# Ešte o transakciách

- Atomičnosť a trvanlivosť použiteľnosť žurnálu
  - RC recoverable
  - ACR bez kaskádového rollbacku
  - Striktné (strict)
- SQL úrovne izolácie
- Deadlock and starvation
  - ignorovanie (pštrosí algoritmus)
  - predchádzanie (bankárov algoritmus)
  - prevencia (čakaj alebo zomri, čakaj alebo udri)
  - detekcia
- Kontrolné body (checkpoints)
- Zálohovanie (back up)
  - úplný
  - diferenciálny

# Príklad

| $T_1$ | $T_2$ |
|---|---|
| … | |
| w(a) | |
| | r(a) |
| | … |
| | w(b) |
| | c |
| ... | |

Journal (undo log)

$T_1$ start

…

$T_2$ start

$T_1$ write a $a_{old}$

$T_2$ read a

…

$T_2$ write b $b_{old}$

$T_2$ commit

Je tento commit vporiadku?

S: … $s_1$ … $s_2$ $w_1a$ $r_2a$ $w_2b$ $c_2$ …$a_1$

Ako urobiť rollback keď $T_1$ abortuje ?

Ak transakcia úspešne skončí, musia sa jej výsledky zachovať. Ak abortuje, nesmie po nej zostať ani stopa.

# Spresnenie definície „číta z“

$T_i$ číta z $T_j$ v rozvrhu $S$ $(T_j \rightarrow_S T_i)$

Ak

1) $w_j(A) \prec_S r_i(A)$ $T_j$ píše A predchádza rozvrhu S $T_i$ číta A

2) $a_j \not\prec_S r_i(A)$ Abort $T_j$ nepredchádza rozvrhu S $T_i$ číta A

3) Ak v rozvrhu S existuje $T_k$ také, že $w_j(A) \prec_S w_k(A) \prec_S r_i(A)$, potom $a_k \prec_S r_i(A)$.

Je to rovnaká definícia ako v predošlej prednáške, navyše berie do úvahy možnosť abortu transakcií.

Definícia: Hovoríme, že rozvrh S je recoverable (RC), ak pre každú dvojicu transakcií $(T_i,T_j)$ v rozvrhu S $T_i \rightarrow_S T_j \Rightarrow c_i \prec_S c_j$.

Príklad: Rozvrh S: $s_1 \ldots s_2\ w_1a\ r_2a\ w_2b\ c_2 \ldots c_1$ je konflikt sérivateľný a nie je recoverable.

# Kaskádový rollback

To, že rozvrh je recoverable, ešte nebráni kaskádovému rollbacku.

Definície:

Hovoríme, že rozvrh S sa vyhýba kaskádovému rollbacku (ACR avoids cascading rollback), ak každá transakcia v rozvrhu S číta len hodnoty napísané už commitovanými transakciami.

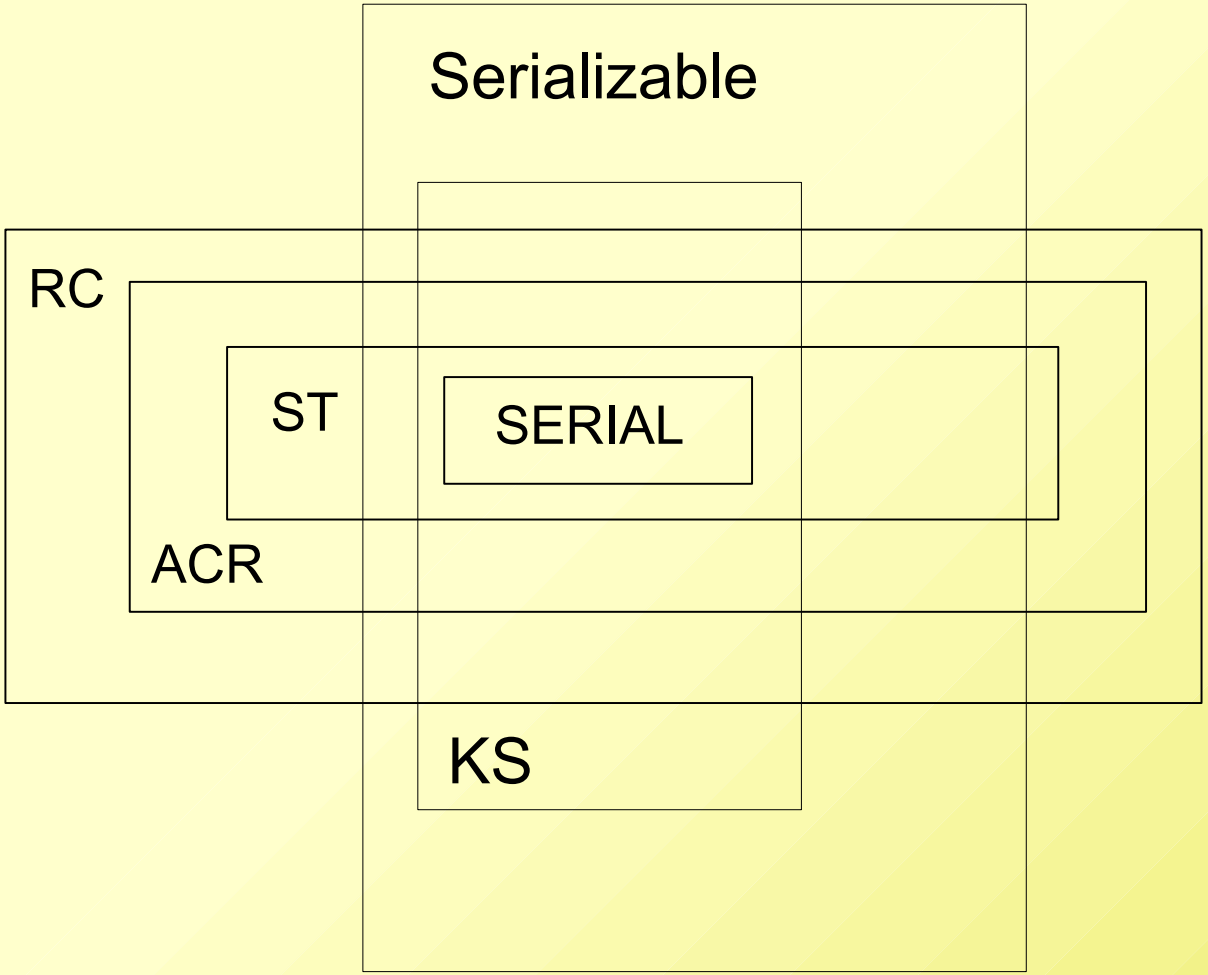
Hovoríme, že je striktný, ak každá transakcia číta a píše len hodnoty napísané už commitovanými transakciami.

Veta: Strict $\subset$ ACR $\subset$ RC. Obe inklúzie sú vlastné.

- Recoverable: $w_1(A)\ w_1(B)\ w_2(A)\ r_2(B)\ c_1\ c_2$
- ACR: $w_1(A)\ w_1(B)\ w_2(A)\ c_1\ r_2(B)\ c_2$
- STRICT: $w_1(A)\ w_1(B)\ c_1\ w_2(A)\ r_2(B)\ c_2$

Prepokladáme, že zápisu $w_2(A)$ nepredchádza čítanie $r_2(A)$.

# Celková situácia



# Izolácia v SQL

Norma SQL definuje štyri úrovne izolácie na základe troch nežiaducích javov, ktoré treba preventovať.

1) Dirty read – transakcia číta dáta z transakcie, ktorá ešte nedosiahla commit.
2) Nonrepeatable read – pri opakovanom čítaní transakcia zistí, že dáta boli modifikované inou už commitovanou transakciou.
3) Phantom read – pri opakovanom vykonaní dotazu, transakcia zistí, že množina riadkov, ktoré mu vyhovujú sa zmenila následkom inej medzičasom commitovanej transakcie.

| Isolation Level | Dirty Read | Nonrepeatable Read | Phantom Read |
|---|---|---|---|
| Read uncommitted | možné | možné | možné |
| Read committed | nemožné | možné | možné |
| Repeatable read | nemožné | nemožné | možné |
| Serializable | nemožné | nemožné | nemožné |

# Príkazy SQL pre prácu s transakciami

<SQL transaction statement> ::=
<start transaction statement>
| <set transaction statement>
| <set constraints mode statement>
| <savepoint statement>
| <release savepoint statement>
| <commit statement>
| <rollback statement>

# SQL 99

<start transaction statement> ::=
**START TRANSACTION** <transaction mode> [ { <comma> <transaction mode> }...]

<transaction mode> ::= <isolation level>
| <transaction access mode>| <diagnostics size>

<transaction access mode> ::= **READ ONLY** | **READ WRITE**

<isolation level> ::= **ISOLATION LEVEL** <level of isolation>

<level of isolation> ::= **READ UNCOMMITTED**
| **READ COMMITTED** | **REPEATABLE READ**
| **SERIALIZABLE**

<diagnostics size> ::=
DIAGNOSTICS SIZE <number of conditions>
<number of conditions> ::= <simple value specification>

# Granularita zámkov

- Čo zamykáme ?
  - tabuľky, riadky, položky, bloky, celú bázu dát
- Zámky na stromových štruktúrach (BVS, B-stromy, … )
  - Ako prvý môžeme zamknúť ľubovolný uzol
  - Každý ďalší zámok môžeme požadovať iba, ak máme zamknutého otca
  - Ak nejaký uzol odomkneme nesmieme ho zamknúť znovu

# Uviaznutie a vyhladovenie I

- Pštrosí prístup: ide o javy vyskytujúce sa zriedka, strčme hlavu do piesku a ignorujme ich. Ak náhodou nastane uviaznutie reštartujeme sytém. Ak operátor zistí, že nejaká transakcia dlho čaká reštartuje ju s väčšou prioritou. Pre databázy nevhodné.
- Vyhýbanie deadlocku teoreticky najstarší prístup založený na bankárovom algoritme. Predpokladá, že každá transakcia dopredu povie všetky nároky na prostriedky a tieto aj dodrží. Plánovač v každom kroku hodnotí, či po prideleni prostriedku vznikne bezpečný stav (t.j. poradie pridelovania prostriedkov transakciam tak, aby všetky mohli skončiť.) Prostriedky sú dáta a vyskytujú sa v jedinom exemplári. Ak vezmeme do úvahy sériovateľnosť to je, že transakcie musia použiť prostriedky v rovnakom poradí, znamená to, že transakcia nemôže dostať žiaden prostriedok, ktorý si nárokuju predchádzajúce transakcie. !?

# Uviaznutie a vyhladovanie II

- Prevencia deadlocku: Dva algoritmy oba uprednostňujú staršie transakcie. Transakcie dostanú časové razítka ts(T).
  - Wait or die – Ak transakcia $T_i$ žiada prostriedok (zámok), ktorý má transakcia $T_j$. Transakcia $T_i$ čaká, ak $ts(T_i) < ts(T_j)$. ($T_i$ je staršia ako $T_j$.) Inak transakcia $T_i$ zomrie (abort a rollback).
  - Wait or wound – Ak transakcia $T_i$ žiada prostriedok (zámok), ktorý má transakcia $T_j$ a $ts(T_i) < ts(T_j)$, $T_i$ „zraní $T_j$“, zranenie je obvykle smrteľné a $T_j$ zomrie a uvolní prostriedky pre $T_i$. Keď $T_j$ už dosiahlo bod commitu a uvolňuje prostriedky zranenie prežije. Ak $ts(T_i) > ts(T_j)$, $T_i$ čaká.
- Ak transakciam pri reštarte ponecháme časové razítka, rieší tento prístup aj starvation.

# Uviaznutie a vyhladovanie III

- Detekcia deadlocku: Deadlockom sa nebránime. Necháme transakciám volný priebeh. Systém periodicky preberá riadenie, zostrojí graf čakania. Ak objavi cyklus, nastal deadlock. System si vyberie jednu z transakcií v cykle (asi najmladšiu) a tú abortuje.
- Graf čakania
  - Uzly sú transakcie
  - Hrana $T_i \rightarrow T_j$, ak $T_j$ čaká na prostriedok, ktorý má pridelená transakcia $T_i$.
- Testovanie cyklu v grafe – topologické triedenie

# Bezpečnosť bázy dát

- Zabespečenie proti zlyhaniu systému
  - Kontrolné body
  - Back up
    - Úplný
    - Difernciálny
- Autorizácia a práva uživateľov
  - Granularita práv
  - Diagram práv
- Grant a Revoke príkaz v SQL

# Kontrolné body – checkpoints

Checkpoint = konzistentný stav bázy dát (stav, čas)

Backup - podobné, ale na náhradnom médiu

- Dočasne zastaví začínanie nových transakcií pokiaľ všetky aktívne transakcie nie sú committed alebo aborted.
- Nájde všetky bloky modifikované v dočasných súboroch a stránky v hlavnej pamäti, ktoré neboli zapísané do databázy.
- Zapamätá v predošlom odstavci nájdené bloky do databázy
- Do journálu (logu) poznamená výskyt checkpointu (dátum, stav, druh checkpointu)
- Potreba čakať, kým neskončia všetky začaté transakcie a nespúšťanie nových transakcií môže nežiadúcim spôsobom znížiť priepustnosť a dostupnosť systému.

# Kontrolné body za prevádzky

- Nič sa zastaví, systém bude pracovať normálne
- Do journalu sa napíše čas, begin checkpoint a zoznam bežiacích transakcií $\mathcal{Z} = (T_1, T_2, \ldots, T_k)$.
- Systém súčasne vykonavá činnosti potrebné pre checkpoint a spracováva transakcie zo $\mathcal{Z}$, prípadne spúšťa nové transakcie. Svoju činnosť zanamenáva do journálu
- Keď skončí výpočet checkpointu a všetky transakci zo zoznamu $\mathcal{Z}$ sú comitované alebo abortované zaznamená do journálu čas a end checkpoint
- Úsek journálu medzi begin checkpoint a end checkpoint je súčasť checkpointu resp. backupu.
- Konzistentný stav v čase end checkpoint dostaneme pomocou redo všetkých akcií transakcií zo zoznamu $\mathcal{Z}$ a undo akcií transakcií začatých po begin checkpoint.

# Uchovávanie backupu

- Vaše dáta musia prežiť požiar, zemetrasenie i bombový útok. Po takejto katastrofe musíte byť schopní najneskôr do 3 dní obnoviť činnosť databázy.
- Budovu a počítač ľahko kúpite, dáta nie.
- Najvhodnejším miestom pre backup je bankový trezor dostatočne vzdialený od Vašej budovy.
- Nikdy neuskladňujte backup vo Vašej počítačovej miestnosti. Ak ho nemôžete dať do banky, odneste si ho domov.
- Ak to prevádzkové pomery dovolujú, snažte sa pravidelne robiť aj úplný systémovy backup
- RAID a cluster sú užitočné hardwareové technológie.
- Aj ukladanie dát na webe sa môže hodiť. Nejaký škandál s únikom informácie Vaša firma prežije, stratu dát nie.

# Model autorizácie

- Operačný systém (napr. Unix), jednoduché
  - Práva: read, write, execute
  - Hierarchia: user, group, world
- Databázový systém, dosť zložité
  - Práva (priviledges):
    - Voči schéme: create, drop, alter
    - Voči tabuľkám: insert, delete, update, query (môžu byť obmedzené na niektoré stĺpce, na skupiny riadkov alebo na hirachickú štruktúru.
  - Role hierachicky usporiadané množiny práv
  - Individuálnym uživateľom môže byť pridelená rola aj individuálne práva
  - Práva voči právam
    - Uživateľ može mať právo postúpiť niektoré  práva iným uživateľom
    - Odbratie práva sa môže kaskádovať

17

# Práva voči tabuľkám

- Najdôležitejšie práva voči reláciám:
  1. SELECT = right to query the relation.
  2. INSERT = right to insert tuples.
     - Môže sa aplikovať na jednotlivé atribúty.
  3. DELETE = right to delete tuples.
  4. UPDATE = right to update tuples.
     - Môže sa aplikovať na jednotlivé atribúty.

# Autorizačné identifikátory

- Na uživateľa odkazuje authorization ID, obvykle názov účtu, „login name“
- Vždy existuje rola PUBLIC.
  - Pridelením práva k roli PUBLIC dostávajú toto právo všetci uživatelia.
  - Inak, každé authorization ID má práva role PUBLIC ako default.
- Aby mohol uživateľ niečo vytvoriť potrebuje mať príslušné právo voči schéme

# Základné pravidlá pridelovania práv

- Každý uživateľ má všetky práva voči objektom (tabuľkám), ktoré sám vytvoril
- Každý uživateľ môže tieto práva postúpiť iným roliam alebo uživateľom včítane role PUBLIC
  - Kedykoľvek pomocou príkazu GRANT
  - Pri vytváraní pomocou grant klauzy v príkaze create
  - Súčasne s právom môže uživateľ odovzdať aj pravo postupovať toto právo iným uživateľom (klauza WITH GRANT OPTION).
- Každý uživateľ môže práva, ktoré pridelil kedykoľvek odobrať pomocou príkazu REVOKE.
  - Príkaz revoke sa obvykle kaskaduje (CASCADE)
  - alebo „vráti chybu“, ak práva boli postupené ďalej (RESTRICT).

# Syntax príkazov GRANT a REVOKE

- Grant príkaz

GRANT <list of privileges>
ON <relation or other object>
TO <list of authorization ID's>
[WITH GRANT OPTION];

Je to veľmi zjednodušené.
Presná syntax je popisaná
v norme pre SQL-99.
Navyše rôzne implementácie
sa často v detailoch odlišujú.

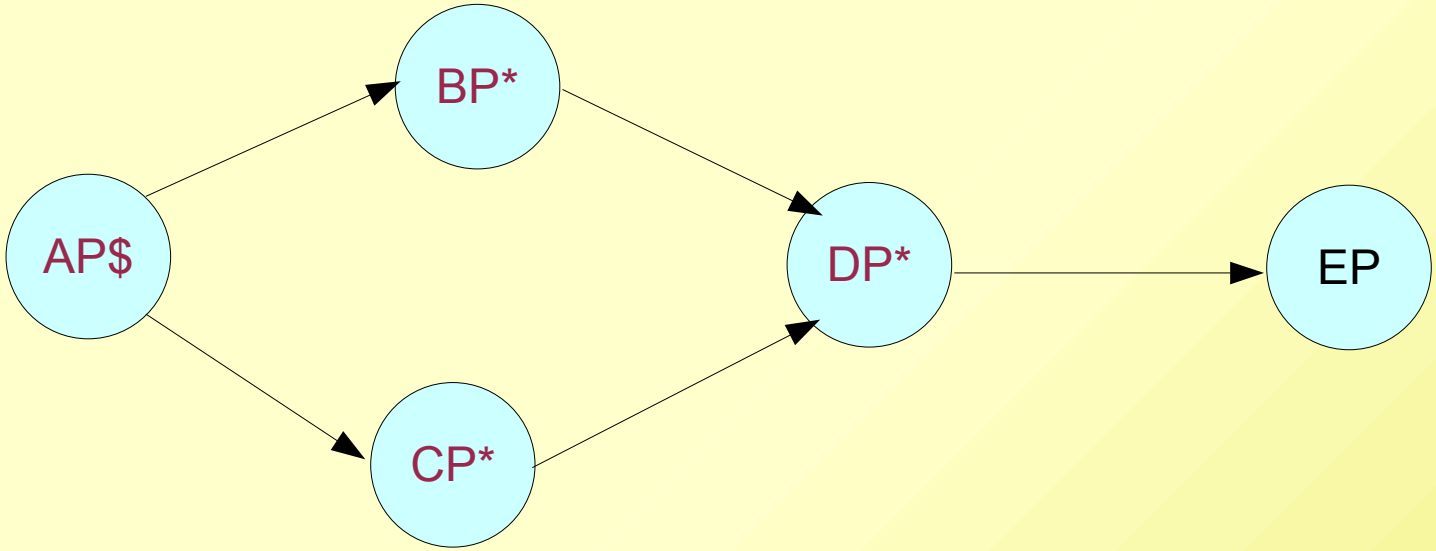
- Revoke príkaz

REVOKE <list of privileges>
ON <relation or other object>
TO <list of authorization ID's>
[CASCADE / RESTRICT];

# Graf (diagram) pridelených práv

- Uzly sú trojice <user, priviledge, option>
  - priviledge P, P* P with grand option, P$ source of P (creator)
- Hrana z X do Y (X → Y) znamená, že Y dostáva pravo P od X
  - X môže udeliť právo P, keď má právo Q, ktoré je nad právom (superpriviledge) P.
- Pravidlá:
  - Ak X má právo Q*/Q$ a P je podprávo (subpriviledge) Q. Potom, keď X pridelí právo P/P* uživateľovi Y, pridáme hranu <X,Q,*/$ > → <Y,P, /$>.
  - Ak X odoberie právo P uživateľovi Y ruší sa „cesta“ začínajúca hranou <X,Q,*/$ > → <Y,P, /$>
  - Uživateľ C má právo P, pokiaľ existuje uzol <C,P,o > a cesta z <X,Q,$>, kde Q je nadprávo P.

# Škaredý príklad




A: Revoke P to B cascade  Čo: C: Revoke P to D cascade

Ukazuje to, že či sa cesta kaskáduje treba testovať v každom uzle.