknteod-x3p))/Sessions/Event10/Temporal_Snapshot_Fact_Table)

Periodic Snapshot Fact Table

Periodic Snapshot Fact Table-

- برای مژره‌های جمع‌پذیر لازم است تصمیم‌گیری شود که به صورت جمع شده یا تراکنشی وارد شوند. مثل مبلغ پرداختی مشتری از اصل تسهیلات

نگاه تراکنشی

مبلغ پرداختی از اصل	تاریخ موثر	کلید تسهیلات
۱۰۰	۹۷/۴/۲۵	۱
۰	۹۷/۴/۲۶	۱
۲۰۰	۹۷/۴/۲۷	۱

مبلغ پرداختی از اصل	تاریخ پرداخت	کلید تسهیلات
۱۰۰	۹۷/۴/۲۵	۱
۲۰۰	۹۷/۴/۲۷	۱



نگاه جمع شده

مبلغ پرداختی از اصل	تاریخ موثر	کلید تسهیلات
۱۰۰	۹۷/۴/۲۵	۱
۱۰۰	۹۷/۴/۲۶	۱
۳۰۰	۹۷/۴/۲۷	۱

توی فکت های که از جنس اسنپ شاتی هستند در مورد مژرها باید فکر کنیم که به چه صورت باید ذخیره بکنیم: بعضی از مژرها خاصیت جمع پذیری دارند و باید تصمیم گیری بکنیم که وقتی داریم روزها اضافه میشه توی فکت ما این مژرها که از جنس جمع پذیری هستند به صورت تجمیعی وارد بکنیم داخل فکت یا از جنس تراکنش ببینیمش و تغییر و اتفاقی که توی اون روز افتاده رو فقط بیایم ثبت بکنیم

مثال:

فکت روزانه است پس باید روز 26ام هم داشته باشیم توی فکت

نکته: این دوتا نگاه هردوشون خوبه و گاهی اوقات می ارزه هر دوتاشو بیاریم توی دو ستون متفاوت ولی معمولاً یکی از اینا باشه کافیه --> دغدغه ای که باید این وسط داشته باشیم اینه که کاربر انبار داده ما از نوع نگاه ما آگاه باشه ینی باید اطلاعات رو منتقل کنیم بهش

فکت تراکنشی و Daily این است که در تراکنشی این ID رکوردی (تراکنشی) نداشته باشه، رکوردی ندارد.
ولی در Daily fact ، برای روزهایی که تراکنشی نبوده هم رکوردی ثبت می شود.

FACT TABLE

- ❖ وجود کلید نال مربوط به یک بعد در فکت منطقی نیست
- ❖ معمولاً در صورت وجود یک رکورد با مقدار کلید ۱- در بعد وجود دارد که در فکت نیز ۱- برای آن بعد ثبت می شود. این رکورد نماینده مقدار نامشخص است.
- ❖ وجود مقادیر نال برای مژرها ممکن است.

نال به عنوان کلید dimension داخل فکت معنایی نداره

مثلا برای بعد شعبه کلیدش میشه 1- و نام شعبه میشه نامشخص و کد استان میشه 1- و نام استان میشه نامشخص و..

نکته: مزیتی که 1- نسبت به نال داره: توی aggregation ها، توی جویین ها و... است و دیگه کار رو خراب نمی کنه مثلا جویین که میخواستیم بکنیم نال با اون ور جویین نمیشد و همین باعث میشد که کار خراب بشه

Dimension

- وجود یک بعد به کاربرد آن بستگی دارد.
- ممکن است یک بعد را بتوان به عنوان ویژگی های یک بعد دیگر یا به عنوان یک بعد مستقل در نظر گرفت
 - ❖ این مساله به میزان استفاده از آن بعد بستگی دارد
 - ❖ مثلاً نوع سپرده را می توان یک بعد جداگانه در نظر گرفت یا به عنوان یک ویژگی در بعد سپرده

مثال:

بعد نوع تسهیلات و بعد تسهیلات

نکته: توی دیبی 2 تکرار داریم

نکته: نوع تسهیلات رو به عنوان یک dimension به فکت اضافه کردیم حالا تبعات این چیه

از لحاظ حافظه؟ وقتی یک ستون اضافه میکنیم به تعداد رکوردهای فکت داریم به جدول حافظه

اضافه میکنیم --> وجود بعد جدید هزینه داره

نکته: ما الان اینجا بعد نوع تسهیلات داریم الان سوال اینه که چرا این 4 تا فیلدی که توی بعد نوع

تسهیلات است باید توی بعد تسهیلات هم باشه؟ برای خود تسهیلات قرار شد هر ان چیزی که بهش

ربط داره توی یک جدول باشه --> خیلی از نیازمندی ها از همون بعد تسهیلات حل میشه

* Dimension: هر یک بعد، به کار بردن تسهیلات دارد. مثال:

بعد تسهیلات

شماره تسهیلات

تاریخ دریافت

مبلغ وام

تعداد اقساط

کد نوع وام

شرح نوع وام

کد زیرنوع وام

شرح زیرنوع وام

→ اگر در fact ای که داریم، از نوع تسهیلات استغادهای زیادی
شده باشه، می توان یک بعد جدا برای آن در نظر گرفت.

(تسهیلات به کار برد، با این حجم داده ها بالا می رود، ولی باز هم به عنوان یک dim جدای متصل به fact آنرا در نظر می گیریم.

نکته:

نکته:

توی فکت های اسنپ شاتی یک بک اپ فول توی اون اسنپ شات زمانی از موجود اصلیمون می گیریم و ذخیره می کردیم مثلا اگر داریم روزانه ذخیره می کنیم معمولا سطح های بالاتر اسنپ شات هم ذخیره می کنیم چون حجم خاصی نمی گیرن و بعضا کار ما هم راحت تر می کنن مثلا اگر روزانه گرفتیم --> ماهانه و سالانه هم می گیریم --> فکت روزانه و فکت ماهانه و فکت سالانه داریم الان

کلید Dimension

❖ Natural Key

- ❖ در صورتی یک کلید طبیعی ساده برای موجودیت مربوطه در بیزینس وجود دارد می توان از آن برای کلید بعد استفاده کرد.
- ❖ مثلاً برای بعد شعبه، می توان کد شعبه را در نظر گرفت.
- ❖ بهتر است کلید را از نوع عددی ایجاد نماییم.

❖ Surrogate Key

- ❖ معمولاً بهتر است برای هر بعد یک Surrogate Key در نظر گرفت.

Natural Key: کلیدی است که توی بیزینس معنا داره مثل برای آموزش دانشگاه، شماره

دانشجویی برایشون میشه کلید **Natural** اون بیزینس برای دانشجو

داخل بیزینس یک مفهومی داریم به اسم شماره دانشجویی و اگر واسه دانشگاه اومدیم بعد دانشجو طراحی کردیم بعد می‌تونیم همون شماره دانشجویی رو استفاده بکنیم برای بعد دانشجو به عنوان کلیدش

بهتر است کلید را از نوع عددی ایجاد بکنیم چرا عددی بهتره؟ چون توی جویین ها سرعت رو خیلی افزایش میده

Surrogate Key میشه چی؟ مثلاً یک کلید داریم برای جدول دایمنشن دانشجو که میشه همون شماره دانشجویی و به دلایلی مثلاً می‌خوایم براش **scd** تعریف بکنیم یا تایپ فیلدمون عددی نیست یا طول این کلید خیلی طولانی است و... و نمی‌خوایم اینطوری باهاش کار بکنیم چون باعث میشه پرفرمنس تحت تاثیر قرار بگیره توی اینجور مواقع توی دایمنشن برای راحتی کار بهتره یک کلید **Surrogate** بذاریم
اینم عددی است

ممکنه هر دو جفت **Surrogate Key** و **Natural Key** رو داشته باشیم مثلاً **Natural Key** دارد ولی عددی نیست پس کنارش یک **Surrogate Key** هم می‌ذاریم یا اینکه خیلی طولانی است یا اینکه می‌خوایم **scd** بزنیم یا... <-- توی این حالت **Surrogate Key** رو هم میاریم

توی فکت چی میخوره؟ یک جدولی داریم که هم **Surrogate Key** داره و هم **Natural Key** و الان اینجا **Surrogate Key** میشه کلید اصلیمون توی فکت پس توی فکت **Surrogate Key** قرار خواهد گرفت و این میشه کلیدمون و کاری به **Natural Key** نداریم پس اگر **Surrogate Key** توی جدولی داشتیم این حتماً توی فکت ثبت میشه

توی عمل اگر Surrogate Key داشتیم برای دایمنشن و Natural Key هم داریم <--
Surrogate Key حتما داخل فکت ثبت میشه ممکنه به دلایلی ما Natural Key هم داخل فکت
بزنیم <-- یک بحث، بحث نگهداری دیتابیس هست که جلوتر می گه و یک بحث هم بحث بیزینسی
است که برای اون کاربر Natural Key خیلی واضح تر است مثل شماره مشتری یا شماره
دانشجویی یا ... براش معنادارتر است و Surrogate Key براش خیلی واضح نیست مثل کد 5

تا الان اینو داشته باشیم که Surrogate Key رو حتما توی فکت می زنیم و یه جاهایی خیلی خیلی
خاص شاید ما Natural Key هم داخل فکت بزنیم

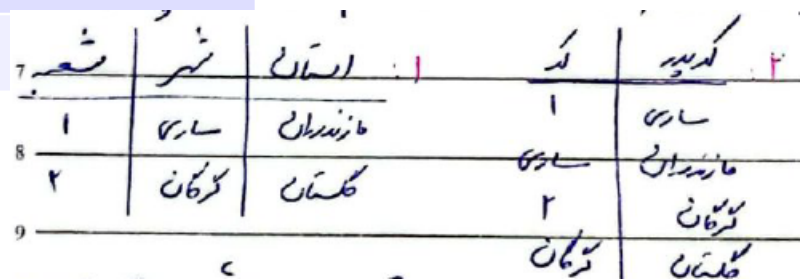
سلسله مراتب در Dimension

- لزومی ندارد که تمام ویژگی های در سلسله مراتب باشند
- می توان چندین سلسله مراتب در یک بعد متصور شد.
 - شعبه، شهر، استان
 - شهر، استان
 - شعبه، استان

سلسله مراتب در Dimension

- سلسله مراتب نوع ۱ : برای هر سطح یک ویژگی در بعد داریم:
 - شعبه، شهر، استان
- سلسله مراتب نوع ۲: دو ویژگی داریم که رابطه پدر فرزندی دارند.
 - سرفصل حسابداری

کد سرفصل	کد سرفصل لایه بالاتر
10000	1000
1000	100
100	10
10	1
20000	2000
300	30



کد	استان	شهر	شعبه
۱	مازندران	ساری	۱
۲	گلستان	گزنجان	۲
۳	گلستان	گزنجان	۳

ما می‌تونیم سلسله مراتب رو توی دایمنشن به دو صورت مرسوم انجام بدیم:

طراحی 1: ما به ازای سطوح مختلف سلسله مراتب برای هر سطح یک ویژگی داخل دایمنشن داشته باشیم مثل شعبه، شهر و استان و کشور --> برای هر کدوم از اینا باید یه دونه ستون توی جدول داشته باشیم پس برای اینا 4 تا ستون مستقل داریم --> نرمال نیست اگر بخوایم نرمالش بکنیم چجوری این کارو میکنیم؟؟؟

برای مثال توی اسلاید: که شعبه و شهر و استان است --> توی سورس سه تا جدول شعبه و شهر و استان داریم که توی جدول شعبه: کد شعبه و نام شعبه و کد شهر خورده و توی جدول شهر: کد شهر و نام شهر و کد استان خورده و برای جدول استان: کد استان و نام استان است --> این سه تا جدول میشه جدول سورس ما و دایمنشن شعبه از جوین این سه تا پر میشه و کجا؟ توی ETL پس ETL این سه تا رو جوین میکنه و اینسرت میکنه توی دایمنشن شعبه

طراحی 2: از جنس پدر- فرزند است ینی با دوتا ستون سلسله مراتب های مختلف رو ایجاد میکنیم مثلا برای یک کد سرفصلی پدرشو جلوش می‌نویسیم ینی جلوی هر کد سرفصل پدرشو نوشتیم که این می‌تونه یک سلسله مراتبی برامون ایجاد می‌کنیم توی این مثال الان 5 تا سطح توی همین دیتا وجود داره!!
مثلا برای شعبه و شهر و استان --> کد شعبه پدرش میشه کد شهر

نکته: فرق این دوتا طراحی: توی این طراحی 1 ما سه تا ستون داریم برای این مثال --> توی جدول دایمنشن برای هر سطح از سلسله مراتب ها اینجا یک ستون جداگانه گذاشتیم اما توی طراحی 2 می‌تونیم دو ستون داشته باشیم مثال:

توی لایه cube برای سلسله مراتب بستری فراهم است --> اینجا وقتی می خوایم یک سلسله مراتب ایجاد بکنیم برای دایمنشن، هر دو نوع رو cube ساپورت می کنه

نکته: یک فرزند نباید دوتا پدر داشته باشه ولی اگه داشت نباید بیاریمش توی انبار داده چون کار رو خراب میکنه و باید صبر کنیم تا اون فردی که سورس رو بهمون داده اینو درست بکنه و بعد روش بتونیم کار بکنیم ینی توی فرایند ETL یه جا می تونیم براش در نظر بگیریم و بعد به اون فرد بگیم که اینا درست نیستن برو سورس رو درست کن و دوباره بهمون بده

* در Cube می توان برای یک dimension
سلسله مراتب ها را تعریف کرد.

نگهداری تغییرات یک ویژگی در Dimension

Slowly Changing Dimension (SCD)

تغییرات توی انبار داده اروم است ینی ما هر دقیقه توش تغییر نداریم

➤ ممکن است لازم باشد تغییرات یک یا چند ویژگی را در یک بعد نگهداری نماییم.

➤ مثلاً در بعد مشتری، می خواهیم در صورت تغییر شماره تلفن همراه شماره جدید را (نیز) داشته باشیم.

➤ برای این کار روشهای مختلفی وجود دارد

➤ بروزرسانی

➤ ایجاد رکورد جدید

➤ ...

SCD نوع ۱

➤ در این روش در صورت مشاهده تغییر در جداول سورس، مقدار قبلی موجود در بعد را با مقدار جدید جایگزین می نماییم.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

Surrogate Key

Natural Key



Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	IL

نوع 0: ینی تغییرات اصلا برامون مهم نیست مثلا اینجا اگر شغل مشتری عوض بشه نمی خواد بهش دست بزنه

نوع 1: از جنس ایدیت است ینی شغل مشتری عوض شده و شغل جدید رو می خوایم داشته باشیم

اینجا Surrogate Key و Natural Key هم می بینیم

SCD نوع ۲

- در این روش در صورت مشاهده تغییر در جداول سورس، رکورد جدیدی ایجاد می نماییم.
- در این روش وجود **Surrogate Key** در بعد الزامی است.
- سه فیلد برای تاریخ شروع، تاریخ پایان و مشخص کردن فعال بودن رکورد جاری، در بعد در نظر گرفته می شود.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

مشاهده تغییر در
Supplier_State

Surrogate
Key

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004	0
124	ABC	Acme Supply Co	IL	22-Dec-2004		1

نوع 2: این Scd پیچیدگی بیشتری دارد
ما سعی می کنیم اینجا کل هیستوری رو داشته باشیم
ولی توی نوع 1 ما فقط آخرین وضعیتش رو داشتیم

توی نوع 2 میگه به ازای هر رکوردی که مقدار ستون مورد نظرش تغییر کرده باید یک رکورد داشته باشیم

توی این حالت سه تا ستون دیگه هم باید داشته باشیم : تاریخ شروع - تاریخ پایان - فلگ
تاریخ شروع و تاریخ پایان برای CA: میگه از چه روزی تا چه روزی توی این حالت بوده
برای رکوردی که در حال حاضر معتبر است ما تاریخ شروع داریم ولی تاریخ پایان نال است چون
هنوز معتبر و ولید است
فلگ: اگر رکورد جاری فعاله مقدار این فلگ 1 است ولی اگر داره به هیستوری و قبل اشاره میکنه
مقدار ستونش 0 است

نکته: وقتی دوتا رکورد زدیم Natural Key خاصیت یکتایشو از دست میده --> توی جداولی که
از نوع scd نوع 2 هستن ما دیگه Natural Key برامون معنایی نداره
پس اگر scd نوع 2 داشته باشیم قطعاً باید Surrogate Key هم داشته باشیم و توی فکت
Surrogate Key می خوره

برای Surrogate Key دیتابیس اینجوری کار میکنه میگه مقدار max چنده بعد بیا اونو به اضافه
یک بکن و این میشه Surrogate Key ما پس لزوماً عددهای Surrogate Key پشت سر هم
نیستن

نکته: میخوایم هم اگر شغل مشتری عوض شد track اش بکنیم و هم اگر تلفنش عوض شد: مثلا اینجا می خوایم دوتا فیلد رو track بکنیم <-- یکی از روش های مرسومیه که استفاده میشه استفاده کردن از یک فیلد چهارمی است

فیلد چهارم به ما کمک میکنه که به ما بگه چیا تغییر کرده توی این رکورد <-- و اگر دوتا فیلد بود ما دوتا بیت پر بکنیم بیت اول برای یک فیلده و بیت دوم برای یک فیلد دیگه و اگر سه تا فیلد تغییر می کرد اون ستون اخرم که داره track میکنه چی تغییر کرده سه تا بیت داره در این حالت (مثلا سه بیت ینی 8 حالت ولی حالت 000 معنایی نداره چون اگر تغییری نکنه رکورد نمی زنیم ینی عملا 7 تا حالت داریم که این 7 حالت را با 3 بیت داره کنترل می کنه مثلا فیلد اول اگر تغییر کرده بیتش میشه یک و اگر تغییر نکرده میشه صفر و برای فیلدهای دیگه هم همینطور)

نکته: بحث scd حتما روی دایمنشن است

نکته: Surrogate Key حتما داخل فکت میاد <-- داخل فکت چه بلایی سر Surrogate Key میاد؟ عکس پایین رو ببین <-- در 402/1/1 کلید شده 124 و 124 توی فکت می خوره و قبلی هاش 123 بود ولی کاری به قبلی ها نداریم چون اگر بخوایم 123 ها رو اپدیت بکنیم خیلی کارمون سخت میشه و این جنس تغییرات توی فکت هزینه بالایی داره

می خوایم فکت و بعد رو جوین بکنیم روی Surrogate Key

ردیف	Fact	Surrogate Key
21	123	401, 1, 1
22	124, ABC, CA, 401, 1, 1, 401, 12, 29, 0	401, 1, 1, 29
23	124, ABC, IL, 402, 1, 1, 402, 1, 28, 0	402, 1, 1, 1
24	125, ABC, X, 402, 1, 29, null, 1	402, 1, 1, 29
25	125	402, 1, 1, 29

نکته: داخل فکت که حتما Surrogate Key رو داریم ایا Natural Key هم باید بزنینم یا نه؟
می دونیم تبعات یک فیلد اضافه کردن داخل فکت سنگین است پس باید اهمیت داشته باشه که بزنینم
وگرنه هزینه زیادی برامون داره --> جدا از اینکه که یکم از جهت بیزینسی برای کاربر قابل فهم تر
میشه رکوردها از یک جنبه ای برای نگهداری یا maintenance انبار داده بد نیست و توصیه
میشه داخل فکت Natural Key رو هم بزنینم (اینو توی شرایط Scd داریم می گیم نه توی همه
شرایط)

به چه دردی می خوره؟ دایمنشن هایی که scd نوع 2 فقط براشون داره ست میشه در دسرهایی
براشون داریم یکی از این در دسرها اینه که دایمنشن از جمله خطای انسانی پاک کنیم اون وقت اون
هیستوری ممکنه به راحتی برنگرده در این حالت دیگه فکت می ره روی هوا --> راه حل: از
طریق Natural Key داخل فکت از این طریق می تونه هیستوری برگرده ولی کار پیچیده ای است
ولی می تونه برگرده

فرض میکنیم این اتفاق افتاد حالا دایمنشن رو چجوری بسازیم اگر داخل فکت Natural Key وجود
داشته باشه؟ مهمترین کار اینه که دایمنشن کلیدهاش برگرده که این کلیدها رو داخل فکت داریم -->
نفهمیدم چجوری شد؟؟؟؟ --> اگر Natural Key داخل فکت نبود فکت خراب میشد و پر کردن
فکت کار راحتی نیست

SCD نوع 3

- در این روش فیلدهای اضافی برای تغییرات احتمالی در مقادیر فیلد مربوطه در نظر گرفته می شود. مثلاً برای شماره تلفن ۳ ستون مختلف در نظر گرفته می شود. در صورت وجود شماره جدید در ستون بعدی ثبت می شود.
- تاریخچه تغییرات را به صورت محدود ذخیره می کند.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA



Supplier_Key	Supplier_Code	Supplier_Name	Original_Supplier_State	Effective_Date	Current_Supplier_State
123	ABC	Acme Supply Co	CA	22-Dec-2004	IL

نوع 3:

اگر جایی داخل دایمنشن بدونیم این ستونی که می خواهیم هیستوریش رو نگه داریم یه بار یا دوبار یا سه بار بیشتر در کل تاریخچه اش عوض نمیشه --> برای این بیا چندتا فیلد بذار و به تعداد تغییراتی که توی هیستوری ممکنه براش اتفاق بیوفته اینو در نظر بگیر

مثلا میدونیم که مقدار **Supplier_State** کلا دو مقدار بیشتر در کل تاریخچه نمی گیره اولی رو اسمشو می داریم **Original** و دومی رو می داریم **Current** --> تا زمانی که اولی بوده توی **Original** این رو می داریم و اگر یه زمانی تغییر کرد مقدار قبلی رو توی **Original** میذاریم و مقدار جدید رو توی **Current** نگه میداریم

سامانه مدیریت پروژه

- سامانه ای جهت کنترل و مدیریت پروژه‌های یک سازمان پروژه محور
- ثبت اطلاعات اولیه هر پروژه
- مانند نام پروژه، ناظر پروژه، تاریخ شروع و....
- ثبت مبالغ ورودی پروژه‌ها
- ثبت مبالغ هزینه شده برای هر پروژه با مشخص بودن نوع هزینه

مثال: یک سامانه برای مدیریت پروژه و پروژه های یک سازمان داره ثبت میشه توی اطلاعات سورس و اطلاعاتی برای هر پروژه نگه می داریم مثل این پروژه اسمش چی بوده، ناظر پروژه کیه و تاریخ پروژه کیه و تاریخ پایان کیه و

مدل طراحی شده برای انبارداده

وقتی پروژه یکسری ورودی هایی داره ینی
داره تامین میشه --> ینی مبالغ داره بهش
داده میشه و این مبلغ رو از کجا گرفته مهم
میشه اینجا

بعد محل
تامین اعتبار

بعد تاریخ

فکت تراکنشی
پروژه

بعد پروژه

این بعد به ما کمک میکنه که هزینه هامون
رو کنترل بکنیم
مثلا یکی از هزینه ها بحث پرسنل باشه مثلا
خدماتی ها چیا بودن یا مدیر پروژه کیا بودن
و...

بعد سرفصل

بعد پروژه

نام فارسی فیلد	نام لاتین فیلد
کد پروژه	Project_Code
عنوان	Project_Title
مرکز	Center
مدیر پروژه	Project_Manager
نوع پروژه	Project_Type
تاریخ شروع	Start_Date
طول پروژه-روز	Day Duration
طول پروژه-ماه	Month Duration
طول پروژه-سال	Year Duration
جریمه هر ماه	Penalty Each Month
مبلغ پروژه	Total Project Amount
طبقه بندی	Classification

نکته: ما محدودیتی روی اینکه بعد sum بزنیم نداریم

نکته: طبیعت فکت aggregate زدن و لی خب یک جدول داریم که مبلغ تمام وام ها توش خورده و می خوایم بدونیم که جمع مبلغ کل وام ها چقدره پس روی همین دایمنشن sum می تونیم بزنیم یا اینکه می تونیم از فکت ACC استفاده کنیم --> فکت های ACC فکت هایی هستن که اینجور مواردی که انگار ویژگی دایمنشن هستن رو توی فکت ACC می داریم --> حالا از اینکه از این بعد بگیریم یا از اون فکت بگیریم خیلی فرقی نمی کنه

طبقه بندی: مثلا نوع محرمانگی رو می خواد مشخص بکنه مثلا محرمانه است یا محرمانه نیست یا.. --> یک برجسب برای طبقه بندی پروژه ها

جریمه هر ماه: به ازای هر ماه تاخیر اینقدر از ما کسر می کنه پس یک مبلغ ثابت است

نکته:

سوال: توی دیتاورهاوس میتونستیم روی مبلغ پروژه جریمه aggregate انجام بدیم برای دایمنشن ولی توی cube نمی تونیم حالا اگر توی cube بخوایم این کارو انجام بدیم باید چی کار کنیم؟ و چرا اصلا نمیشه این کار رو کرد؟ چون توی cube اون حالتی که مقدار هزینه جریمه پروژه ما توی فکت یا دایمنشن است فرقی در این است که ما ابزارشو نداریم که بخوایم خودمون توی cube جمعش رو حساب بکنیم برای یک دایمنشن --> فلسفه aggregate برای مژرهای داخل فکت است و بقیه فیلدها رو به عنوان ویژگی میخوایم ببینیم و با این هدف که بخوایم جمع این ویژگی ها رو ببینیم ربطی به cube نداره ولی به عنوان ویژگی می تونیم ببینیم توی cube

بعد تاریخ

نام فارسی فیلد	نام لاتین فیلد
کلید تاریخ	Date_Key
کلید تاریخ شمسی	Shamsi_Date_Key
سال میلادی	Year
سال شمسی	Shamsi_Year
فصل میلادی	Quarter
فصل شمسی	Shamsi_Quarter
ماه میلادی	Month
ماه شمسی	Shamsi_Month
روز هفته میلادی	Weak_Day
روز هفته شمسی	Shamsi_Weak_Day

بعد محل تامین اعتبار

نام فارسی فیلد	نام لاتین فیلد
کد محل اعتبار	Resource_Code
نام محل اعتبار	Resource_Name

بعد سرفصل

نام فارسی فیلد	نام لاتین فیلد
سرفصل سطح ۱	GLlevel1
سرفصل سطح ۲	GLlevel2
شرح سرفصل سطح ۱	GLlevel1_Desc
شرح سرفصل سطح ۲	GLlevel2_Desc

فکت پروژه (از نوع تراکنشی)

نام فارسی فیلد	نام لاتین فیلد
کد پروژه	Project_Code
کلید تاریخ	Date_Key
کد محل تامین اعتبار	Resource_Code
سرفصل سطح ۲	GLlevel2
هزینه کرد	Turn_Over_Bed
دریافتی	Turn_Over_Bes
پیشرفت برنامه ای	Progress_Programmatic
پیشرفت واقعی	Progress_Supervisor
مبلغ جریمه	Penalty

هزینه کرد: ینی چه هزینه ای ما پرداخت کردیم

دریافتی: ینی چه مبلغی به عنوان ورودی گرفتی

فرق مبلغ جریمه اینجا با دایمنشن: اونی که توی دایمنشن بود ثابت بود و اون نشون میداد که اگر هر ماه پروژه رو دیر تحویل بدیم اینقدر از مون کم میشه ولی اینجا داره میگه اینقدر جریمه شدی الان

پر شدن این فکت چند حالت مختلف داره: ممکنه فقط رکوردی بزنینم که هزینه کرد داره ولی دریافتی نداره در این حالت دریافتیش میشه صفر (ولی به جای صفر بهتره نال بذاریم این دقیقا میشه همون موضوعی که گفتیم مژرها می تونه نال بشه چرا نال؟ مثلا فرض کن بحث میانگین گیری باشه در این حالت مقدار صفر کار رو خراب میکنه پس بهتره نال باشه در کل) و..

نکته: برای جدول های فکت primary key معنایی نداره

مثلا میخوایم فکت اسنپ شات روزانه هم داشته باشیم برای این مثال:
دایمنشن ها: تاریخ - کد پروژه

فکت روزانه رو براساس چه موجودیت هایی می سازیم؟ ریزدانگیش رو چی تعیین میکنه؟ از جنبه زمان میشه روز - کد پروژه هم هست (اینجا رو همون طوری که برای پروژه خوندی بخون مثلا در هر روز برای هر پروژه می خوایم مثلا یه کاری بکنیم مثلا) --> اگر برای یکسال بخوایم دیتا بزنین تا الان فکت شده 365 هزار رکورد الان می خوایم جلوتر هم بریم یا نه؟ مثلا اگر بخوایم برای هر سر فصل سطح 2 بگیم اینو که این بستگی به نیازمون داره که ببینیم اصلا اینو میخوایم یا نه پس اگر بخوایم بگیم برای هر سر فصل سطح 2 در این حالت باید به تعداد رکوردهای داخل جدول سرفصل اونو ضربدر 365 هزارتا بکنیم و اگر فرض کنیم تعداد رکوردهای داخل سرفصل 100 تا باشه در این حالت میشه 100 در 365 هزارتا

و کد محل تامین اعتبار هم به همین صورت است که ایا واقعا لزومی داره که اینو توی فکت بیاریم؟
مثلا اگر 20 تا رکورد محل تامین اعتبار داریم در این حالت داریم فکت رو ضربدر 20 می کنیم

پس وقتی که میخوایم یک موجودیت دیگری رو درگیر بکنیم اگر اون موجودیت توی دل اون یکی نباشه و به عنوان ویژگی اون دایمنشن قبلی وجود نداشته باشه پس در این حالت به تعداد رکوردهای این باید توی فکت هم مقدار داشته باشیم: حرف پایین
بازم نفهمیدم چرا ؟؟؟

Dimension هایی که نیاز داریم حتما باید لزوماً برسی بشود.
الریک موجودیت در دل موجودیت های دیگر نباشه ، به تعداد رکوردهای اون ، در رکوردها گنیت
ضرر: سی شود و اگر لزوماً به وجود اون موجودیت نباشه ، چون هم نیاز به گنایت افعال شه ،
آنها اضافه نمی کنیم.

مژرها: مبلغ جریمه تا اون روز - مبلغ جریمه در اون روز ==> این دوتا، دوتا دید متفاوت هستن که هر دوتاش معنادار است ولی نکته اینه که هر کدوم رو که گذاشتیم حتما کاربر باید مطلع بشه که کدوم است این - میزان پیشرفت واقعی - میزان پیشرفت برنامه - هزینه کرد - دریافتی - میزان تاخیر پروژه از زمانی که باید تمام بشه (هر روز که انجام نشه یکی بهش اضافه میشه تا قبل از اتمامش صفره ولی بعد از اتمامش هی می تونه بهش اضافه بشه)

مثال

سپرده‌های یک بانک در نظر بگیرید که به صورت روزانه روی آنها تراکنش اعمال می‌شود. فرض کنید که جدول تراکنش سپرده به صورت زیر باشد:

کلید سپرده	لحظه انجام تراکنش	گردش بدهکار	گردش بستانکار	Id تراکنش در روز
۱	10/20/2016 2:19:27	0	0	1
۱	10/20/2016 2:19:27	0	20	2
۱	10/21/2016 2:19:27	15	0	1
۲	10/25/2016 2:19:29	0	200	1
۲	10/25/2016 2:19:29	0	100	2
۳	10/26/2016 2:19:29	0	20	1
۱	10/26/2016 1:19:29	5	0	1

سپرده‌ها قبل از ورود به جدول تراکنش، هیچ مانده ای ندارند، (یعنی اولین تراکنش یک سپرده، مانده آن را تعیین می کند. مثلاً در مثال بالا، اولین مانده سپرده ۱ برابر ۰ و اولین مانده سپرده ۲ برابر ۲۰۰ می باشد).

نکته: توی فکت ها از نوع تراکنشی و اسنپ شات اضافه شدن فیلد تبعات زیادی داره ولی توی فکت لس و ACC و دایمنشن ها تبعات زیادی دیگه نداره

گردش بدهکار: برداشت از حساب
گردش بستانکار: واریز به حساب

id تراکنش در روز: ممکنه در یک لحظه از زمان چند تراکنش رو با هم بیاین ثبت بکنن --> برای یک سپرده در یک لحظه که ساعتشون یکیه می خوان چندتا تراکنش رو ثبت بکنن --> در واقع این ترتیب رو مشخص میکنه (خودشون این ترتیب رو از اول مشخص کردن و به ما میدن ینی اینا توی جدول سورت است) --> جدول سورت که چجوری شکل گرفته برای ما مهم نیست و چیزی که برامون مهمه این که این جدول رو بفهمیم

نکته: ترتیب تراکنش ها اولین چیزش زمان است ینی براساس زمان مشخص شده ترتیب

چه اهمیتی داره کدومش یکه و کدومش 2: مثلا توی حسابه 10 واحد پول داریم و یک تراکنش داریم 100 واحد واریز و یک تراکنش دیگه داریم 50 واحد برداشت --> قطعا اول واریز انجام میشه و بعد برداشت انجام میشه (یه جورایی ترتیب برامون مهمه اینجا برای تراکنش های توی یک لحظه)

یک فکت روزانه با ابعاد سپرده و تاریخ ایجاد کنید که در آن مقدار حداقل، حداکثر، میانگین (بر حسب تعداد تراکنش روز)، آخرین مانده و تعداد روزهای بدون عملکرد هر سپرده را در هر روز ثبت شده باشد. مثالی از خروجی مورد نظر که لازم است در یک جدول ذخیره شده باشد در زیر آمده است.

کلید سپرده	تاریخ	حداقل مانده	حداکثر مانده	میانگین مانده	آخرین مانده	تعداد روزهای بدون عملکرد
1	10/20/2016	0	20	10	۲۰	0
۱	10/2۱/2016	۵	۲۰	۱۲.۵	۵	0
۱	10/22/2016	۵	5	5	۵	1
۱	10/23/2016	۵	5	5	۵	2
۱	10/24/2016	۵	5	5	۵	3
۱	10/25/2016	۵	5	5	۵	4
۲	10/25/2016	۲۰۰	۳۰۰	۲۵۰	۳۰۰	0
1	10/26/2016	0	5	2.5	0	0
۲	10/26/2016	300	۳۰۰	300	۳۰۰	1
3	10/26/2016	20	20	20	20	0

مژر

حداقل مانده: ممکنه یک سپرده در یک روز تراکنشی نداشته باشه در این حالت حداقل مانده اش همون مقداری میشه که داشته ینی مانده روز قبلش میشه در واقع

تعداد روزهای بدون عملکرد: ینی چند روزه تا حالا حساب کار نکرده --> برای حساب کردنش: آخرین روزی که تراکنش بوده کی بود و امروز چه روزیه --> اختلاف اینا میشه تعداد روزهای بدون عملکرد

برای میانگین مانده از روز قبل هم استفاده میکنیم ینی مانده اون روز با روز قبل می گیریم و میانگین رو حساب میکنیم ینی یه جوری همش به روز قبل هم وابسته میشه کارامون اره؟؟

این کد و این پروسیجر برای زمانی استفاده میشه که ما در حالت روتین باشیم ینی تا یه جایی از فکت ما به روزه و می
خوایم بریم جلو ولی اگر اولین بار است که فکت و انبار داده رو به روز می کنیم مقدار ما یکمی متفاوت است --> کدها
خیلی شبیه این است ولی اونو با احتیاط بیشتر انجام میدیم **پاسخ**

```
procedure Ins_factDeposit(fromdate date, todate date) is  
    currDate date;
```

```
begin
```

```
    currDate := fromdate;
```

```
    while currDate <= toDate loop
```

```
        truncate table tmp_all_trn;
```

```
        truncate table tmp_dpst_sum_trnover;
```

```
        truncate table tmp_all_remain;
```

```
        truncate table tmp_last_remain;
```

```
        truncate table tmp_last_remain2;
```

parallel: ینی داره به DBMS میگو به صورت
parallel برو محاسبات رو انجام بده و این خیلی
سریعتر می تونه بره کار بکنه
نکته: این مثال روی اوراکل است

```
insert into tmp_last_remain
```

```
    select /*+parallel(8)*/
```

```
        f.dpst_num, f.effdate, f.bal, f.passivedays
```

```
        from factdeposit f
```

```
            and f.effdate = currDate - 1;
```

```
commit;
```

اینجاش چیش غلطه؟



منطقی که توی این پروسیجر داریم اینه که: فرضش اینه که ما از یک fromdata تا یک todate قراره فکت رو به روز بکنیم

این پروسیجر دوتا ورودی داره الان و نمی خوایم برای این دوتا هر شب دستی مقدار بدیم --> پس باید یک پروسیجر دیگه داشته باشیم که to , from رو می ره کش می کنه و پاس میده به این --> مثلاً می تونیم بریم دیتای سورس رو نگاه بکنیم تا کی به روز شده و دیتای فکت تا کی به روز شده و این یک اختلافی داره مثلاً سورس سه روز جلوتر از فکت ما است در این حالت to , from به دست میاد

پس این to , from میگه برو برای این بازه زمانی فکتت رو به روز کن

وقتی میگیریم فکت روزانه ما این تعداد روز عقبه --> یک to , from گرفته و برای این که کار پیچیدگی زیادی نداشته باشه داخل یک حلقه هر بار کار ما انجام میشه و می ره جلو --> بار ما رو به شدت کم میکنه

توی حلقه: که میگه از fromdate شروع کن و تا todate برو جلو truncate یعنی رکورد های جدول رو بدون ایجاد لاگ و امکان بازیابی حذف کن و همینطور سریعتر از دیلیت است --> اگر جداول ایندکس داشت برای سریعتر شدن اول ایندکس را drop می کنیم و بعد truncate میکنیم

و وقتی که داریم از truncate استفاده میکنیم اون امکان ریکآوری رو از بین بردیم و اینجا برامون مهم نیست که ریکآوری نداشته باشیم
توی دیلیت ما ریکآوری داریم

رکوردهای روز قبل برامون مهمه : factdeposit همین جدولی است که داریم پرش میکنیم
:f.effdate = currDate - 1

currDate الان کیه ؟ از همون روزی که باید شروع بکنیم و این میگه یه روز قبلش هم در نظر بگیر
ینی در واقع داره اطلاعات فکت روز قبل رو برمیداره

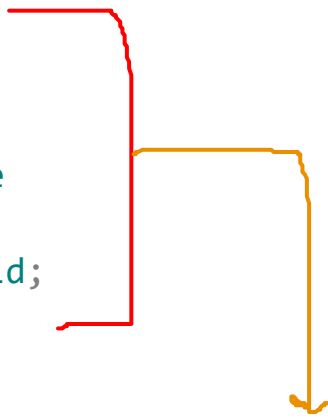
و در اخر می ریزه توی tmp_last_remain یعنی روز قبل از فکت رو توی این tmp ریخته

```

insert into tmp_all_trn
  select /*+parallel(8)*/
    t.dpst_num, t.trns_time, t.dpst_trnover_bes, t.dpst_trnover_bed, t.trun_id
  from SA.dpst_trn t
  where t.trns_time >= currDate
        and t.trns_time < currDate + 1
commit;

insert into tmp_last_remain2
  select /*+parallel(8)*/
    t.dpst_num, t.trns_time effdate, sum(t2.dpst_trnover_bes - t2.dpst_trnover_bed)
dpst_bal
  from tmp_all_trn t
 inner join tmp_all_trn t2
    on t.dpst_num = t2.dpst_num
    and t.trns_time >= t2.trns_time
    and t.trun_id >= t2.trun_id
 group by t.dpst_num, t.trns_time, t.trun_id;
commit;

```



اینجا داره tmp_all_trn رو داره با خودش جوین میکنه: این داره هر رکورد رو با تمام رکورد های قبل تر از خودش جوین می کنه چرا این کارو میکنیم؟ برای حساب کردن مانده

tmp_all_trn: این tmp اطلاعات سورس امروز رو می گیره ینی سورس رو ریخت توی این tmp

tmp_all_remain: برای اینکه متوجه بشه سپرده هایی که توی امروز تراکنش داشتن چیان

-- mande dirooz sepordehaee ke tarakonesh dashteand ra dar tmp_all_remain mirizad

insert into tmp_all_remain

select /*+parallel(8)*/

به جای min می تونستیم max یا avg .. هم بذاریم چون

1 t.dpst_num, currdte, min(t.dpst_bal) dpst_bal

فرقی نمی کرد

from tmp_last_remain t

توی tmp_last_remain همه سپرده

currdte inner join tmp_all_trn t2

های دیروز هستن ولی فقط الان اونایی

متغیر است on t.dpst_num = t2.dpst_num

رو اینجا آورده که فقط امروز تراکنش

group by t.dpst_num;

داشتن

commit;

-- Mandehaye jadid ra dar tmp_all_remain mirizad, bad az anjame in marhale agar sepordehee n tarakonesh dashte bashad, n+1 mande barayash darim

insert into tmp_all_remain

select /*+parallel(8)*/

nvl(t.dpst_num, t2.dpst_num),

nvl(t2.effdate, t.effdate + 1),

2 nvl(t.dpst_bal, 0) + nvl(t2.dpst_bal, 0) dpst_bal

from tmp_last_remain t

tmp_last_remain: مانده دیروز فکت داخلشه

full outer join tmp_last_remain2 t2

on t.dpst_num = t2.dpst_num;

commit;

tmp_all_remain: این جدول برای هر سپرده یک مقدار داره

توی این داره مشخص میکنه که ایا سپرده تراکنشی داشته یا نداشته --> اگر سپرده تراکنش داشته با نداشته باشه برامون فرق داره

و این داره سپرده هایی ک تراکنش داشتن رو برای خودش دربیاره

که توی بعدی مانده جدید رو توی tmp_all_remain بریزه و بعد از انجام این مرحله اگر سپرده ها تراکنشی داشته باشن $n+1$ مانده براشون ایجاد میشه

توی 1 یک مانده برای اونایی که تراکنش داشتن حساب کرد --> حالا توی 2 می خواد n تا مانده دیگه هم بیاره

این nvl مثل isnull است

مثلا برای $\text{nvl}(t.\text{dpst_num}, t2.\text{dpst_num})$: اگر اولی نال بود دومی رو می داریم به جاش ولی اگر هر دوتا مقدار داشتن همون مقدار اولی در نظر گرفته میشه

چرا full outer join کرده؟ شاید سپرده ای امروز بوده که دیروز وجود نداشته - یا شاید امروز تراکنش نداشته

```
truncate table tmp_min_max_avg;
insert into tmp_min_max_avg
select /*+parallel(8)*/
t.dpst_num, min(t.dpst_bal), max(t.dpst_bal), avg(t.dpst_bal)
from tmp_all_remain t
group by t.dpst_num;
commit;
```

```
insert into tmp_dpst_sum_trnover
select /*+parallel(8)*/
t.dpst_num,
SUM(t.dpst_trnover_bes),
SUM(t.dpst_trnover_bed)
from SA.dpst_trn t
where t.trns_time >= currDate
and t.trns_time < currDate + 1
group by t.dpst_num;
commit;
```

این جمع گردش بدهکار و بستانکار
رو داره می زنه که سوال ازمون
نخواسته بود


```

insert into factdeposit
select /*+parallel(8)*/
  nvl(tmp.dpst_num,tr.dpst_num)dpst_num,
  currDate effdate,
  nvl(tmp.dpst_bal, 0) + nvl(tr.trnoverbes, 0) - nvl(tr.trnoverbed, 0) bal,
  nvl(tr.trnoverbes, 0) trnoverbes,
  nvl(tr.trnoverbed, 0)trnoverbed,
  nvl(tmp_bal.min_dpst_bal, 0) endayminbal,
  nvl(tmp_bal.max_dpst_bal, 0) endaymaxbal,
  nvl(tmp_bal.avg_dpst_bal, 0) endayavgbal,
  case
    when tr.dpst_num is null then
      nvl(tmp.passivedays + 1, 0) +
    else
      0
  end passivedays
from tmp_last_remain tmp
  on tmp.dpst_num = f.dpst_num
full outer join tmp_dpst_sum_trnover tr
  on tr.dpst_num = f.dpst_num
full outer join tmp_min_max_avg tmp_bal
  on tmp_bal.dpst_num = f.dpst_num;
commit;

currDate := currDate + 1;
end loop;
end;

```

نکته: بعد از هر دستور اینسرت باید commit کنیم

مثلا این 0 (nvl(tr.trnoverbes, 0) جمع بستانکارا است و این صفر برای چیه؟ اگر امروز تراکنشی نداشته مقدارشو بزن صفر

***:

این case-when اینجا کارش چیه؟

passivedays: تعداد روزهای بدون عملکرد سپرده است

اگر اون روز تراکنش داشته باشد توی جدول tr رکوردی داره

پس میگه اگه توی این tr نیست یعنی تراکنشی نداشته tmp.passivedays روبیا به اضافه یک کن

که این tmp رکوردهای فکت دیروز است یعنی دیروز رو نگاه کن passivedays چیه پس

tmp.passivedays یعنی این سپرده دیروز passivedaysاش چند بوده و بعد بیا اینو به اضافه

یک کن و اگر توی فکت هم چیزی براش نیست صفر در نظر بگیر --< +

از جدول سورس اینجا استفاده نکرد؟ چون اونجا گروپ بای میخواست در صورتی که این جدوله

خودش الان گروپ بای شده است--< که این به تعداد تراکنش ها رکورد نداره و به تعداد سپرده هایی

که در ان تراکنش داشتن این رکورد داره یعنی اگر یک سپرده 10 تا تراکنش داشته باشه فقط یک

رکورد براش داره