Databasesystemer, forår 2006
IT Universitetet i København

Forelæsning 4: Business rules, constraints & triggers.

2. marts 2005

Forelæser: Esben Rune Hansen
Constraints and triggers

- Uniqueness constraints (identifiers/candidate keys, identifier/primary key)
- Assertion-based constraints
- Foreign keys
- Triggers
In this lecture I will assume that you remember:

- Identifiers in E-R diagrams.
- Cardinality constraints in E-R diagrams.
- How to convert an E-R diagram into relations.
- The SQL used when modifying or adding tuples in a relation.
Next: Uniqueness constraints.
Identifiers in E-R diagrams

When translated to relations, identifiers in E-R diagrams serve to:

- Uniquely identify tuples in relations corresponding to entities.
- Together, identifiers of participating entities uniquely identify tuples in relations corresponding to relationships.

We would like the DBMS to make sure that data conforms to these uniqueness constraints. This is done by declaring the unique identifiers as “primary keys”.
Declaring a primary key

Add to the relation schema a line of the form:

```plaintext
PRIMARY KEY (<list of attributes>)
```

If the primary key has just one attribute, we may instead write PRIMARY KEY immediately after the definition of the data type of the attribute, e.g.:

```plaintext
id INT PRIMARY KEY,
```

NULL values are not allowed in attributes of a primary key.
When a key constraint is violated

When a key constraint is violated, an error message is produced.

The state of the database (i.e., the data it contains) is restored to what it was before the action that caused the violation.

Updates in SQL are grouped in units called transactions (more about transactions later in the course).

Constraint-violating transactions are undone (or rolled back).
SQL> CREATE TABLE Students ( 
2  cpr VARCHAR(10) PRIMARY KEY, 
3  name VARCHAR(20), 
4  address VARCHAR(20) 
5 );
Table created.

SQL> INSERT INTO Students VALUES ('0602751127', 'Ethan Longwinder', 'My Way 2');
1 row created.

SQL> INSERT INTO Students VALUES ('0602751127', 'Ethan Longwinder II', 'My Way 2');
INSERT INTO Students VALUES ('0602751127', 'Ethan Longwinder II', 'My Way 2')
ERROR at line 1:
ORA-00001: unique constraint (ESBEN.SYS_C005079) violated
SQL> INSERT INTO Students VALUES (NULL,'Mysterio Student','No Way 8');
ERROR at line 1:
ORA-01400: cannot insert NULL into ("ESBEN"."STUDENTS"."CPR")

SQL> SELECT * FROM Students;

<table>
<thead>
<tr>
<th>CPR</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0602751127</td>
<td>Ethan Longwinder</td>
<td>My Way 2</td>
</tr>
</tbody>
</table>
If we want the DBMS to check other uniqueness constraints, we may add to the SQL relation schema any number of lines of the form:

```sql
UNIQUE (<list of attributes in key>)
```

Uniqueness is *not* guaranteed for tuples having NULL values in the key attributes. However, NULL values can be prevented by adding a NOT NULL constraint after the declaration of each key attribute.
Unique Key Example, slide 1 of 3

```sql
SQL> CREATE TABLE Students (  
    cpr VARCHAR(10) PRIMARY KEY,  
    name VARCHAR(30) NOT NULL,  
    address VARCHAR(20)  
) CONSTRAINT my_constraint UNIQUE (name,address) ;
Table created.
```

```sql
SQL> INSERT INTO Students VALUES ('0602751127','Ethan Longwinder','My Way 2');
1 row created.
```

```sql
SQL> INSERT INTO Students VALUES ('0602751129','Ethan Longwinder','My Way 2');
ERROR at line 1:
ORA-00001: unique constraint (ESBEN.MY_CONSRAINT) violated
```
SQL> INSERT INTO Students VALUES ('2103780002', 'H. Omeless', NULL);
1 row created.

SQL> INSERT INTO Students VALUES ('2103780004', NULL, NULL);
ERROR at line 1:
ORA-01400: cannot insert NULL into ("ESBEN"."STUDENTS"."NAME")

SQL> SELECT * FROM Students;

CPR       NAME          ADDRESS
--------- ----------- ----------
0602751127 Ethan Longwinder My Way 2
2103780002 H. Omeless
SQL> INSERT INTO Students VALUES ('0602751129','Bullie Bank','Goa Way 10');
1 row created.

SQL> INSERT INTO Students VALUES ('0602751131','Ethan Longwinder','My Way 4');
1 row created.

SQL> UPDATE Students SET address = 'Urban Collective';

UPDATE Students SET address = 'Urban Collective'
ERROR at line 1:
ORA-00001: unique constraint (ESBEN.MYCONSTRAINT) violated

SQL> SELECT * FROM Students;

<table>
<thead>
<tr>
<th>CPR</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0602751127</td>
<td>Ethan Longwinder</td>
<td>My Way 2</td>
</tr>
<tr>
<td>2103780002</td>
<td>H. Omeless</td>
<td></td>
</tr>
<tr>
<td>0602751129</td>
<td>Bullie Bank</td>
<td>Goa Way 10</td>
</tr>
<tr>
<td>0602751131</td>
<td>Ethan Longwinder</td>
<td>My Way 4</td>
</tr>
</tbody>
</table>
Should every relation have a primary key?

Short answer: Not necessarily.

Consider for example the relation corresponding to a simple multivalued attribute in an E-R diagram:

- Typically, the only candidate key would consist of both attributes.
- Thus, a primary key constraint would only serve the purpose of eliminating duplicate tuples.
Next: Assertion-based constraints.
The constraint NOT NULL may be specified for any attribute in a relation schema, indicating that NULL is not a legal value.

In general, any attribute that does not correspond to an optional attribute in the E-R diagram should be declared NOT NULL.
Many business rules can be expressed as so-called CHECK constraints, which are assertions (i.e., conditions that must be true) about attributes or tuples of a relation.

- A CHECK constraint on an attribute is checked every time
  - a value of this attribute is modified.
  - a new tuple is inserted.

- A CHECK constraint on tuples is checked every time
  - an attribute value changes.
  - a new tuple is inserted.

- If a constraint is violated, the current transaction is rolled back, and an error message is produced.
Writing attribute-based \texttt{CHECK} constraints

A constraint $C$ on an attribute is declared by writing

\begin{verbatim}
CHECK  C
\end{verbatim}

immediately after the datatype definition.

The condition $C$ may refer to other attributes of the relation, and even to other relations, using a subquery.

(However, Oracle does not allow SQL queries in $C$.)

Examples:

- percentage INT CHECK (percentage $\geq 0$ AND percentage $\leq 100$)
- cpr CHAR(10) CHECK (cpr IN (SELECT cpr FROM students))
Writing tuple-based **CHECK** constraints

A constraint $C$ on tuples is declared by adding the line

```
CHECK $C$
```

to the relation schema definition.

The only difference to attribute-based CHECK constraints is *when* the constraint is checked.

**Examples:**

- CHECK (upper-bound $=>$ lower-bound)
- CHECK (cpr IN (SELECT cpr FROM students))
Next: Foreign keys.
A **foreign key constraint** on an attribute is a constraint saying that its attribute values can *always be found in exactly one place in another relation*.

Foreign key constraints are typically used to express **referential integrity**, i.e., that values supposed to refer to tuples in other tables indeed do so.

If we want the DBMS to check foreign key constraints, we may add to the SQL relation schema any number of declarations of the form:

```
FOREIGN KEY (<attribute name>)
    REFERENCES <tablename>(<attribute name>)
```
Foreign keys may be **composite**, i.e., consist of several attributes.

The syntax for declaring composite foreign keys is the obvious extension of what we saw before:

```sql
FOREIGN KEY (<list of attribute names>)
    REFERENCES <table name> (<list of attribute names>)
```
Semantics of a foreign key constraint

Suppose the schema for relation $R$ contains the declaration

$$\text{FOREIGN KEY } (A_1, \ldots, A_n) \text{ REFERENCES } S(B_1, \ldots, B_n).$$

Then the relation $S$ must have $B_1, \ldots, B_n$ as primary keys or contain a declaration like

$$\text{UNIQUE } (B_1, \ldots, B_n).$$

This means that the DBMS checks that any values of $A_1, \ldots, A_n$ in a tuple of $R$ can also be found as values of $B_1, \ldots, B_n$ in a tuple of $S$. 
### Assertion Example, slide 1 of 3

**SQL>** CREATE TABLE ITUpeople (  
2     cpr VARCHAR(10) PRIMARY KEY,  
3     name VARCHAR(30) NOT NULL,  
4     address VARCHAR(20)
5     );

Table created.

**SQL>** CREATE TABLE Students (  
2     cpr VARCHAR(10) PRIMARY KEY  
3     CONSTRAINT ValidCPR REFERENCES ITUpeople(cpr),  
4     enrolled VARCHAR(10),  
5     graduated VARCHAR(10),  
6     gpa REAL CHECK (gpa>=6 AND gpa<=13),  
7     CONSTRAINT PositiveStudyTime CHECK (enrolled < graduated)
8     );

Table created.

**SQL>** INSERT INTO ITUpeople VALUES ('0602751129','Bullie Bank','Goa Way 10');

1 row created.
### Assertion Example, slide 2 of 3

```
SQL> INSERT INTO Students VALUES ('0602751129', '2003-08-01', NULL, NULL);
1 row created.
```

```
SQL> UPDATE Students SET graduated = '2001-02-28' WHERE cpr='0602751129';
```

```
ERROR at line 1:
ORA-02290: check constraint (ESBEN.POSITIVESTUDYTIME) violated
```

```
SQL> DELETE FROM ITUpeople WHERE cpr='0602751129';
```

```
ERROR at line 1:
ORA-02292: integrity constraint (ESBEN.VALIDCPR) violated - child record found
```

```
SQL> SELECT * FROM ITUpeople;
```

<table>
<thead>
<tr>
<th>CPR</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0602751129</td>
<td>Bullie Bank</td>
<td>Goa Way 10</td>
</tr>
</tbody>
</table>
SQL> ALTER TABLE Students DROP CONSTRAINT ValidCPR;
Table altered.

SQL> ALTER TABLE Students ADD CONSTRAINT ValidCPR
     2 FOREIGN KEY (cpr) REFERENCES ITUpeople(cpr) ON DELETE CASCADE;
Table altered.

SQL> DELETE FROM ITUpeople WHERE cpr='0602751129';
1 row deleted.

SQL> SELECT * FROM ITUpeople;
no rows selected

SQL> SELECT * FROM Students;
no rows selected
What is the difference (if any) between the CHECK constraint

cpr CHAR(10) CHECK (cpr IN (SELECT cpr FROM students))

and the referential integrity constraint

cpr CHAR(10) REFERENCES students(cpr)
Referential integrity from E-R diagrams

If a relationship in our E-R diagram has an “exactly one” cardinality constraint, it can be expressed as a foreign key constraint.

This means that the DBMS maintains the referential integrity of the relationship.

There seems to be no general way to express an “at least one” cardinality constraint.

Note that in supertype-subtype relationships there is an implicit “exactly one” cardinality constraint.
--- Maintaining referential integrity ---

The default (i.e., standard) policy when a transaction violates a foreign key constraint is to roll the transaction back.

However, for each referential constraint we may choose from two other policies for handling changes to the referenced relation:

- **The cascade policy:**
  - If the foreign key attribute values of a tuple were changed, change all references to this tuple to the new value.
  - If a tuple is deleted, delete all tuples referencing it.

- **The set-null policy:**
  - If some reference became invalid, set all its attribute values to NULL.
Next: Triggers.
Triggers is a general mechanism for:

- Enforcing constraints/business rules, and more generally
- Making the DBMS perform actions on certain events.

The definition of a trigger consists of an event, a condition, and an action.

- Triggers are awakened (or triggered) when the event, a certain change to the database, occurs.
- If the condition associated with the trigger is true, then the action is performed.
Key features of triggers in SQL:

- Triggering events are insertions, deletions, and updates of tuples.
- The action can be any SQL statement. (But most RDBMSs have restrictions on the SQL allowed in the action.)
- The action can refer to values from both before and after the event.
- The action can be performed either
  - After each event that activates the trigger, or
  - At the end of each transaction where one or more events activated the trigger.
Trigger Example, slide 1 of 4

SQL> select * from MovieExec;

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>CERT</th>
<th>NETWORTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>George Lucas</td>
<td>Oak Rd.</td>
<td>555</td>
<td>2000000000</td>
</tr>
<tr>
<td>Ted Turner</td>
<td>Turner Av.</td>
<td>333</td>
<td>1250000000</td>
</tr>
<tr>
<td>Stephen Spielberg</td>
<td>123 ET road</td>
<td>222</td>
<td>1000000000</td>
</tr>
<tr>
<td>Merv Griffin</td>
<td>Riot Rd.</td>
<td>199</td>
<td>1120000000</td>
</tr>
<tr>
<td>Calvin Coolidge</td>
<td>Fast Lane</td>
<td>123</td>
<td>200000000</td>
</tr>
</tbody>
</table>

---

SQL> CREATE TABLE NetworthHistory ( name VARCHAR(25), oldnetworth INT, newnetworth INT );
Table created.
```sql
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING
OLD AS Oldtuple
NEW AS Newtuple
FOR EACH ROW
WHEN (Oldtuple.networth <> NewTuple.networth)
BEGIN
    INSERT INTO NetworthHistory
    VALUES (:Oldtuple.name,:Oldtuple.networth,:Newtuple.networth);
END;
/
```

Trigger created.
Trigger Example, slide 3 of 4

\[
\text{SQL> } \text{UPDATE MovieExec}
\]
\[
\begin{align*}
2 & \quad \text{SET netWorth} = 29000000 \\
3 & \quad \text{WHERE name='George Lucas';}
\end{align*}
\]

1 row updated.

\[
\text{SQL> } \text{SELECT * FROM NetworthHistory;}
\]

<table>
<thead>
<tr>
<th>NAME</th>
<th>OLDDNETWORTH</th>
<th>NEWNETWORTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>George Lucas</td>
<td>200000000</td>
<td>290000000</td>
</tr>
</tbody>
</table>
Trigger Example, slide 4 of 4

```
SQL> UPDATE MovieExec
  2      SET netWorth = 25000000
  3      WHERE name='George Lucas';

1 row updated.
```

```
SQL> SELECT * FROM NetworthHistory;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>OLDPNETWORTH</th>
<th>NEWPNETWORTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>George Lucas</td>
<td>2000000000</td>
<td>290000000</td>
</tr>
<tr>
<td>George Lucas</td>
<td>290000000</td>
<td>250000000</td>
</tr>
</tbody>
</table>
Syntax in Oracle (differs slightly from SQL definition):

```
CREATE TRIGGER <name of trigger> AFTER
INSERT | DELETE | UPDATE
[OF <attribute name>] ON <name of relation or view>
[REFERENCING OLD AS <name>, NEW AS <name>]
[FOR EACH ROW
[WHEN <condition>]]
BEGIN
<PL/SQL commands>
END;
```

Vertical lines | between alternatives. Brackets [] around optional parts.

Variables in <PL/SQL commands> must be prefixed by semicolon (:old.a).
As a minimum, you should after this week:

- Know how to declare key constraints and referential integrity (i.e., foreign key) constraints in SQL.
- Understand the basic mechanisms for maintaining referential integrity.
- Know how to declare tuple-based CHECK constraints, and know how these are checked.
- Understand how to define triggers, and the mechanism for executing triggers in SQL.