

BCNF and Transaction

Université Grenoble Alpes

20/04/2023

Bahareh Afshinpour

bahareh.afshinpour@univ-grenoble-alpes.fr

Main reference:

A First Course in Database Systems (and associated material) by
J. Ullman and J. Widom, Prentice-Hall

Example

- Is R in the 3NF?

$R(A,B,C,D,E,F)$ $FD=\{AB \rightarrow CDEF, BD \rightarrow F\}$

$ABCDEF^+ = \{A,B,C,D,E,F\}$

$AB^+ = \{A,B,C,D,E,F\}$ AB is a super key. Check out candidate key : $A^+ = \{A\}$ $B^+ = \{B\}$

AB is a candidate key so prime={A,B}

To check out if more candidate key exist : see right side and find prime attribute

No more candidate key : so candidate key={AB} , Prime={A,B} Non-Prime={C,D,E,F}



Non prime attribute → Non prime attribute

$FD=\{AB \rightarrow CDEF, BD \rightarrow F\}$ $AB \rightarrow CDEF$ prime → non-prime

$BD \rightarrow F$ B is prime, D is non-prime. BD is Non-prime Non-prime → Non-prime

R is not in 3NF

BCNF(Boyce-Codd Normal Form)

- It is strong version of 3NF.

The relation is in BCNF:

- It is in 3NF
- For each non-trivial functional dependency
The **left** hand side of dependency must be a Supekey (**X**->Y)

Example

- $R(A,B,C)$ FD={A→B, B→C, C→A}

First find out the candidate key:

$ABC^+ = \{A,B,C\}$ we have all the relation so it is a super key

We start to discard as many attribute that we can since a candidate key is a minimal super key.

~~$ABC^+ = \{A,B,C\}$~~ $A^+ = \{A,B,C\}$ A has no proper subset. If no proper subset is possible, then there is no chance to have superkey CK= yes, Prime attribute={A}

More candidate key ??? Check the right side of the FD.

If you can find the prime attribute we have more candidate key

C→A so here we have more candidate key

Example

$R(A,B,C)$ $FD=\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

$C \rightarrow A$ so here we have more candidate key

$CK=A$, replace A with C. Is C candidate key?? We have to check.

$C^+ = \{C, A, B\}$ and C has no proper subset. $CK=yes$

Prime attributes= $\{A, C\}$

see right side. We have $B \rightarrow C$. So we replace C by B.

Prime attributes= $\{A, C, B\}$

See the left side:

$A \rightarrow B$

$B \rightarrow C$

$C \rightarrow A$

All the left side are candidate key. Definitely they are super key. So

This relation is in the BCNF

Find the highest normal form in R

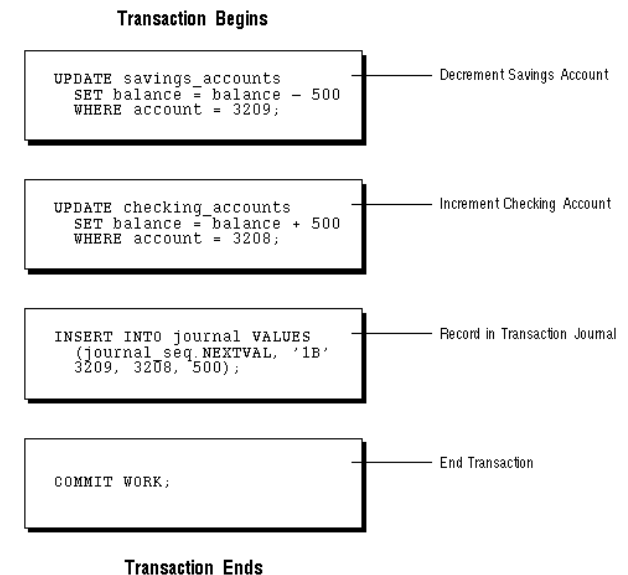
- $R(A,B,C,D,E)$ FD={A→BCDE, BC→ACE, D→E}

Transaction


Transaction

- Transaction is a group or set of tasks into a single execution unit.
- Each transaction begins with a specific task and ends when all the tasks in the group successfully complete.
- If any of tasks fails the transactions fails.
- The effects of all the SQL statements in a transaction can be either all *committed* (applied to the database) or all *rolled back* (undone from the database).

- Decrement the savings account
- Increment the checking account
- Record the transaction in the transaction journal



Transaction

Unit of works  Tx

In this unit of work, there could be multiple changes on **multiple different rows** in **different tables** that occur all that once.

For example 

Insert Into Table1
Update Table 2

A transaction is a logical, atomic unit of work that contains one or more SQL statements.

An important trait (property) of a transaction is the fact that these two things must either succeed or fail together as a unit
Therefore, a transactions has only two results : - success -Failure

Example 6.41: Let us picture another common sort of database: a bank's account records. We can represent the situation by a relation

`Accounts(acctNo, balance)`

Consider the operation of transferring \$100 from the account numbered 123 to the account 456. We might first check whether there is at least \$100 in account 123, and if so, we execute the following two steps:

1. Add \$100 to account 456 by the SQL update statement:

```
UPDATE Accounts
SET balance = balance + 100
WHERE acctNo = 456;
```

2. Subtract \$100 from account 123 by the SQL update statement:

```
UPDATE Accounts
SET balance = balance - 100
WHERE acctNo = 123;
```

Now, consider what happens if there is a failure after Step (1) but before Step (2). Perhaps the computer fails, or the network connecting the database to the processor that is actually performing the transfer fails. Then the database is left in a state where money has been transferred into the second account, but the money has not been taken out of the first account. The bank has in effect given away the amount of money that was to be transferred. □

- When using the generic SQL interface, each statement is a transaction by itself.
- Also, SQL allows the programmer to group several statement into a single transaction.
- The SQL command **START TRANSACTION** is used to mark the beginning of the transaction.
- There are two ways to end a transaction:
 - Using COMMIT
 - Using ROLLBACK

Example

We have two different users.

```
SQL> select * from department;
```

DEPTID	DEPTNAME
10	BCA
20	MCA
30	BBA
40	MSc
50	MTech
70	Btech
80	MTechCS

7 rows selected.

```
SQL> select * from system.department;
```

DEPTID	DEPTNAME
10	BCA
20	MCA
30	BBA
40	MSc
50	MTech
70	Btech
80	MTechCS

7 rows selected.

User1(system) want to add one new tuple in the department

Example

```
SQL> insert into department values(90,'MTechIT');  
1 row created.  
SQL> select * from department;  
  
  DEPTID DEPTNAME  
-----  
      10  BCA  
      20  MCA  
      30  BBA  
      40  MSc  
      50  MTech  
      70  Btech  
      80  MTechCS  
      90  MTechIT  
  
8 rows selected.
```

If we proceed in the sys terminal, and check the department

```
SQL> select * from system.department;  
  
  DEPTID DEPTNAME  
-----  
      10  BCA  
      20  MCA  
      30  BBA  
      40  MSc  
      50  MTech  
      70  Btech  
      80  MTechCS  
  
7 rows selected.
```

We can not see any changes in the table

The changes must be saved otherwise it is discarded

When any user is updating any changes, only the user itself can see the changes without commit or rollback. No other user can access or view the updates, as it is not permanently saved by the user who performed them.

Example

```
SQL> select * from department;
```

DEPTID	DEPTNAME
--------	----------

10	BCA
20	MCA
30	BBA
40	MSc
50	MTech
70	Btech
80	MTechCS
90	MTechIT

8 rows selected.

```
SQL> commit;
```

Commit complete.

```
SQL> select * from system.department;
```

DEPTID	DEPTNAME
--------	----------

10	BCA
20	MCA
30	BBA
40	MSc
50	MTech
70	Btech
80	MTechCS
90	MTechIT

8 rows selected.

ROLLBACK

```
SQL> delete from department;
```

```
8 rows deleted.
```

```
SQL> select * from department;
```

```
no rows selected
```

```
SQL> rollback;
```

```
Rollback complete.
```

```
SQL> select * from department;
```

```
DEPTID DEPTNAME
```

```
-----
```

```
10 BCA
```

```
20 MCA
```

```
30 BBA
```

```
40 MSc
```

```
50 MTech
```

```
70 Btech
```

```
80 MTechCS
```

```
90 MTechIT
```

```
8 rows selected.
```

- Without Commit or Rollback, the permanent update is not possible.
- But if the user changes or updates sth and disconnects from the database properly, commits occur.

If user disconnects from the database after some changes auto commits occurs.

Transaction

- **START TRANSACTION** or **BEGIN** start a new transaction.
- **COMMIT** commits the current transaction, making its changes permanent.
- **ROLLBACK** rolls back the current transaction, canceling its changes.
- **SET autocommit** disables or enables the default autocommit mode for the current session.

```
--Begin the transaction  
SET TRANSACTION READ WRITE;
```

```
--Create the new project  
INSERT INTO project  
  SELECT 1007, project_name, project_budget FROM project  
  WHERE project_id = 1002;
```

```
--Point the time log rows in project_hours to the new project number  
UPDATE project_hours  
SET project_id = 1007  
WHERE project_id = 1002;
```

```
--Delete the original project record  
DELETE FROM project  
WHERE project_id=1002;
```

```
COMMIT;
```

SET TRANSACTION marks the beginning of a transaction. Any changes you make to your data following the beginning of a transaction are not made permanent until you issue a **COMMIT**.

Tip

Using **SET TRANSACTION** to begin a transaction is **optional**. A new transaction begins implicitly with the first DML statement that you execute after you make a database connection or with the first DML statement that you execute following a **COMMIT** or a **ROLLBACK** (or any DDL statement such as **TRUNCATE**). You need to use **SET TRANSACTION** only when you want transaction attributes such as **READ ONLY** that are not the default.

read/write transaction : Such a transaction is the default, and it allows you to issue statements such as UPDATE and DELETE.

You can also create **read-only transactions**:

If we tell the SQL execution system that our current transaction is *read-only*, that is, it will never change the database, then it is quite possible that the SQL system will be able to take advantage of that knowledge. Generally it will be possible for many read-only transactions accessing the same data to run in parallel, while they would not be allowed to run in parallel with a transaction that wrote the same data.

We tell the SQL system that the next transaction is read-only by:

```
SET TRANSACTION READ ONLY;
```

This statement must be executed before the transaction begins.

ACID Transactions

- A DBMS is expected to support “*ACID transactions*,” processes that are:
 - *Atomic* : All actions of a transaction are atomic and either they are all performed or none of the actions are performed.
 - *Consistent* : Each transaction, when run alone, must preserve the consistency of the database.
 - *Isolated* : Each transaction is isolated (protected) from the effects of other concurrently running transactions.
 - *Durable* : Effects of a process do not get lost if the system crashes. once a transaction commits, the data should persist in the database even if the system crashes before the data is written to non-volatile storage.

Isolation Levels

- SQL defines four *isolation levels* = choices about what interactions are allowed by transactions that execute at about the same time.
- How a DBMS implements these isolation levels is highly complex, and a typical DBMS provides its own options.

SQL

Updates

- To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>
```

```
SET <list of attribute assignments>
```

```
WHERE <condition on tuples>;
```

Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers  
SET phone = '555-1212'  
WHERE name = 'Fred';
```

Example: Update Several Tuples

- Make \$4 the maximum price for beer:

```
UPDATE Sells  
SET price = 4.00  
WHERE price > 4.00;
```

Adding Attributes

- We may add a new attribute (“column”) to a relation schema by:

```
ALTER TABLE <name> ADD  
    <attribute declaration>;
```

- Example:

```
ALTER TABLE Bars ADD phone CHAR(16)
```


Deleting Attributes

- Remove an attribute from a relation schema by:

ALTER TABLE <name>

DROP <attribute>;

- Example: we don't really need the license attribute for bars:

```
ALTER TABLE Bars DROP license;
```

Views

- A *view* is a “virtual table” = a relation defined in terms of the contents of other tables and views.
 - $V = \text{viewquery}(R_1, R_2, \dots, R_N)$
- Declare by:
CREATE VIEW <name> AS <query>;

Example: View Definition

- `CanDrink(drinker, beer)` is a view “containing” the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
  SELECT drinker, beer
  FROM Frequents, Sells
  WHERE Frequents.bar = Sells.bar;
```

Example: Accessing a View

- a limited ability to modify views if it makes sense as a modification of one underlying base table.
- Example query:

```
SELECT beer FROM CanDrink  
WHERE drinker = 'Sally';
```

Query + Subquery Solution

```
SELECT bar  
FROM Sells  
WHERE beer = 'Miller' AND
```

```
price = (SELECT price  
        FROM Sells  
        WHERE bar = 'Joe''s Bar'  
        AND beer = 'Bud');
```

The price at
which Joe
sells Bud



The IN Operator

- `<tuple> IN <relation>` is true if and only if the tuple is a member of the relation.
 - `<tuple> NOT IN <relation>` means the opposite.
- IN-expressions can appear in WHERE clauses.
- The `<relation>` is often a subquery.

Example

- From **Beers(name, manf)** and **Likes(drinker, beer)**, find the name and manufacturer of each beer that Fred likes.

```
SELECT *
```

```
FROM Beers
```

```
WHERE name IN (SELECT beer
```

The set of
beers Fred
likes



```
FROM Likes
```

```
WHERE drinker = 'Fred');
```

The Operator ANY

- $x = \text{ANY}(\langle \text{relation} \rangle)$ is a boolean condition true if x equals at least one tuple in the relation.
- Similarly, $=$ can be replaced by any of the comparison operators.
- Example: $x \geq \text{ANY}(\langle \text{relation} \rangle)$ means x is not the smallest tuple in the relation.
 - Note tuples must have one component only.

The Operator ALL

- Similarly, $x \neq \text{ALL}(\langle \text{relation} \rangle)$ is true if and only if for every tuple t in the relation, x is not equal to t .
 - That is, x is not a member of the relation.
- The \neq can be replaced by any comparison operator.
- Example: $x \geq \text{ALL}(\langle \text{relation} \rangle)$ means there is no tuple larger than x in the relation.

Example

- From `Sells(bar, beer, price)`, find the beer(s) sold for the highest price.

```
SELECT beer
```

```
FROM Sells
```

```
WHERE price >= ALL(  
    SELECT price  
    FROM Sells);
```

price from the outer
Sells must not be
less than any price.

