

به نام آنکه جان را فکرت آموخت



## بخش چهارم: معماری پایگاه داده‌ها

مرتضی امینی

نیمسال اول ۹۲-۹۳

(محتویات اسلایدها برگرفته از یادداشت‌های کلاسی استاد محمدتقی روحانی رانکوهی است.)



☐ نیاز به یک معماری واحد از دیدگاه **داده شناسانه** (و نه دیدگاه عملکردی یا دیدگاه مولفه-مبنا)

☐ عدم وجود اتفاق نظر در چگونگی معماری پایگاه داده‌ها در سالهای آغازین ایجاد

☐ پیشنهاد معماری سه سطحی از سوی ANSI / SPARC



- سه سطح معماری ANSI، در واقع سه سطح تعریف و کنترل داده‌ها است.
- دو سطح در محیط انتزاعی و یک سطح در محیط فایلینگ منطقی.

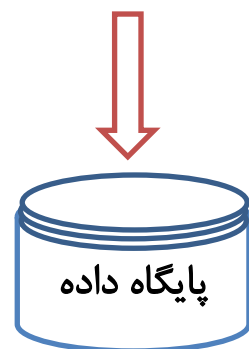
سطح خارجی  External Level

---

سطح ادراکی  Conceptual Level

---

سطح داخلی  Internal Level

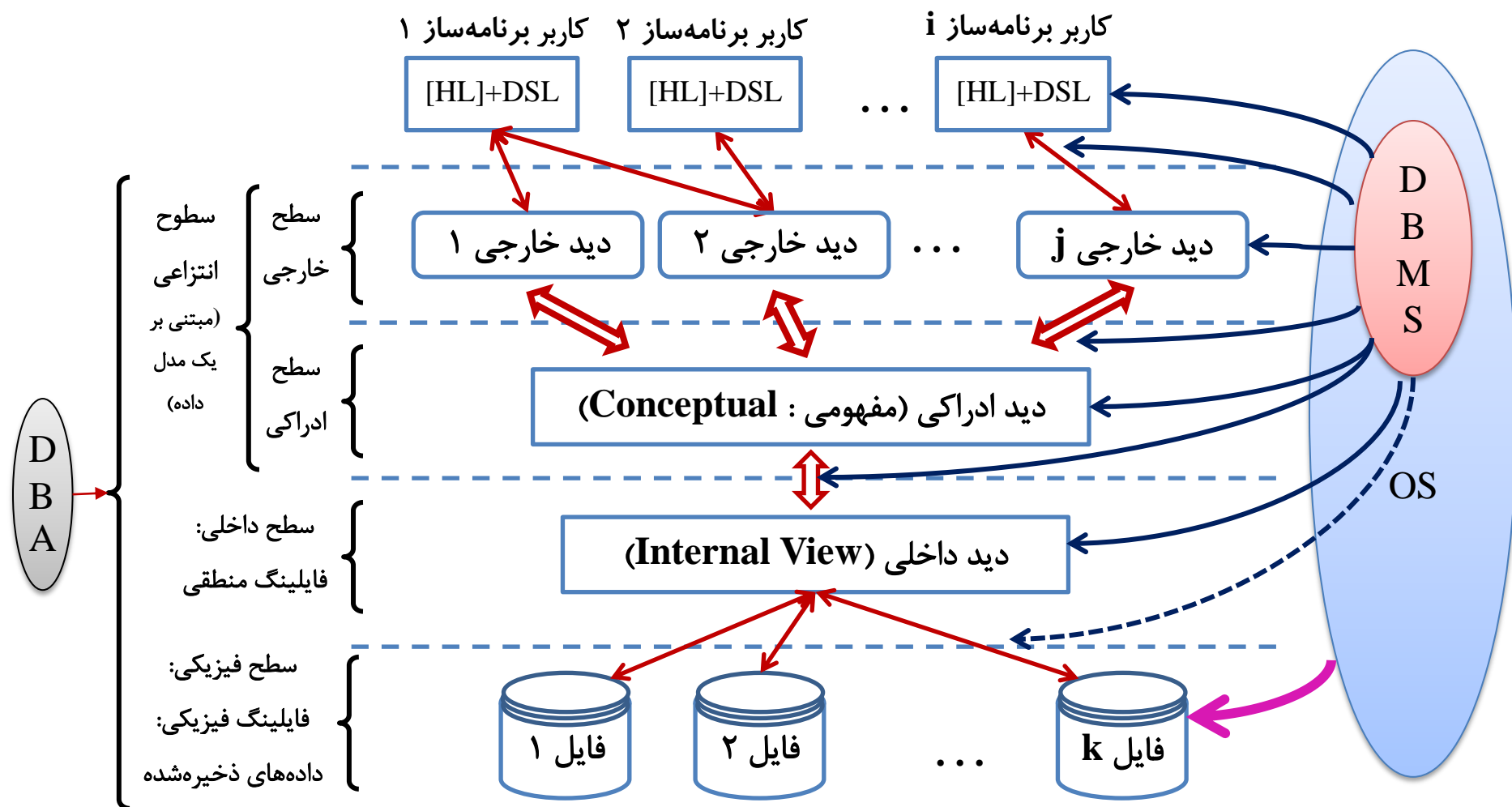




بخش چهارم: معماری پایگاه داده‌ها

نشان‌دهنده نگاشت (تبدیل) بین سطوح

معماری سه سطحی





## □ اجزای معماری سه سطحی پایگاه داده‌ها:

- ۱- کاربر برنامه‌ساز - User
- ۲- زبان میزبان - Host Language (HL): مانند زبانهای جاوا، C# و دلفی
- ۳- زبان داده‌ای فرعی (زیرزبان داده‌ای) - Data Sub Language (DSL): ممکن است در یک یا چند زبان میزبان ادغام شود

- 
- ۴- دید خارجی (نمای خارجی) → سطح خارجی
  - ۵- دید ادراکی (فرایافتی یا مفهومی) → سطح ادراکی
  - ۶- دید داخلی → سطح داخلی

- 
- ۷- فایل‌های فیزیکی
  - ۸- سیستم مدیریت پایگاه داده‌ها (کوتاه‌تر: س.م.پ.د.)
  - ۹- مدیر پایگاه داده‌ها (DBA)

- 
- ۱۰- تبدیلات بین سطوح



## دید (نمای) ادراکی (فرایافتی یا مفهومی)

دید طراح نسبت به داده‌های ذخیره‌شده (و نه‌ایتناً ذخیره‌شده) در پایگاه داده‌ها

✓ دیدی جامع : دربرگیرنده نیازهای همه کاربران محیط

✓ این دید مبتنی است بر یک ساختار داده‌ی مشخص از یک مدل داده مشخص در یک محیط مشخص؟؟؟

✓ مطرح در محیط انتزاعی (فرافیلی) ← مبتنی بر یک ساختار داده مشخص

✓ طراحی با عنصر (عناصر) ساختاری اساسی (مثلاً با فرض DS جدولی با جدول)

✓ پس از طراحی ← توصیف شود ← **شِمای ادراکی (Conceptual Schema)**



نوعی «برنامه» حاوی دستورات  $\left. \begin{array}{l} \text{DDL} \\ \text{DCL} \end{array} \right\}$  و نه دستورات DML

✓ شِمای ادراکی به سیستم مدیریت داده می‌شود و در کاتالوگ آن نگهداری می‌شود.



بخش چهارم: معماری پایگاه داده‌ها

CREATE TABLE COT ...

CREATE TABLE STT ...

CREATE TABLE STCOT ...

⋮



□ دید ادراکی (با فرض TDS): جدول‌های مبنای STT و STCOT و COT

کاتالوگ سیستم

در سیستم‌های جدولی: در تعدادی جدول  
که خود سیستم ایجاد می‌کند.

در کجا ؟

چگونه ؟

□ شمای ادراکی: همان تعریف جدول‌ها است.

□ به این جدول‌ها **جدول‌های مبنا** می‌گوییم.

□ اطلاعات شمای ادراکی به سیستم مدیریت داده می‌شود و در کاتالوگ آن نگهداری می‌شود.

کاتالوگ سیستم : متا داده‌ها (Data Dictionary : Meta Data)



□ حاوی : ✓ تمامی اطلاعات شمای ادراکی

✓ داده‌های کنترلی

✓ Data About Data

✓ ...



جدول systables:



systables

نام جدول	ایجاد کننده	تاریخ ایجاد	تعداد ستون	کلید اصلی	...
...	...	...	...	...	...

کاربر پیاده‌ساز : **CREATE TABLE STT ...**

سیستم : **INSERT INTO SYSTABLES**  
**VALUES ( 'STT' , 'c1' , 'd1' , 5 , 'STID' , ... )**

---

کاربر پیاده‌ساز : **DROP TABLE STCOT ...**

سیستم : **DELETE FROM SYSTABLES**  
**WHERE TNAME = 'STCOT'**





کاربر پیاده‌ساز : **ALTER TABLE STT**

□ اضافه کردن ستون به یک جدول:

**ADD SADDRESS CHAR (80)**

سیستم : **UPDATE SYSTABLES**

**SET ColN = 6**

**WHERE TNAME = 'STT'**

سیستم برای جدولی که تعداد ستون‌های آن تغییر می‌کند در سطح فایلینگ چگونه عمل می‌کند؟



آیا با دستور DELETE جدول کاتالوگ تغییر می‌کند؟



**DELETE FROM STT**

**WHERE STID='777'**



## دید (نمای) داخلی



سطحی است که فایل‌های منطقی پایگاه داده در آن تعریف می‌شود [توسط DBMS و در مواردی با

نظر طراح پایگاه داده]

✓ مطرح در سطح فایلینگ منطقی (و گاه مجازی) ←

✓ مبتنی بر یک [یا چند] ساختار فایل (با توجه به امکانات DBMS) →

1:1 پیش فرض (یک جدول : یک فایل)  
N:1 (چند جدول : یک فایل)  
1:N نادر (یک جدول : چند فایل)

✓ سطحی که فایل‌های منطقی پایگاه داده‌ها تعریف می‌شود. ←

✓ تناظر بین «ساخت» های سطح ادراکی و «ساخت» های سطح داخلی

Table	TableFile
STT	STTFile
COT	COTFile
STCOT	STCOTFile
...	...

تناظر 1:1 بین ساخت‌های سطح ادراکی و سطح داخلی






تعریف فایل‌ها  
کنترل فایل‌ها

□ توصیف دید داخلی ← **شِمای داخلی (Internal Schema)** ← دستورهای



 نوعی برنامه که توسط خود DBMS (و گاه براساس **اطلاعاتی** که طراح - پیاده‌ساز به سیستم می‌دهد) تولید می‌شود و شرح و وصف فایلینگ منطقی پایگاه داده‌هاست.

□ **توجه:** در شِمای داخلی انواع رکوردها تعریف می‌شوند و دستورهای لازم جهت ایجاد فایل‌ها و کنترل آنها در این شِما وجود دارد.

فرض کنید داریم:



Create Table STT ...

**TYPE**      STUDENT = **RECORD**

STUDENT-ID      : String ;

STUDENT-NAME    : String ;

STUDENT-LEV      : String ;

STUDENT-MJR      : String ;

STUDENT-DEPT    : String ;

شِمای داخلی ساده‌شده در یک زبان شبه پاسکال



□ نکته: اطلاعاتی که طراح-پیاده ساز به سیستم می‌دهد (مانند نمایه) در دید داخلی تاثیر می‌گذارد.

□ در سیستم‌های جدولی: خود سیستم روی کلید اصلی (PK) نمایه خودکار (Automatic Index)

ایجاد می‌کند. (عمدتاً B-Tree)

تفاوت نمایه خوشه‌ساز (Clustered) با نمایه ناخوشه‌ساز (Non Clustered) چیست؟



□ برای ایجاد نمایه روی دیگر ستون‌ها، پیاده‌ساز باید درخواست کند.

ایجاد نمایه بر روی ستون STNAME که PK نیست:



**CREATE INDEX SNNAMEX**

**ON STT ( STNAME )**

-----  
[ CLUSTERED ] ؟ خوشه‌بندی



ویژگی‌های ستون شاخص؟

✓ تغییر ناپذیر (حتی الامکان)

✓ پر کاربرد در کلاز WHERE

✓ ... ؟



حذف شاخص:

SNX INDEX DROP



با اجرای دستور

در سیستم چه اتفاقی می‌افتد؟  
DROP TABLE  
DROP INDEX



مثالی از وضعیتی بیان کنید که براساس آن طراح-پیاده‌ساز تصمیم به ایجاد نمایه می‌گیرد.



# دید منطقی DBMS نسبت به داده‌های ذخیره‌شده

۱۴

بخش چهارم: معماری پایگاه داده‌ها





# دید منطقی DBMS نسبت به داده‌های ذخیره‌شده (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۱۵

می‌داند: جنبه های فایلینگ منطقی [مجازی] ....

- ☐ چگونگی نشست فایل‌ها روی دیسک
- ☐ استراتژی دستیابی (مثلا نمایه) چگونه پیاده‌سازی شده‌اند.
- ☐ اندازه بلاک (Block) فیزیکی
- ☐ جزئیات تکنیک‌های Blocking
- ☐ Locality رکوردهای فایل‌ها
- ☐ توالی منطقی رکوردها چگونه پیاده‌سازی شده‌اند
- ☐ ...

نمی‌داند: جنبه های فایلینگ فیزیکی

DBMS ☐

Locality چیست و بر کدام یک از عملیات روی فایل‌ها تاثیر می‌گذارد؟



سطح فایلینگ مجازی چیست؟





□ در بعضی از سیستم‌های مدیریت جدید، سیستم مدیریت، کل فضای پایگاه داده را به صورت مجموعه‌ای از مجموعه صفحات می‌بیند، یعنی نوعی **نمای مجازی** از داده‌های ذخیره‌شده در پایگاه داده دارد.

در سطح فایلینگ مجازی  $DB = \{ \{pages\} \}$



شماره صفحات	تعداد صفحات	نام جدول
p1 ... p10	10	STT
p15 ... p29	15	COT
P101 ... P1000	900	STCOT

**SELECT** STT.\*

**FROM** STT

**WHERE** STID = '444'

**DBMS :** **READ** P1

(فرض کنید '444' در P1 است)

سیستم مدیریت، بعد از خواندن صفحات چه می‌کند؟







## □ دید (نمای) خارجی



دید کاربر (برنامه ساز) خاص است نسبت به داده‌های ذخیره‌شده [مثلاً دید یک AP نویسنده]

✓ دید جزئی (Partial): دربرگیرنده نیازهای داده‌ای یک کاربر مشخص [برای یک AP مشخص]

✓ مطرح در سطح انتزاعی ← مبتنی بر یک ساختار داده‌ای مشخص



آیا این ساختار داده همان ساختار داده سطح دید ادراکی است؟

✓ روی دید ادراکی طراحی و تعریف می‌شود.

✓ } یک کاربر ← چند دید متفاوت  
چند کاربر ← یک دید مشترک



✓ توصیف دید خارجی ← **شِمای خارجی**



نوعی «برنامه» که کاربر سطح خارجی می‌نویسد، حاوی دستورات «تعریف داده‌ها» و **معدود** دستورات «کنترل داده‌ها» ( **کجایی؟** چرا معدود؟ )



✓ **شِمای خارجی** ← ذخیره در کاتالوگ

در سیستم‌های جدولی، دید خارجی خود نوعی جدول است، اما **مجازی** (Virtual Table) و نه ذخیره‌شده



دید خارجی در واقع پنجره‌ای است که از آن کاربر خارجی محدوده‌ی داده‌ای خود را می‌بیند و نه بیشتر.





بخش چهارم: معماری پایگاه داده‌ها

کاربر ۱

v1

STID STNAME

777 st7

444 st4

⋮ ⋮

دو ستون از یک جدول

v2

CONUM

COTITLE

دگرنامی ستون



STT

STID STNAME

777 st7

888 st8

444 st4

⋮ ⋮

STLEV

bs

ms

bs

⋮

STMJR

phys

math

comp

⋮

STDEID

d11

d12

d14

⋮

COT

COID

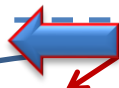
...

STCOT

STID

COID ...

تناظر یک به یک



ST FILE

COT FILE

STCOT FILE



بخش چهارم: معماری پایگاه داده‌ها

کاربر ۲

v1

STID STNAME

777

st7

دید مشترک با کاربر ۱

444

st4

⋮

⋮



STT

STID	STNAME	STLEV	STMJR	STDEID
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

COT

COID	...

STCOT

STID	COID ...

ST FILE

COT FILE

STCOT FILE



بخش چهارم: معماری پایگاه داده‌ها

کاربر ۳

v1

STNUM	STNAME	COTITLE	TR	YR
-------	--------	---------	----	----

دید روی بیش از یک جدول



STT

STID	STNAME	STLEV	STMJR	STDEID
777	st7	bs	phys	d11
888	st8	ms	math	d12
444	st4	bs	comp	d14
⋮	⋮	⋮	⋮	⋮

COT

COID	...

STCOT

STID	COID ...

ST FILE

COT FILE

STCOT FILE



بخش چهارم: معماری پایگاه داده‌ها

کاربر ۴

v2

STID

STNAM

TR

YR

AVG

صفت مجازی



STT

STID

STNAME

STLEV

STMJR

STDEID

777

st7

bs

phys

d11

888

st8

ms

math

d12

444

st4

bs

comp

d14

:

:

:

:

:

COT

COID

...

STCOT

STID

COID ...

ST FILE

COT FILE

STCOT FILE



از این مثال‌ها نتیجه می‌گیریم که تعریف، طراحی و توصیف دید خارجی در سیستم‌های جدولی از پویایی بالایی برخوردار است.

یعنی انواع جدول‌های مجازی را می‌توان روی لایه‌های زیرین تعریف کرد.

تعریف شمای خارجی کاربر ۱ (با استفاده از مفهوم دید):



```
CREATE VIEW V1 [(STID, STNAME)]
AS SELECT STT.STID, STT.STNAE
FROM STT;
```

```
CREATE VIEW V2 [(SN, SJ, SL)]
AS SELECT STID, STJ, STL
FROM STT
WHERE STJ != 'phys;
[WITH CHECK OPTION]
```



شرط تعریف دید

در شرط تعریف دید می‌توان از نام ستونی که در محدوده دید نیست استفاده کرد.



هر کاربر با اجازه Admin می‌تواند دید (View) خودش را داشته باشد (Sub-database).



دستور SELECT در متن دستور تعریف دید، «اجرایی» نیست بلکه «اعلانی» است. یعنی هیچ داده‌ای بازیابی نمی‌شود و صرفاً برای اعلام محدوده داده‌ای کاربران است. □



تا آنجا که به تعریف دید مربوط است هر دستور SELECT **معتبر** با هر میزان پیچیدگی را می‌توان در CREATE VIEW نوشت.



□ **تمرین:** مثال کاتالوگ پیش‌دیده را به نحوی گسترش دهید که اطلاعات (نه داده‌ها) شمای داخلی و شمای خارجی دیده شده را بتوان در آن ذخیره کرد (جدول دیگری برای کاتالوگ تعریف کنید که بتوان این شمایها را در آن ذخیره کرد).

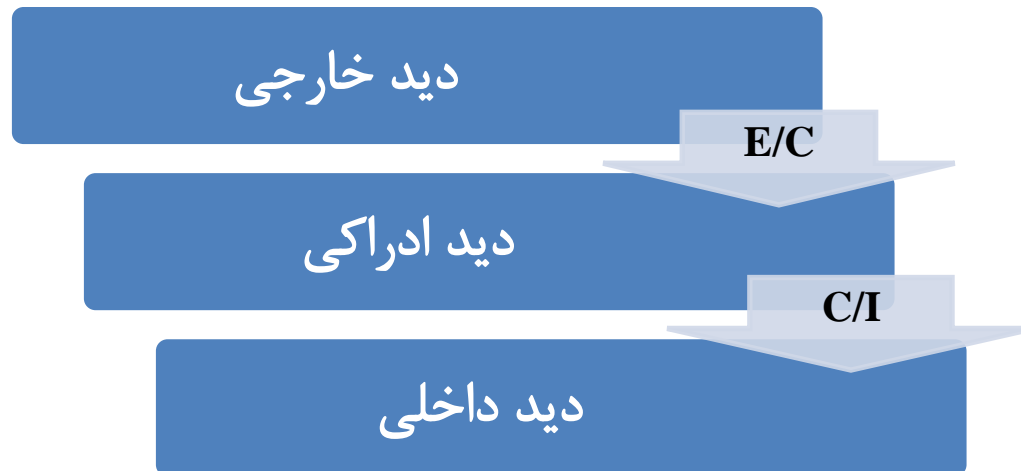




□ نگاشت یا تبدیل بین سطوح (عملیات از دید خارجی در DB):

External to Conceptual Mapping :E/C □

Conceptual to Internal Mapping :C/I □



آیا تبدیل دیگری هم متصور است؟





بازیابی: کاربر حق دارد در محدوده دید خود عمل بازیابی را انجام دهد.

□ عملیات در شمای خارجی

درج

حذف

بروزرسانی

ذخیره‌سازی: به تشخیص Admin مجاز به انجام است.

□ هر دستور [حکم] عمل‌کننده در شمای خارجی (روی دید خارجی)،

□ تبدیل می‌شود به  $N \geq 0$  دستور عمل‌کننده در شمای ادراکی (روی دید ادراکی)

□ و سپس به قطعه برنامه‌ای عمل‌کننده در شمای داخلی (روی دید داخلی)

□ و نهایتاً به عملیاتی در فایل‌های فیزیکی.



عملیات بازیابی: چون دید خارجی در سیستم‌های جدولی، به هر حال نوعی جدول است، برای بازیابی از همان دستور SELECT استفاده می‌کنیم.

```
SELECT V2.SN  
FROM V2  
WHERE SL='ms'
```

E/C

سیستم در نگاشت E/C، شرط یا شرایط داده شده در تعریف دید را AND می‌کند با شرط یا شرایط داده

شده در پرسجوی روی دید. به این عمل، گاه **محاسبه دید** (View Computation) هم می‌گویند.

```
SELECT STT.STID  
FROM STT  
WHERE STL='ms'  
AND STJ != 'phys'
```

C/I



به واحد رکورد

ناحیه پیام      بافر سیستم

```
OPEN  STFILE (R, SysBuf, MessageArea, ...)  
LREAD STFILE  ON  STLINDEX.value='ms';  
  
...  
IF SysBuf.STJ != 'phys'  
    MOVE SysBuf.STID INTO  UBuf[SN]  
  
...  
LOOP Control;
```

فایلینگ منطقی  
در محیط

جستجوی فیزیکی

PSEEK

خواندن فیزیکی

PREAD

فایلینگ فیزیکی  
در محیط

به واحد بلاک



❑ لزوماً از همه انواع دیدها نمی‌توان عملیات ذخیره‌سازی در DB انجام داد.

❑ همه انواع دیدها قابل بهنگام‌سازی (Updatable) نیستند.

❑ محدودیت‌هایی هم در عمل و تا حدی در تئوری وجود دارد.

❑ **دید از نظر قابلیت عملیات ذخیره‌سازی** (بستگی دارد به ساختار دید و مکانیزم تعریف آن):

❑ **پذیرا** (Updatable): می‌توان از آنها عملیات ذخیره‌سازی انجام داد ولی گاه مشکلاتی دارند.

❑ **ناپذیرا** (Non Updatable): تبدیل E/C انجام شدنی نیست.

تعریف شده روی **یک** جدول مبنا

❑ دید

تعریف شده روی **بیش** از یک جدول مبنا ← در عمل ناپذیرا، اما در تئوری بعضی‌ها پذیرا هستند.



# عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا

بخش چهارم: معماری پایگاه داده‌ها

۳۰

☐ دید تعریف شده روی یک جدول مبنا

☐ دید دارای کلید جدول مبنا (Key Preserving) ← پذیرا (در عمل و تئوری) اما مشکلاتی هم دارد.

☐ دید فاقد کلید جدول مبنا (Non Key Preserving) ← ناپذیرا

☐ دید دارای ستون [صفت] مجازی (دیدهای آماری) ← ناپذیرا



# عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۳۱

دید حافظ کلید تعریف شده روی یک جدول مبنا ☐

V2	SN	SJ	SL
	888	ms	math
	444	bs	comp
	⋮	⋮	

STT	STID	STNAME	STL	STJ	STD
	777	st7	bs	phys	d11
	888	st8	ms	math	d12
	444	st4	bs	comp	d14
	⋮	⋮	⋮	⋮	⋮

```
CREATE VIEW V2 [(SN, SJ, SL)]
AS SELECT STID, STJ, STL
FROM STT
WHERE STJ != 'phys'
[WITH CHECK OPTION]
```



# عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۳۲

□ فرض بر مجاز بودن کاربر به انجام عمل داریم و لذا صرفاً شدنی بودن را بررسی می‌کنیم.

□ در دید حافظ کلید انجام عملیات سطری امکان‌پذیر است.

□ زیرا تناظر یک به یک بین سطرهاى دید و سطرهاى جدول مبنا برقرار است.

```
DELETE FROM V2  
WHERE SN='444'
```

E/C

```
DELETE FROM STT  
WHERE STID='444' AND STJ != 'phys'
```

حذف سطر در دید حافظ کلید



□ الان این سطر از جدول STT حذف می‌شود و اگر کاربر دیگری این سطر را در دیدش داشته باشد، دیگر به این سطر دسترسی ندارد.





# عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۳۳

به‌نگام‌سازی در دید حافظ کلید



```
UPDATE V2  
SET SJ='IT'  
WHERE SN='444'
```

E/C

```
UPDATE STT  
SET STJ='IT'  
WHERE STID='444' AND STJ != 'phys'
```



# عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۳۴



از نظر تئوریک درخواست زیر به دلیل **عدم رعایت محدودیت دید** باید رد شود.

**UPDATE V2**

**SET** SJ='phys'

**WHERE** SN='888'

□ در عمل: اگر از عبارت [with check option] استفاده کنیم، سیستم رد می‌کند، وگرنه درخواست

انجام می‌شود اما ...

E/C

**UPDATE STT**

**SET** STJ='phys'

**WHERE** STID='888' AND STJ != 'phys'

□ حال اگر بنویسیم:

**SELECT V2.\* FROM V2**

□ سطر با کلید 888 دیگر در دید کاربر نمی‌آید!



# عملیات ذخیره سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده ها

۳۵



```
INSERT INTO V2  
VALUES ('555', 'chem', 'bs')
```

E/C

```
INSERT INTO STT  
VALUES ('555', ?, 'chem', 'bs', ?)
```

□ اگر هر کدام از ستون های نهان از دید کاربر، محدودیت هیچ مقدار ناپذیری داشته باشند، درخواست رد می شود.

□ حال اگر به جای 555 بنویسیم 777، درخواست رد می شود (تبدیل E/C انجام نمی شود) به دلیل عدم رعایت محدودیت یکتایی مقادیر کلید.

□ حال اگر به جای chem بنویسیم phys، همان پیش می آید که در مثال UPDATE دیدیم.



## عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۳۶

☐ دلایل رد شدن درخواست عمل ذخیره‌سازی در دید تک جدولی حافظ کلید :

☐ عدم رعایت محدودیت دید

☐ عدم رعایت محدودیت یکتایی مقادیر کلید

☐ عدم رعایت محدودیت هیچ‌مقدارناپذیری ستون‌های نهان

☐ ...



# عملیات ذخیره سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده ها

۳۷

□ دید تعریف شده روی یک جدول مبنا و فاقد کلید

چون این دید فاقد کلید است، امکان انجام عملیات سطری وجود ندارد.



```
CREATE VIEW V3  
AS SELECT STNAME, STJ  
FROM STT
```

□ درخواست زیر انجام نمی شود، چون معلوم نیست کدام سطر از رابطه باید حذف شود. پس تبدیل E/C ناممکن است، مگر اینکه بپذیریم این درخواست به صورت مکانیکی انجام شود؛ یعنی تمام سطرهای حائز شرط داده شده (مجموعه ای از سطرها) حذف شوند.

```
DELETE FROM V3  
WHERE STNAME='ali' AND STJ='comp'
```

□ اگر کاربر این پیامد را بپذیرد مشکلی نیست، اما در عمل سیستم ها نمی پذیرند!

□ در دید V3 انجام INSERT نیز غیرممکن است.



# عملیات ذخیره سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده ها

۳۸

حال اگر در تعریف V3، DISTINCT بزنیم چه پیش می آید؟



```
CREATE VIEW V3  
AS SELECT DISTINCT STNAME, STJ  
FROM STT
```

❑ فرقی نمی کند، باز هم همان مشکل پابرجاست:

```
DELETE FROM V3  
WHERE STNAME='a'
```

E/C

تبدیل می شود به حذف مجموعه ای از سطرها




# عملیات ذخیره سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده ها

۳۹

دید تعریف شده روی یک جدول مبنا دارای ستون مجازی 

 این دیدها هم در عمل و هم در تئوری ناپذیرا هستند.



V4	PN	SQ
	P1	100
	P2	210
	P3	80

```
CREATE VIEW V4 (PN, SQ)
AS SELECT P#, SUM(QTY)
FROM SP
GROUP BY P#
```

---

SP	S#	P#	QTY
	S1	P1	100
	S1	P2	140
	S2	P3	80
	S2	P2	70



# عملیات ذخیره‌سازی از دید تعریف شده روی یک جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۴۰

□ انجام عملیات سطری در دید V4 غیرممکن است.

**DELETE FROM V4**  
**WHERE PN='p1'**



سطری نیست، با نوعی تفسیر  
می‌توان گفت که مجموعه‌ای  
از سطرها را حذف می‌کند.

□ از لحاظ تئوریک هم دید V4 نباید پذیرا باشد.

□ زیرا جدول V4 (که مجازی است) و جدول مبنای SP با هم تعارض معنایی (Semantic Conflict)

دارند. یعنی مسند بیانگر معنای رابطه V4 اساساً با مسند بیانگر رابطه SP تفاوت دارد.

[دربحث رابطه‌ای خواهیم دید که هر رابطه (جدول) یک معنا دارد و در اینجا این دو رابطه هیچ ربطی از نظر معنایی با هم ندارند].





# عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا

بخش چهارم: معماری پایگاه داده‌ها

۴۱

دیده‌های تعریف شده روی بیش از یک جدول ☐

☐ در عمل این دیده‌ها ناپذیرا هستند و دخالت خود برنامه‌ساز لازم است.

V5: T1 JOIN T2      دید پیوندی (پیوند طبیعی)

V6: T1 UNION T2

V7: T1 INTERSECT T2

V8: T1 EXCEPT T2

PK-PK: ستون پیوند در هر دو جدول PK است. پذیرا و بدون مشکل

PK-FK: پذیرا به شرط پذیرش پیامدها

FK-FK

(Non-Key) NK-NK

دید پیوندی ☐



# عملیات ذخیره سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده ها

۴۲

دید پیوندی PK-PK ☐

مثال V5 همان STT است اما این بار به صورت یک دید تعریف شده است.



V5	STID	STNAME	STL	STJ	STD
	777	st7	bs	phys	d11
	888	st8	ms	math	d12
	444	st4	bs	comp	d14
	⋮	⋮	⋮	⋮	⋮

```
CREATE VIEW V5
AS SELECT ST1.*, ST2.*
FROM ST1 JOIN ST2
```

ST1	STID	STNAME	STL
	777	st7	bs
	888	st8	ms
	444	st4	bs
	⋮	⋮	⋮

ST2	STID	STJ	STD
	777	phys	d11
	888	math	d12
	444	comp	d14
	⋮	⋮	⋮



# عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۴۳

یک دستور اجراشونده در شِمای خارجی تبدیل می‌شود به دو دستور در شِمای ادراکی. ☐

**INSERT INTO V5**

**VALUES** ('999', 'St9', 'chem', 'bs', 'D15')

E/C

**INSERT INTO ST1**

**VALUES** ('999', 'St9', 'bs')

**INSERT INTO ST2**

**VALUES** ('999', 'chem', 'D15')

عمل DELETE در این دید تبدیل می‌شود به دو عمل حذف از جدول‌های مبنایی زیرین و عمل UPDATE (بسته به ☐

ستونی که می‌خواهیم بروز کنیم) به یک یا دو عمل بهنگام‌سازی در جدول‌های زیرین تبدیل می‌شود.



# عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۴۴

دید پیوندی PK-FK ☐



```
CREATE VIEW V6
AS SELECT STT.*, STCOT.*
FROM STT JOIN STCOT
```

☐ درج در این دید تبدیل می‌شود به درج یک تاپل ناقص در STT به شرط آنکه شماره دانشجویی تکراری نباشد. ولی در STCOT حتما یک تاپل درج می‌شود.

```
INSERT INTO V6
VALUES ('9212345', 'Amir', '40638', 15)
```

E/C

```
INSERT INTO STT
VALUES ('9212345', 'Amir', ?, ?, ?)
```

```
INSERT INTO STCOT
VALUES ('9212345', '40638', 15)
```



## عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۴۵

☐ حذف از این دید مشکل دارد.

☐ اگر از هر دو جدول حذف شود، منجر به حذف داده‌های ناخواسته می‌شود

☐ با حذف یک سطر از جدول STT، برای حفظ جامعیت ارجاعی نیز لازم است یک تعداد سطر دیگر از

STCOT حذف شود، مگر آنکه فقط از STCOT حذف کنیم و از STT سطر مربوطه را حذف نکنیم.

☐ عمل به‌نگام‌سازی نیز مساله مشابه حذف ممکن است داشته باشد.

دید حاصل از پیوند FK-FK و دید حاصل از پیوند NK-NK چه رفتاری در عملیات ذخیره‌سازی دارند؟





# عملیات ذخیره‌سازی از دید تعریف شده روی چند جدول مبنا (ادامه)

بخش چهارم: معماری پایگاه داده‌ها

۴۶

□ دید حاصل از اجتماع، اشتراک، و تفاضل

□ این دیدها از لحاظ تئوری مشکلی در عملیات ذخیره‌سازی ندارند، هرچند نظرات مختلفی مطرح است.

عمل دید	درج	حذف	بهنگام‌سازی
$R_1 \cup R_2$	درج تاپل در $R_1$ و/یا $R_2$	حذف تاپل از $R_1$ و/یا $R_2$	بهنگام‌سازی تاپل در $R_1$ و/یا $R_2$
$R_1 \cap R_2$	درج تاپل در $R_1$ و $R_2$	حذف تاپل از $R_1$ و/یا $R_2$	بهنگام‌سازی تاپل در $R_1$ و $R_2$
$R_1 - R_2$	درج تاپل در $R_1$	حذف تاپل در $R_1$	بهنگام‌سازی تاپل در $R_1$



□ موضوع دیدهای پذیرا در SQL استاندارد چندان روشن نیست. در SQL 2003 دیدهایی که تمام شرایط زیر را داشته باشند، قابل بهنگام‌سازی (درج، حذف و بروزرسانی) هستند.  
[توجه: ممکن است برخی دیگر از دیدها هم قابل بهنگام‌سازی باشند.]

۱- عبارت تعریف‌کننده دید، یک عبارت SELECT ساده باشد (یعنی شامل عملگرهای JOIN، UNION، INTERSECT و EXCEPT نباشد).

۲- در عبارت SELECT گزینه DISTINCT وجود نداشته باشد.

۳- در کلاز FROM عبارت SELECT، فقط یک جدول وجود داشته باشد.

۴- جدول قید شده در کلاز FROM، یک جدول مبنا یا یک دید قابل بهنگام‌سازی باشد.



**۵-** در فهرستِ نام ستون‌ها در عبارت `SELECT`، ستون‌های موردنظر باید در جدول مبنا متناظر داشته باشند و به یک ستون از جدول مبنا بیش از یک بار ارجاع وجود نداشته باشد. ضمناً حاوی ستون کلید باشد.

**۶-** در عبارت `SELECT`، کلاز `GROUP BY` و/یا کلاز `HAVING` وجود نداشته باشد.

**۷-** کلاز `WHERE` در عبارت `SELECT` حاوی کلاز `FROM` نباشد به گونه‌ای که در آن به همان

جدولی ارجاع داده شده باشد که در کلاز `FROM` ذکر شده در شرط ۴.

نتیجه اینکه عملاً دیدهایی که یک زیرمجموعه افقی-عمودی دارای کلید از یک جدول مبنا (یا از دید قابل

بهنگام‌سازی) باشند، قطعاً قابل بهنگام‌سازی هستند.

**[توجه:** به شرط رعایت محدودیت‌های جامعیتی مانند یکتایی کلید و هیچمقدارناپذیری]





❑ معایب مفهوم دید:

❑ محدودیت [و مشکلات] در عملیات ذخیره‌سازی

❑ فزونکاری (overhead) برای انجام تبدیل E/C (محاسبه دید). راه حل: استفاده از تکنیک دید

ذخیره‌شده



## تکنیک دید ذخیره شده [ساخته شده] (Materialized View) □

□ در این تکنیک، دید در سیستم ذخیره می‌شود؛ یعنی دیگر مجازی نیست و جدول ذخیره شده است. تا در هر بار مراجعه به دید لازم نباشد تبدیل E/C انجام شود.

□ **هدف:** برای افزایش سرعت عملیات بازیابی.

□ **شرط استفاده:** در عمل از این تکنیک وقتی استفاده می‌کنیم که داده‌های ذخیره شده در جدول‌های مبنای زیرین حتی‌الامکان تغییر نکنند. به بیان دیگر، نرخ عملیات ذخیره‌سازی در جدول‌های زیرین پایین باشد. زیرا اگر جدول‌های زیرین تغییر کنند، تغییرات متناسباً در جدول‌های دید باید اعمال شوند و این خود سربار ایجاد می‌کند.

□ **کاربرد:** در برنامه‌های آماری، گزارش‌گیری‌ها و برنامه‌های داده‌کاوی (Data Mining)

□ دید ذخیره شده (Stored View) در SQL چگونه پیاده‌سازی می‌شود؟ با `CREATE SNAPSHOT`؟



## □ مزایای مفهوم دید خارجی:

□ فراهم‌کننده محیط انتزاعی فرافایلی برای کاربران با پویایی بالا

□ اشتراک داده‌ها (Data Sharing) ← داده‌ها یک بار ذخیره می‌شوند و کاربران بسته به نیاز خود از داده‌های ذخیره‌شده به صورت هم‌روند استفاده می‌کنند.

□ مکانیزم اتوماتیک برای تامین امنیت داده‌های زیرین. ← از طریق مفهوم داده مخفی ( Hidden Data)، زیرا کاربر خارج از محدوده دید خود هیچ نمی‌بیند (داده‌های نهان امن هستند).

□ تامین‌کننده استقلال داده‌ای (مفهوم اساسی در تکنولوژی DB؛ هم مزیت و هم از اهداف مهم تکنولوژی DB).

□ امکانی است برای کوتاه‌نویسی یا ماکرونویسی پرسش‌ها.



☐ چه زمانی از مفهوم دید استفاده نمی‌کنیم؟

☐ هنگامی که سیستم تک‌کاربره باشد.

☐ هنگامی که به تشخیص admin برای افزایش کارایی سیستم، برخی برنامه‌ها را مستقیماً روی شمای ادراکی (جداول مبنایی) بنویسیم.

☐ هنگامی که کاربر نیازمند انجام عملیات ذخیره‌سازی باشد و از طریق دید امکان آن وجود نداشته باشد.



□ مفهوم استقلال داده‌ای [DI] (جدایی برنامه‌ها از داده‌ها):

□ مصونیت (تاثیرناپذیری) برنامه‌های کاربران [در سطح خارجی] در قبال تغییرات در سطوح زیرین معماری DB.

□ چرا نباید برنامه‌ها تغییر کنند؟

□ چون هر تغییر در برنامه‌ها، هزینه تولید و پشتیبانی و بازتولید برنامه‌ها را بالا می‌برد.

استقلال داده‌ای فیزیکی (PDI)

استقلال داده‌ای منطقی (LDI)

□ استقلال داده‌ای (DI)



## استقلال داده‌ای فیزیکی (PDI) ☐

☐ مصونیت برنامه‌های کاربرانِ سطح خارجی در قبال تغییرات در شمای داخلی DB

☐ تغییرات در شمای داخلی شامل تغییر در جنبه‌های فایلینگ پایگاه

▪ ساختار فایل، طول رکورد، طرز ذخیره‌سازی فایل روی دیسک، گاه با دخالت طراح فیزیکی و گاه فقط توسط

DBMS

☐ زیرا کاربران با مفهوم دید کار می‌کنند که اساساً در سطح فرافایلی مطرح است و برنامه‌ها درگیر

جنبه‌های فایلینگ نیستند.



## استقلال داده‌ای منطقی (LDI) ☐

☐ مصونیت برنامه‌های کاربران در قبال تغییرات در شمای ادراکی DB.

☐ در سیستم‌های پایگاهی تا حد زیادی این استقلال تامین است ولی نه صددرصد.

☐ تغییر در شمای ادراکی } رشد پایگاه داده‌ها (DB Growth)  
تغییر سازمان پایگاه داده‌ها [سازماندهی مجدد DB] (DB Restructuring)

☐ **نکته:** تغییراتی که مورد بررسی قرار می‌دهیم، تغییراتی است که از داده‌ها و ساختار موجود **نمی‌کاهد**،

چرا که تغییرات کاهشی، قطعاً بر روی برنامه‌های سطح خارجی تاثیر می‌گذارد و استقلال داده‌ای حفظ نمی‌شود.



❑ چرا رشد DB: مطرح شدن نیازهای جدید

❑ اضافه شدن ستون(های) جدید به جدول(ها)

❑ ایجاد جدول‌های جدید

❑ استقلال داده‌ای منطقی (LDI) در قبال رشد DB، به کمک مفهوم دید تقریباً صددرصد تامین است، زیرا

کاربر دارای یک دید، خارج از محدوده آن دید، هیچ نمی‌بیند.





شده است.

دیدهای پیش‌تر تعریف شده را روی جدول STT در نظر می‌گیریم. حال نیاز جدیدی برای کاربر مطرح

V1  
STID STNAME

V2

V9  
STID STADR

کاربر ۱

:

:

کاربر ۲

:

STT	STID	STNAME	STL	STJ	STD	STADR
	777	st7	bs	phys	d11	
	888	st8	ms	math	d12	
	444	st4	bs	comp	d14	
	:	:	:	:	:	

ALTER TABLE STT

ADD COLUMN STADR CHAR(70)

این گسترش در سطح فایلینگ چگونه انجام می‌شود؟



□ آیا پیرو نیاز جدید کاربر در حد ستون، طراح همیشه جدول مبنا را گسترش می‌دهد؟

□ خیر، زیرا ممکن است آن ستون مجازی (محاسبه شدنی) باشد.

□ **سازماندهی مجدد DB** یعنی طراح به هر دلیلی طراحی منطقی DB را تغییر دهد. مثلاً یک جدول مبنای

موجود را به دو جدول تجزیه عمودی کند و طبعاً شمای ادراکی هم تغییر می‌کند. می‌خواهیم ببینیم LDI در قبال این تغییر تا چه حد تامین است.

□ در این حالت، LDI به کمک مفهوم دید و امکان تعریف **دید روی دید** (View Definition on View)،

تا حدی تامین است.





```
CREATE TABLE ST1
(STID ...,
...
STL ...)
PRIMARY KEY STID
```

شِمای جدید: ☐

```
CREATE TABLE ST2
(STID ...,
...
STD ...)
PRIMARY KEY STID
```

```
INSERT INTO ST1
(SELECT STID, STNAME, STL
FROM STT)
```

```
INSERT INTO ST2
(SELECT STID, ..., STD
FROM STT)
```

مهاجرت داده‌ها (Data Migration)



□ با حذف جدول مبنای STT، دیدهای قبلاً تعریف شده روی آن نامعتبر می‌شوند و در نتیجه برنامه‌هایی که روی آنها کار می‌کردند، دیگر اجرا نمی‌شوند و LDI دیگر تامین نیست مگر اینکه طراح و پیاده‌ساز تدبیری

بیندیشد.



✓ جدول STT را با همان نام و ساختار به شکل یک دید تعریف می‌کنیم، با مکانیزم پیوند (دید روی دید):

```
CREATE VIEW STT
AS SELECT STID, ..., STD
FROM ST1 JOIN ST2
```

```
DROP TABLE STT
```

□ تعریف این دید وارد کاتالوگ سیستم می‌شود. ← دیدهای قبلاً تعریف شده معتبر می‌شوند.



□ با این تدبیر، LDI برای برنامه‌هایی که بازیابی انجام می‌دهند، صددرصد تامین می‌شود، به قیمت افزایش سربار برای انجام تبدیل E/E علاوه بر E/C و C/I. زیرا از تکنیک **دید روی دید** استفاده کرده‌ایم.

□ اما LDI برای برنامه‌هایی که عملیات ذخیره‌سازی انجام می‌دادند، ممکن است تامین نباشد. زیرا این بار STT خود یک دید است و دیدها در عملیات ذخیره‌سازی عمدتاً مشکل دارند. ولی در این مثال خاص از نظر **تئوریک** مشکلی بروز نمی‌کند. ← چون STT یک دید پیوندی PK-PK است.



□ دلایل این نوع تجزیه (که کلید در هر دو جدول باشد) چه می‌تواند باشد؟

□ افزایش کارایی سیستم در رده فایلینگ با فرض 1-Table:1-File برای بعضی برنامه‌ها (مثلاً برنامه‌هایی

با فرکانس بالاتری نسبت به ستون‌های ST1 و با فرکانس پایین‌تری به ستون‌های ST2 ارجاع داشته

باشد، فایل‌ها را جدا می‌کند).

□ توزیع داده‌ها در سایت‌ها وقتی پایگاه داده توزیع شده (DDB) داشته باشیم.

□ کاهش حجم Null Value

□ بهینه‌سازی طراحی (رجوع شود به بحث نرمال‌سازی رابطه‌ها)

□ ...



**پرسش و پاسخ ...**

**amini@sharif.edu**