6. Защита на данните

6.1. Сигурност на данните

Под понятието сигурност да данните се разбира защита на данните от неправомерен достъп, изменение и унищожаване. Следователно СУБД трябва да разрешава изпълнението на операция над БД само ако потребителят има право. Това означава, че сигурността се реализира от СУБД на базата на изисквания формулирани от АБД на езика SQL.

Принципи при осигуряване на сигурност на данните

Кои са основните компоненти в системата за защита на данните в БД?

Потребители. Когато СУБД изпълнява операция над БД тя винаги е от името на определен потребител. Следователно, потребителите са действащите лица в БД или субектите в системата за защита.

Обекти. Елементите, достъпът до които ще се контролира, са обектите в БД. Преди всичко това са таблици – базови и виртуални, и евентуално други типове, като индекси, съхранени процедури.

Привилегии. Правото на потребител да извършва определено действие над обект се нарича привилегия. Кои са привилегиите според стандартите? Обикновено реализираните типове привилегии са: select, insert, update, delete, което означава правото за изпълнение на съответния SQL оператор над таблицата. Има и една важна привилегия – привилегията собственост. Тя е важна защото дава пълни права на потребителя, включително и правото да унищожи обекта, и е първата привилегия, която се дава за обект.

6.1.1. Идентификация на потребители

За да има система на сигурност е необходимо потребителите да бъдат различавани, т.е. да имат имена или идентификатори. В стандартите се използва термина идентификатор на права на достъп (authorization-id), но повечето СУБД използват понятието идентификатор на потребител (user-id). Някои системи използват собствено пространство от имена на потребители, като Oracle. Други като Informix, Ingres, Postgres използват имената на потребителите, регистрирани в операционната система.

Идентификацията на потребител за достъп до БД включва обикновено същите стъпки, както във всяка многопотребителска ОС. Най напред потребителят задава името си, а след това и паролата за да докаже, че е този за когото се представя. Кога се извършва идентификацията на потребител? При интерактивен SQL обикновено това става в началото на сесията.

Например в Oracle стартирането на програмата за интерактивна работа с SQL може да изглежда така:

```
sqlplus scott/tiger
```

Като аргументи се задават името и паролата на потребителя. В Oracle има още два оператора, които могат да се изпълняват след като е стартирана програмата `sqlplus`, а именно:

```
disconnect
connect scott/tiger
```

Чрез `disconnect` се завършва текущата сесия, а с `connect` се започва нова сесия.

В Informix, Ingres и Postgres се разчита на текущия потребителски идентификатор от ОС, т.е. предполага се, че потребителят, който в момента работи в ОС започва и сесия с БД. Затова там при свързване с БД не е необходимо да се задава име и парола. Спомнете си SQL опереторите:

```
DATABASE  име-на-бд
```

```
CLOSE DATABASE
```

Но е възможно в рамките на една сесия на ОС да се започне сесия с БД от името на друг потребител. Има и оператори:

```
DISCONNECT CURRENT
CONNECT TO 'име-на-бд' USER 'име-на-потребител'
```

Информация за потребителите, имащи право на достъп до БД се съхранява в системна таблица. В Informix това е таблицата sysusers.

6.1.2. Категории потребители

Всеки потребител, който има право на достъп до БД, принадлежи към една от трите категории – `CONNECT, RESOURCE` или `DBA`, което определя неговите права на ниво цялата БД. Какви са привилегиите на всяка от категориите?

`CONNECT`

- Да изпълнява операторите `select, insert, update, delete` над таблици, за които са му дадени тези привилегии.
- Да създава временни таблици, индекси над тях и да ги унищожава.
- Да създава виртуални таблици над таблици, за които има привилегия `select`.
- Да създава синоними.

`RESOURCE`

- Всички привилегии на `CONNECT`.
- Да създава нови таблици, индекси над тях и да ги унищожава.
- Да изменя структурата на таблици, за които има тази привилегия.

`DBA`

- Всички привилегии на `RESOURCE`.
- Да унижожава всякакви обекти в БД.
- Да унижожи БД.
- Да предоставя и отнема привилегии на потребители.

Следващият въпрос е как на потребител се дава право на достъп до БД в определена категория. В Informix потребителят, който създаде БД става неин собственик и е първият потребител с привилегия `DBA`. Първоначално само той има право на достъп до БД. Той може да регистрира други потребители и да им даде привилегия на ниво БД. Операторите, с които само потребител `DBA` предоставя и отнема привилегии на ниво БД са:

```
GRANT {CONNECT|RESOURCE|DBA} TO {PUBLIC|списък-имена-потребители }

REVOKE {CONNECT|RESOURCE|DBA} FROM {PUBLIC|списък-имена-потребители }
```

Има една специална ключова дума `PUBLIC` вместо име на потребител, която означава всички потребители в момента и в бъдеще, т.е. всички потребители, които са регистрирани в ОС в момента на проверка на привилегиите.

Поради йерархичната организация на привилегиите, ако се предостави право `DBA` на потребител, не е необходимо да му се дават и права `CONNECT` и `RESOURCE`. Ако се отнеме `CONNECT` или `RESOURCE` от `DBA` това няма никакъв ефект. Ако се отнеме `RESOURCE` или `DBA` от потребител, той се понижава до `CONNECT`. За да се отнеме всяко право на достъп до БД от потребител, трябва да му се отнеме по-високата привилегия, ако има такава, след което да му се отнеме и `CONNECT`.

В Oracle категориите потребители са същите, операторите също, но е различно правилото, по което се определя първия `DBA` потребител. При създаване на БД автоматично се създава първият `DBA` потребител с име `system` и парола `manager`. Той след това може да регистрира други потребители от различни категории.

6.1.3. Привилегии на ниво таблици

Потребителят, създал таблица, става неин собственик. Привилегията собственост не може да се отнема или да се предава на друг потребител. Според

ANSI стандарта, първоначално само той има пълни права над таблицата, включително и да я унищожи. Само собственикът или DBA потребител може да дава на други потребители привилегии. Какви са привилегиите за таблици в Informix?

```
SELECT [(списък-имена-на-колони)]
UPDATE[(списък-имена-на-колони)]
INSERT
DELETE
INDEX
ALTER
REFERENCES[(списък-имена-на-колони)]
ALL [PRIVILEGES]
```

В Informix има възможност БД да се създаде не по ANSI стандарта. Тогава при създаване на таблицата системата автоматично предоставя на всички потребители (PUBLIC) всички привилегии без ALTER и REFERENCES.

Предоставяне на привилегии за таблица

Операторът, с който собственикът на таблица или DBA потребител може да предостави на други потребители привилегии е:

```
GRANT списък-от-привилегии ON име-на-таблица
TO {PUBLIC|списък-имена-потребители }
[WITH GRANT OPTION] [AS име-на-потребител ]
```

Таблицата *име-на-таблица* може да е базова, виртуална таблица или синоним. Фразата WITH GRANT OPTION означава, че потребителя получава указаните привилегии заедно с правото да ги предава на други потребители. Фразата AS може да се използва само от DBA потребител. Чрез нея той предоставя привилегиите от името на друг потребител *име-на-потребител*.

Привилегиите ALTER и INDEX могат да се дават само на RESOURCE потребител, тъй като предимство имат по-ограничителните привилегии.

Отнемане на привилегии за таблица

Динамично в процеса на работа с БД привилегиите могат както да се предоставят, така и да се отнемат. Собственикът на таблица или DBA потребител може да отнеме всички или част от привилегиите, дадени на потребител с оператора:

```
REVOKE списък-от-привилегии ON име-на-таблица
FROM {PUBLIC|списък-имена-потребители }
```

Ако са дадени привилегии на PUBLIC то се отнемат от PUBLIC. Ако са дадени SELECT(*списък-имена-на-колони*) или UPDATE(*списък-имена-на-колони*) не могат да се отнемат правата за част от колоните. Ако се отнеме привилегия от потребител, предоставена му с WITH GRANT OPTION, то автоматично се отнемат и всички разпространени от него.

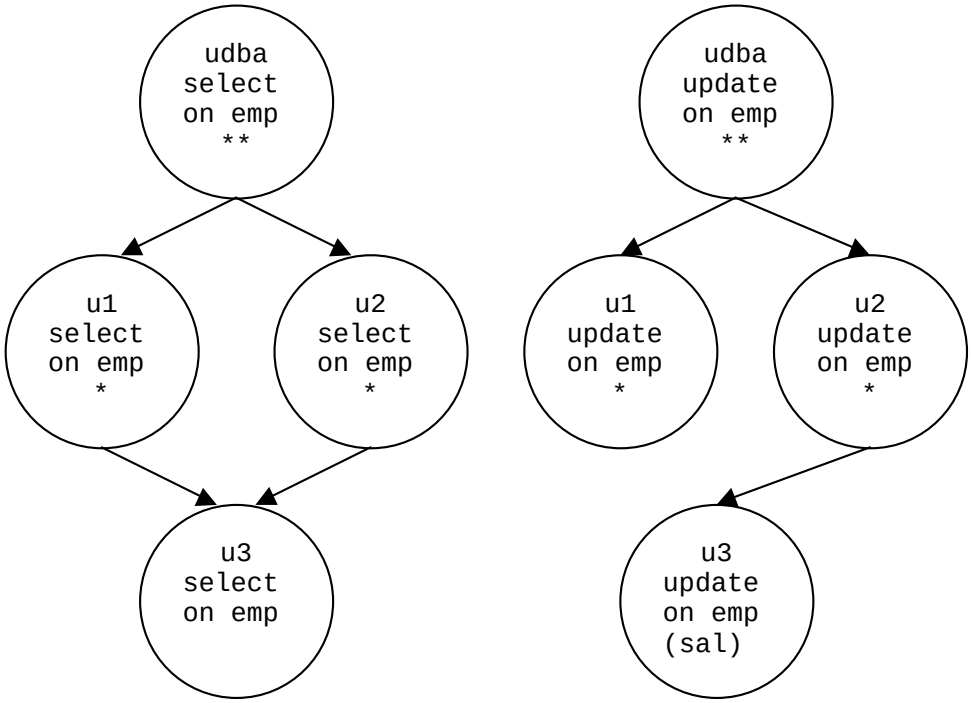
Диаграма на привилегиите

В резултат на изпълнение на операторите GRANT и REVOKE може да се получи сложна система от частично препокриващи се привилегии. За по-доброто разбиране е полезно да се представи процеса на предоставяне и отнемане на привилегии чрез диаграма. Тази диаграма е ориентиран граф, в който всеки връх представя комбинацията “потребител/привилегия”. Ребро от връх “U/P” към връх “V/Q” означава, че потребител U е предоставил на потребител V привилегия Q, която се базира на привилегията P. Символите ** представят привилегията собственост. Един символ * означава правото да се предава привилегията.

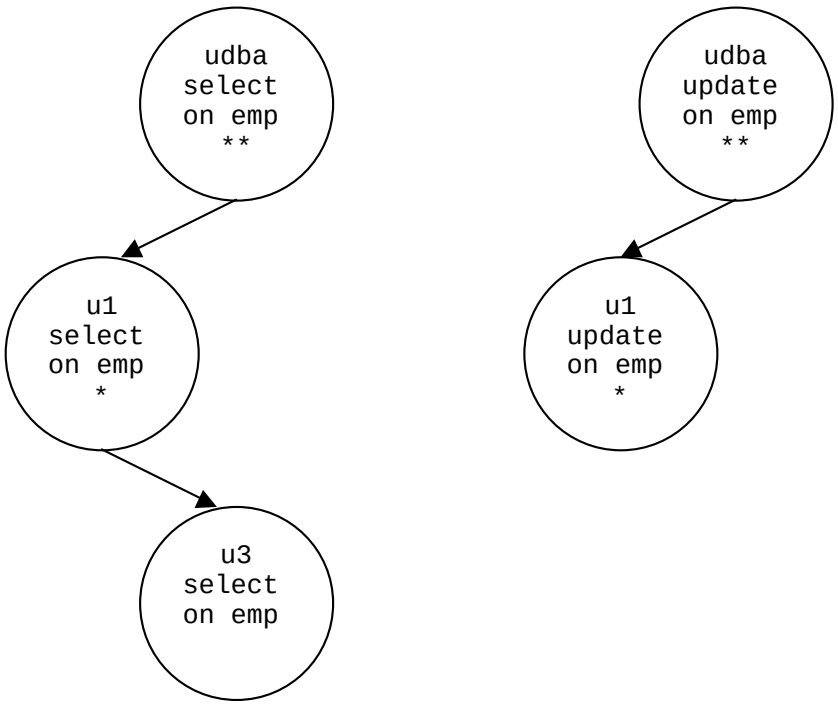
♦ Нека таблицата `emp` е собственост на `udba`, който е `DBA`, а `u1`, `u2` и `u3` са потребители, регистрирани в БД като `CONNECT`. Ако са изпълнени следните оператори в тази последователност.

Стъпка	Потребител	Действие
1	udba	`GRANT SELECT, UPDATE ON emp TO u1, u2 WITH GRANT OPTION;`
2	u1	`GRANT SELECT ON emp TO u3;`
3	u2	`GRANT SELECT, UPDATE(sal) ON emp TO u3;`
4	udba	`REVOKE SELECT, UPDATE ON emp FROM u2;`

След изпълнение на стъпки 1-3 диаграмата ще има следния вид.



След изпълнение на стъпка 4 диаграмата ще има вида.



Виртуални таблици и осигуряване на сигурност

Виртуалните таблици могат да се използват за да се ограничи достъпа на потребител до част от данните в таблица или само до производни данни, но не и до детайлните данни.

♦ На потребител `u1` да се даде достъп само до данните за отдел 1. Потребителят udba, който е собственик на таблицата `emp`, създава виртуална таблица, съдържаща пълните данни за служителите от отдел с номер 1. След това предоставя привилегии на `u1` само към виртуалната таблица.

```
CREATE VIEW vemp1
AS SELECT * FROM emp WHERE dno = 1
WITH CHECK OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON vemp1 TO u1;
```

Чрез виртуална таблица потребител не може да получи неконтролирано допълнителни привилегии. Потребител, който създава виртуална таблица трябва да има `SELECT` права за всички колони от всички таблици, участващи в дефиницията. Той става неин собственик и получава право `SELECT`. Ако потребителят има права `INSERT, UPDATE, DELETE` над базовите таблици, участващи в дефиницията, и ако виртуалната таблица е обновяема, то той автоматично получава съответните права и над нея. Ако привилегиите за базовите таблици са му дадени `WITH GRANT OPTION` или е техен собственик, то може да предава правата над виртуалната таблица.

6.2. Цялостност на данните

Понятието цялостност на данните означава защита на данните от неправилно изменение и унищожаване. Целта е осигуряване на правдоподобни (ако може и правилни), непротиворечиви данните в БД.

Причини за нарушаване целостността на данните

- Апаратни грешки
- Програмни грешки в ОС, СУБД, приложната програма
- Външни причини – токов удар, земетресение, наводнение
- Грешки на потребителя
- Конкурентен достъп до данните – използване на едни и същи данни по едно и също време от много потребители.

Тъй като причините, които могат да доведат до нарушаване на цялостността, са най-различни, то и механизмите, реализирани в СУБД са различни. Един от механизмите, който се реализира във всяка СУБД, са така наречените ограничения за цялостност. Това са условия, формулирани от АБД на езика SQL, на които трябва да отговарят данните за да се считат за правилни.

6.2.1. Ограничения за цялостност

Ще разгледаме видовете ограничения за цялостност и как те се поддържат в езика SQL и от СУБД. В повечето случаи, ако ограничението се поддържа, то това е в операторите `CREATE TABLE/ALTER TABLE` или `CREATE INDEX`.

Ограничения за значението на колона

- **Тип на данните**
  Всички значения в една колона са от определен тип на данните.
- **Определеност (задължителност) на значенията** на колона – **`NOT NULL`**
  Забранява редове, в които колоната е със значение `NULL`.
- **Условие за проверка** върху значенията на колона

Изброяват се допустимите значения в колоната или се задава интервал(и) на допустимите значения, или шаблон, на който да отговарят значенията. Този вид ограничения могат да се изразят чрез фразата CHECK, която е включена в стандартите и се реализира в повечето СУБД.

♦ Колоната spec в student може да приема само значенията Math, PM, Inf, Math&Inf.

```
CREATE TABLE student (...,
spec CHAR(10) CHECK ( spec IN ( 'Math','PM','Inf','Inf&Math')),...);
```

♦ Заплатата на всеки служител трябва да е по-голяма или равна на 0.

```
CREATE TABLE emp (...,
sal MONEY CHECK ( sal >= 0 ),...);
```

♦ Колоната ocenka в s_p може да приема значения в затворения интервал от 2 до 6 или NULL.

```
CREATE TABLE s_p (...,
ocenka DECIMAL(4,2)
      CHECK((ocenka BETWEEN 2 AND 6) OR ocenka IS NULL),...);
```

Ограниченията от тези видове се проверяват при изпълнението всеки оператор insert или update, когато колоната получава ново значение.

Ограничение за възможен ключ - UNIQUE

Това е ограничение за уникалност на значенията на колона(и) в редовете на една таблица, което забранява съществуването на няколко реда в таблицата с еднакви значения в тези колони. Всяка таблица може да има няколко възможни ключа.

Ограничението се проверява при изпълнението всеки оператор insert или update, изменящ значението на възможния ключ. СУБД трябва да провери дали в таблицата не съществува друг ред със същото (новото) значение на възможния ключ. Тази проверка значително ще се ускори, ако съществува индекс по възможния ключ, тъй като търсенето ще се извърши в индекса, а не в таблицата. Следователно индексът има важно значение при прилагането на това ограничение, затова някои СУБД автоматично създават индекс за всеки възможен ключ.

Този вид ограничение се задава в оператора CREATE TABLE/ALTER TABLE, но в някои СУБД може и чрез оператора CREATE UNIQUE INDEX

Ограничение за първичен ключ – PRIMARY KEY (Entity integrity)

Всяка таблица може да има само един първичен ключ. Ограниченията за първичен и възможен ключ са идентични (според теорията първичният ключ е възможен ключ, който е избран), но някои СУБД им дават леко различен смисъл. Напр. в Informix, за първичния ключ се предполага и ограничение NOT NULL, а за възможния не се предполага това ограничение. Но и за двата вида ключа се създава автоматично индекс.

Ограничение за външен ключ – FOREIGN KEY (Referential integrity)

Всеки външен ключ в една таблица е свързан с първичен (възможен) ключ на друга таблица, на който съответства. Този вид ограничение засяга данните в две таблици, затова реализацията му е по-сложна. При прилагането на това ограничение от СУБД има три варианта на действие или три правила.

- **ограничително правило** (RESTRICT)

Това е правилото по премълчаване. Отхвърля се обновяването на БД, ако то нарушава ограничението. Това е единственото правило, което се прилага при оператор insert в таблицата с външния ключ и при update на външния ключ.

Може да се прилага при delete за таблицата с първичния ключ или update на първичния ключ.

- **каскадно правило** (CASCADE)

Изпълнява се исканото обновяване на БД и ако се наруши ограничението, съответният обновяващ оператор се прилага и върху таблицата, съдържаща външния ключ. Може да се прилага при delete за таблицата с първичния ключ или update на първичния ключ.

- **правило с присвояване на неопределено значение** (SET NULL)

Изпълнява се исканото обновяване на БД и ако се наруши ограничението, на външния ключ в съответните редове се присвоява NULL. Може да се прилага при delete за таблицата с първичния ключ или update на първичния ключ.

Последните две правила могат да се прилагат независимо по отношение на операторите delete и update.

♦ При изтриване на отдел от dep, за служителите от изтрития отдел в emp да се присвои NULL на dno. При изменение на номера на отдел dno в dep, да се измени по същия начин и външния ключ dno в emp .

```
CREATE TABLE emp (..., dno SMALLINT,
FOREIGN KEY (dno) REFERENCES dep(dno)
                  ON DELETE SET NULL
                  ON UPDATE CASCADE );
```

Условия върху значенията на колони в един ред

Това е условие, което може да бъде проверено върху значенията в един ред на таблица, разглеждайки го независимо. Този вид ограничения могат да се изразят чрез фразата CHECK.

♦ Хорариумът на всеки предмет не може да е 0, т.е. сумата от брой лекции и брой упражнения трябва да е по-голяма от 0.

```
CREATE TABLE predmet (...,
CHECK ( br_l + br_u > 0 ), ...);
```

♦ Ако специалността на студент е PM, то първата цифра от фак. номер да е 3, ако специалността на студент е Math, то първата цифра от фак. номер да е 1, и.т.н.

```
CREATE TABLE student (fn CHAR(6),..., spec CHAR(10), ...,
CHECK ((spec='PM' AND  fn LIKE '3%') OR
       ((spec='Math' OR spec='Math&Inf') AND  fn LIKE '1%') OR
       (spec='Inf' AND  fn LIKE '4%') OR
       fn[1]='M'),...);
```

Статистически ограничения

Това са условия върху значенията в цяла таблица или в няколко таблици. В стандарта SQL2 в ограничение CHECK може да има сложни условия с вложен оператор select.

♦ Средната заплата да е по-малка от 1000.

```
CREATE TABLE emp (...,
CHECK ( 1000 > (SELECT AVG(sal) FROM emp)),...);
```

♦ Сумата от заплатите на служителите във всеки отдел да е по-малка от бюджета на отдела.

```
CREATE TABLE emp (...,
CHECK ( dep.budjet > (SELECT SUM(sal) FROM emp WHERE dno=dep.dno)),...);
```

Динамични ограничения

Това са условия, които сравняват старото и новото съдържание на БД. Този вид ограничения не могат да се изразят директно в оператор `CREATE TABLE.`

♦ Новата заплата на служител не може да е два пъти по-голяма от старата му заплата.

Ограниченията за цялостност се прилагат при изменение на обектите в БД, към които се отнасят, т.е. проверката им се активизира при настъпване на определени събития в БД. Затова ги наричат **активни елементи** в БД. Напр. ограничението за колона `CHECK` се проверява при изменение на колоната в някой ред (при изпълнение на оператори `update` или `insert`). В релационните БД има и други видове активни елементи - тригери.

6.2.2. Тригери

През 1986 в Sybase е въведено понятието тригер (trigger), което представлява следваща стъпка напред към включването на по-общи ограничения (бизнес правила) в БД. Тригерите са включени и в стандарта SQL3 и се реализират в много СУБД. Тригерите са активни елементи от типа "събитие-условие-действие" (ECA).

1. Тригер се прилага при настъпване на **събитие** (trigger event), определено от програмиста.

2. Тогава се проверява зададено **условие** и ако то не е изпълнено, не се изпълнява никакво действие.

3. Ако условието е истина, СУБД изпълнява **действието** (trigger action), свързано с тригера.

Следващите примери са с използване на тригери в Informix.

Там събитията, с които може да бъде свързан един тригер, т.е които ще го активизират, са изпълнение на един от следните SQL оператори над определена таблица:

```
INSERT ON таблица
DELETE ON таблица
UPDATE [OF  колона [, ...]] ON таблица
```

Действието е последователност от SQL операторите `INSERT`, `DELETE`, `UPDATE` и `EXECUTE PROCEDURE` (извиква за изпълнение съхранена процедура). Действието може да се изпълни:

`BEFORE` - преди изпълнение на оператора, активизирал тригера;

`AFTER` - след завършване на оператора, активизирал тригера;

`FOR EACH ROW` - след изменението на всеки ред от оператора, активизирал тригера.

За един тригер може да има действие `BEFORE`, `FOR EACH ROW` и `AFTER`, но в този ред. Всяка такава част от действието може да включва условие (фраза `WHEN`).

Ако действието се изпълнява `FOR EACH ROW`, то в него може да използват както старите, така и новите значения в редовете, изменяни от оператора, активизирал тригера (фраза `REFERENCING`).

♦ При добавяне на нов служител (въвеждане на ред в `emp`) да се увеличи бюджета на съответния отдел със заплатата му.

```
CREATE TRIGGER tr1
INSERT ON emp
REFERENCING NEW AS post
FOR EACH ROW
(UPDATE dep SET budjet=budjet+post.sal WHERE dno=post.dno);
```

♦ При изменение на заплатата на служител, ако новата заплата е два пъти по-голяма от стара, това да се сигнализира като се добави ред в таблица `warn_tab,` следяща за такива нарушения.

```
CREATE TRIGGER tr2
UPDATE OF sal ON emp
REFERENCING OLD AS pre NEW AS post
FOR EACH ROW WHEN ( post.sal > pre.sal*2 )
(INSERT INTO warn_tab VALUES(pre.eno, pre.sal, post.sal, USER, TODAY));
```

При БД с транзакции проверката на всички ограничения за цялостност (зададени в `CREATE TABLE`) се отлага след след завършване на действието на тригера. Това позволява използването на тригери за поддържане на различните правила за външен ключ (ако не се поддържат в `CREATE TABLE`) или по-общи бизнес правила.

♦ За външния ключ `dno` в таблица `emp` да се реализира каскадно правило при `UPDATE` и правило `SET NULL` при `DELETE`.

```
CREATE TABLE emp (...,
FOREIGN KEY (dno) REFERENCES dep(dno));
```

```
CREATE TRIGGER del_dep
DELETE ON dep
REFERENCING OLD AS pre_del
FOR EACH ROW (UPDATE emp SET dno = NULL WHERE dno = pre_del.dno);
```

```
CREATE TRIGGER upd_dep_dno
UPDATE OF dno ON dep
REFERENCING OLD AS pre_upd NEW AS post_upd
FOR EACH ROW
(UPDATE emp SET dno=post_upd.dno WHERE dno = pre_upd.dno);
```

♦ След изменение на заплатата на служители, ако новата средна заплата е по-голяма от 1000, то обновяването да се отмени.

```
CREATE TRIGGER tr3
UPDATE OF sal ON emp
AFTER (EXECUTE PROCEDURE upd_sal());
```

След завършване на оператора `UPDATE` се извиква процедурата `upd_sal`. Тя проверява дали не е нарушено ограничението за средната заплата и ако е отменя всички направени изменения.

```
CREATE PROCEDURE upd_sal()
DEFINE new_avg MONEY;
LET new_sal = (SELECT AVG(sal) FROM emp);
IF new_sal > 1000
      RAISE EXCEPTION –746, 0, 'Too big average salary';
END IF;
END PROCEDURE;
```

6.3. Транзакции и управление на конкурентния достъп

Транзакцията е логическа единица работа над БД, която е:

- **атомарна** (неделима) от гледна точка на предметната област, но
- включва обикновено **няколко операции** над БД,
- които преобразуват БД от едно непротиворечиво състояние в друго **непротиворечиво състояние**.

Операциите над БД могат да са изразени чрез един или няколко SQL оператора. Напр. увеличаване заплатата на всички служители с 10% (един `update`), или изтриване на служител от БД (`delete` от `emp_pro`, `delete` от `emp` и `update` в `dep` на `mgr`), но за предприятието това е едно неделимо действие. Ако състоянието на БД е непротиворечиво преди транзакцията, то ще е непротиворечиво и след нея, но това не се гарантира в междинните точки (между операциите).

Атомарността на транзакцията означава, че СУБД трябва да осигури успешното изпълнение на всички операции в транзакцията или нито една, каквото и да се случи – грешка при изпълнение на оператор, срив на системата. Затова атомарността я наричат принцип “all or nothing”.

6.3.1. Модели на транзакциите

Как програмистът отбелязва къде започва и къде свършва една транзакция? Има няколко SQL оператора за тази цел и различни правила за използването им.

`BEGIN WORK`

Отбелязва началото на транзакция.

`COMMIT WORK`

Отбелязва успешен край на транзакция. СУБД потвърждава всички изменения в БД от началото на транзакцията, т.е. те стават трайни.

`ROLLBACK WORK`

Отбелязва неуспешен край на транзакция. СУБД трябва да отмени всички изменения в БД от началото на транзакцията.

- **Модел на ANSI**

Транзакцията започва автоматично с първия SQL оператор или след края на предходната транзакция (след `COMMIT WORK`, `ROLLBACK WORK`). Тук няма оператор `BEGIN WORK.`

- **Модел на Informix**

Транзакцията започва с оператор `BEGIN WORK`. Ако не се използват операторите `BEGIN WORK`, `COMMIT WORK`, `ROLLBACK WORK`, то се поддържат еднооператорни транзакции. Всеки SQL оператор е транзакция, т.е. СУБД гарантира неговата атомарност.

По отношение на поддържане на транзакциите в Informix има три възможности, които се избират независимо за всяка БД при създаването й.

- БД без транзакции

`CREATE DATABASE` *`име-на-бд`*

- БД с транзакции по модела на ANSI

`CREATE DATABASE` *`име-на-бд`* `WITH LOG MODE ANSI`

- БД с транзакции по модела на Informix

`CREATE DATABASE` *`име-на-бд`* `WITH [BUFFERED] LOG`

6.3.2. Журнал на транзакциите (Transaction Log)

Как СУБД реализира транзакциите? Как осигурява атомарността дори ако преди края й стане срив в системата, т.е. транзакцията остане незавършена? Как

осигурява отмяната на измененията при `ROLLBACK`. Историята на всички изменения над БД по време на транзакциите се съхранява в специално място, наречено Transaction Log. В Informix това са така наречените logical log страници в основната област. В Oracle се наричат rollback сегменти, които са в някоя от областите.

При изменение на БД се прилага **WAL протокол** (Write Ahead Logging). Преди да се извърши каквото и да е изменение в БД, съответен log запис, който включва старото и новото значение на реда, се записва на диска. След това се изменя и реда в таблицата на БД. При `COMMIT` се записва log запис за успешен край на транзакцията. При `ROLLBACK` се извличат от log записите оригиналните значения на редовете и се връщат в БД, след което се записва log запис за неуспешен край на транзакцията.

След системен срив ефекта от частично изпълнените транзакции се отменя автоматично когато СУБД отново се стартира (процедура recovery). Тогава СУБД изпълнява автоматично `ROLLBACK` за всички незавършили транзакции. Това е възможно благодарение на WAL протокола (ако няма log запис, то и изменението в БД не е извършено).

Използването на транзакции увеличава значително времето за изменения в БД. С цел по-добра производителност, а и по-висока надеждност, е добре Transaction Log и БД да са на различни дискове. Някои СУБД позволяват да се работи и без транзакции, което е приемливо при учебни и малки БД.

6.3.3. Управление на конкурентния достъп

В общия случай няколко приложни програми (следователно транзакции) работят едновременно с БД. Докато една транзакция изменя БД, други също могат да четат и изменят същите данни. Сега СУБД не само трябва да се грижи за атомарността на транзакциите, но и да гарантира коректно взаимодействие на конкурентните транзакции, т.е. че те няма да си пречат. В противен случай са възможни нежелателни проблеми, като загуба на изменения, достъп до междинни (несъществуващи) или несъгласувани данни.

Загуба на изменения (lost update problem)

Транзакция A	Време	Транзакция B
`select t`	t1	
	t2	`select t`
`update t`	t3	
	t4	`update t`

Изменението на транзакция A се губи.

Достъп до междинни данни (uncommited dependency problem)

Транзакция A	Време	Транзакция B
	t1	`update t`
`select t`	t2	
	t3	`rollback work`

Транзакцията A чете данни от непотвърдената конкурентна транзакция B, която впоследствие е отменена, т.е. вижда несъществуващи данни. Ако t1 се припокрие частично с t2, то транзакция A ще прочете смесица от променени и непроменени редове.

Достъп до несъгласувани данни

Транзакция A	Време	Транзакция B
`select t`	t1	

	t2	`update t`
	t3	`commit work`
`select t`	t4	

Транзакцията A чете данни от потвърдената конкурентна транзакция, но въпреки това от нейна гледна точка състоянието на БД е несъгласувано. В рамките на транзакцията A едни и същи редове съдържат различни данни.

Подобен проблем се получава и ако вместо `update` конкурентната транзакция изпълни `insert into t`. Тогава при повторния `select` транзакцията A ще види редове-призраци, които преди не са съществували.

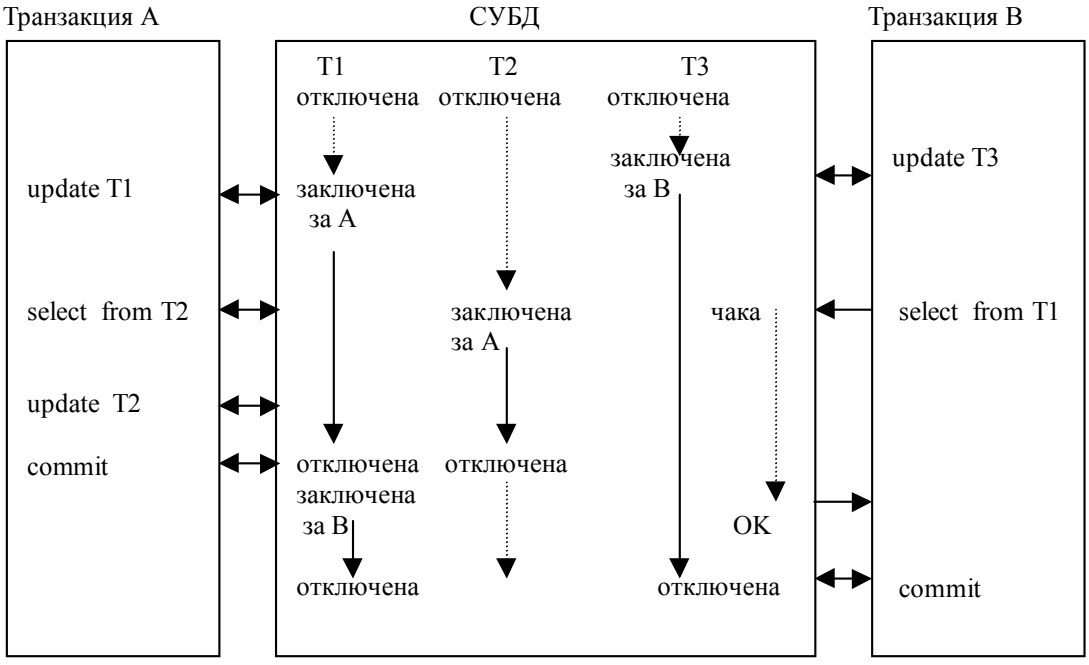
Следователно, за да се осигури цялостност на данните в условията на конкурентно изпълнявани транзакции, е необходим механизъм за управление на конкурентния достъп (Concurrency Control Mechanism). Той трябва да осигури изпълнението на конкурентни транзакции при следното условие:

Когато две транзакции A и B се изпълняват едновременно, СУБД гарантира, че резултатите от тях ще бъдат същите както и ако транзакциите са изпълнени последователно: (а) първо се изпълни A след това B или (б) първо се изпълни B след това A.

Този принцип се нарича **сериализиране** на транзакциите. На практика това означава, че всеки потребител работи с БД така, като че ли няма други потребители, т.е. осигурена е пълна изолация на потребителите един от друг.

Най-популярният механизъм се състои в **заключване/блокиране** на достъпа до част от БД, и е известен като locking. СУБД разполага с набор от ключалки (locks). Преди достъп до обект в БД транзакцията трябва да получи ключалка за него, като може да се наложи да чака докато системата е в състояние да й даде. Когато транзакция T_i получи ключалка за обект (обектът е заключен от T_i), то СУБД гарантира, че никоя друга конкурентна транзакция T_j няма да изменя обекта, докато T_i не го отключи. Освобождаването на ключалките става обикновено в края на транзакцията. Заключване се прилага дори когато в БД не се поддържат транзакции, тогава има конкурентни приложни програми вместо транзакции, но правилата за получаване и освобождаване на ключалка са по-различни.

Един прост пример, илюстриращ използването на блокиране.



В промишлените СУБД се реализира по-сложен механизъм на блокиране с цел повишаване на степента на паралелизъм в системата.

Видове заключване

Обикновено се реализират няколко вида ключалки. Основните са:

SHARE (read)

Заключва обекта за четене, като СУБД не позволява изменението му от други конкурентни транзакции. Това означава, че няколко транзакции могат да получат ключалка SHARE за един обект по едно и също време.

EXCLUSIVE (write)

Заключва обекта за монополно ползване. Когато една транзакция получи ключалка EXCLUSIVE за обект, то никоя друга транзакция не може да получи каквато и да е ключалка за този обект. Използва се когато транзакцията изменя обекта.

Област на заключване (lock granularity)

Какви са обектите в БД, които се заключват с ключалки или каква е областта на действие на една ключалка? Това може да е:

- цялата БД
- цяла таблица
- ред от таблица или страница.

Цяла БД в Informix се заключва с една ключалка при отваряне. Видът на ключалката зависи от оператора DATABASE.

DATABASE *име-на-бд*

Заключва БД с SHARE ключалка. Докато не се освободи ключалката, БД не може да бъде унищожена от друга програма. Отключването става при CLOSE DATABASE.

DATABASE *име-на-бд* EXCLUSIVE

Заключва БД с EXCLUSIVE ключалка. Програмата получава монополен достъп до БД. Операторът се изпълнява успешно ако никоя друга програма не е отворила БД в момента. След успешното изпълнение на оператора, никоя друга програма не може да отвори БД, докато тя не се отключи.

Цяла таблица може да се заключи с една ключалка. Това се извършва автоматично от СУБД с EXCLUSIVE ключалка, при изпълнение на операторите: ALTER TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX. При завършване на оператора таблицата се отключва. Цяла таблица може да се заключи с една ключалка и явно чрез оператор:

LOCK TABLE *име-на-таблица* IN {SHARE|EXCLUSIVE} MODE

В БД с транзакции операторът трябва да се изпълнява в транзакция.

Отключването на таблица в БД без транзакции се извършва при затваряне на БД или с оператора:

UNLOCK TABLE *име-на-таблица*

В БД с транзакции всички ключалки, включително и за таблица, се освобождават в края на транзакцията (COMMIT/ROLLBACK).

Ред от таблица (или страницата, в която е реда) може да се заключи с една ключалка. Ключалка за ред се получава и освобждава само автоматично. Правилата зависят от изпълнявания SQL оператор, дали се поддържат транзакции в БД и от нивото на изолация.

При изпълнение на INSERT, UPDATE, DELETE винаги трябва да се получи EXCLUSIVE ключалка за всеки изменян ред, освен ако цялата таблица не е заключена вече. Кога ще се освободят така получените ключалки зависи от това дали се поддържат или не транзакции в БД. Ако БД е с транзакции, всички ключалки се

освобождават в края на транзакцията. Ако БД е без транзакции, освобождаването на заключения ред става след като той се запише на диска. Дали ще се заключва ред от таблица или страницата, в която е той, се определя при създаването на таблицата или може да бъде променено впоследствие с операторите:

```
CREATE TABLE име-на-таблица ( ... ) LOCK MODE ({PAGE | ROW})

ALTER TABLE име-на-таблица LOCK MODE ({PAGE | ROW})
```

Нивото на изолация определя как се използват ключалки при изпълнение на оператор SELECT.

Нива на изолация (Isolation levels)

Според строгото определение за транзакция – принципа за сериализиране, нито една конкурентна транзакция не може да повлияе на данните виждани от друга транзакция. Тази абсолютна изолация на една транзакция от другите изисква доста ресурси от СУБД. В някои случаи усилията по блокиране могат да се съкратят, ако СУБД е информирана предварително за действията на транзакцията. Затова е въведено понятието ниво на изолация, което дава възможност на програмиста да постигне компромис между степента на изолация и ефективността на работа. Нивото на изолация определя степента, до която четяща програма е изолирана от конкурентните действия на други програми, изменящи БД. В ANSI стандарта и в Informix са реализирани няколко нива, при които действат различни правила за използване на ключалки при четене от БД.

READ UNCOMMITTED / DIRTY READ

При изпълнение на SELECT оператор редовете се четат, независимо дали са заключени от друг, и не се заключват. Това означава, че може да се прочетат редове от непотвърдени транзакции или несъгласувани данни. Не се осигурява никаква изолация на четящата програма. Предимството е ефективност – четящата програма никога не чака и не кара други да чакат. Това е единственото ниво в БД без транзакции.

READ COMMITTED / COMMITTED READ

Гарантира се, че всеки ред извлечен от SELECT е от потвърдена транзакция, т.е.извлича ред ако не е заключен с EXCLUSIVE ключалка от друга транзакция. Четящата програма не заключва извлечените редове. При това ниво не се решава проблема с несъгласуваните данни. Това е нивото по премълчаване в БД с транзакции по модела на Informix.

REPEATABLE READ / SERIALIZABLE

Транзакцията получава SHARE ключалка за всеки извлечен ред. Това означава, че друга конкурентна транзакция може да получи след това за същите редове само SHARE ключалка. Следователно до края на транзакцията редовете не могат да бъдат изменени от друг. Това ниво използва най-много ключалки и ги държи най-дълго, но решава проблема с несъгласуваните данни. Това е нивото по премълчаване в БД с транзакции по модела на ANSI.

В Informix няма разлика между нивата REPEATABLE READ и SERIALIZABLE, въпреки че според стандарта има. Най-високата степен на изолация е SERIALIZABLE.

Операторът в стандарта, с който се изменя нивото на изолация и се задава типа на достъп до БД, може да се използва само в транзакция.

```
SET TRANSACTION [READ ONLY | READ WRITE]
      ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED
                        |REPEATABLE READ | SERIALIZABLE }
```

Освен него в Informix се поддържа още един оператор, с който само се сменя нивото на изолация в транзакцията и може да се използва няколко пъти в една транзакция.

```
SET ISOLATION TO { DIRTY READ | COMMITED READ | REPEATABLE READ }
```

Параметри на блокиране

Ако транзакция се опитва да получи ключалка за ред или таблица, но СУБД не е в състояние да даде ключалката веднага, каква е реакцията на СУБД. До сега предполагахме, че тогава изпълнението на програмата се подтиска, докато СУБД е в състояние да даде ключалката (програмата чака). В някои СУБД са реализирани няколко възможни реакции при неуспешен опит за блокиране. В Informix се поддържа оператор, с който програмата определя една от три възможни реакции:

```
SET LOCK MODE TO { NOT WAIT | WAIT [ секунди ] }
```

`NOT WAIT` - Програмата не чака, а операторът завършва с грешка.

`WAIT` - Изпълнението на програмата се отлага, докато СУБД е в състояние да й даде ключалката.

`WAIT` *`секунди`* - Програмата чака най-много указания брой секунди.

ACID аксиоми за транзакциите

Всичко за транзакциите може кратко да се обобщи в следните четири свойства на транзакциите, които са известни като ACID аксиоми.

Atomicity

Изпълнява се цялата транзакция или нищо.

Consistency

Ако състоянието на БД е непротиворечиво преди транзакцията, то ще е непротиворечиво и след нея.

Isolation

Изпълнението на една транзакция е изолирано от това на другите конкурентно изпълнявани транзакциии. Всеки потребител трябва да работи с БД без да се безпокои от действията на другите потребители, т.е. като че има монополен достъп до БД.

Durability

След потвърждаване на транзакцията ефекта от нея е постоянен.