# Updates to Relational Schemata

Query classification:

- Schema definition queries:
  - Define and/or change relations and constraints
- Data definition queries:
  - Passive (ask a question)
  - Active (update the database)

We now look at active data-definition queries.

# Flavors of Update Directives within SQL

Within SQL, there are two fundamental flavors of update operations:

- *Cursor Operations* involve the use of a special variable, called a cursor, which is used to traverse a set of tuples, one at a time.
  - This approach is used primarily in embedded and module-based approaches, in which the SQL lives within a host programming language.
  - The cursor is typically a variable in a host programming language.
  - These notes will not look further at cursor operations.

- *Noncursor Operations* do not involve the use of cursors.
  - Applicable to direct SQL.
  - Examined in these notes.
  - Four principal forms:
    - Select ... into ...
    - Insert into ...
    - Delete from ...
    - Update ... set ...

Here are some variations of the Insert command, using the Company schema of the textbook:

```
Insert into Employee values
('Kari',' ','Nordmann','000000001',
 Date '1960-12-25',
’Thunes vei 10A, 0274 Oslo','F', 100000,null,5);

Insert into Employee values
('Ola',' ','Nordmann','000000002',
 Date '1955-12-25',
'Thunes vei 10A, 0274 Oslo','M',
50000.50,null,5);
```

In the following example, unspecified fields are left null.

```
Insert into Employee
(LName,FName,SSN,DNo,Salary)
values
('Garnett','Kevin','111111111',5,21000000);
```

Here is a more complex insertion example, in which a table of supervisors is created.

```
Create Table Bosses
    (FName    Varchar(15)   not null,
     MInit    Char(1),
     LName    Varchar(15)   not null,
     SSN      Char(9)       not null,
     DName    Varchar(15)   not null,
     Constraint pkey_boss primary key (SSN));

Insert Into Bosses
Select  E.FName, E.Minit, E.LName, E.SSN,
        D.DName
From    Employee E,  Department D
Where  (E.SSN = D.MgrSSN) AND
        (E.DNo = D.DNumber);
```

- The *Select ... Into ...* directive in PostgreSQL and Microsoft Access has the effect of creating a table and then inserting values.

- The following example creates at table named Bosses1 which contains the same tuples as the Bosses table of the previous example.

```
Select   E.FName, E.MInit, E.LName, E.SSN,
          D.DName
Into     Bosses1
From     Employee E, Department D
Where    (E.SSN = D.MgrSSN) AND
          (E.DNo = D.DNumber);
```

- Warning: It is not clear that this is standard SQL.

- Some SQL references describe quite different semantics for this directive.

- Use it with caution in code which may need to be ported to another system.

- For portability and readability, it is best to use a *Create Table* followed by an *Insert*, as illustrated on the previous slide.

- The *Update ... Set ...* directive is fairly straightforward.

- Here is an example which makes everyone on the Computerization project work harder.

```
Update    Works_On
Set       Hours = Hours + 10
Where     PNo in
          (Select PNo
          From   Project, Works_On
          Where  PNo = PNumber
          and PName = 'Computerization'
          );
```

- The *Delete From* directive is very straightforward.

- Here is an example which removes all working instances of greater than 40 hours.

```
Delete From    Works_On
Where          Hours >= 40;
```

- Some Difficulties Surrounding Updates

- The aspect of managing updates which makes things nontrivial is checking *integrity constraints.*

- Two basic forms of update philosophies:

  - *Tuple-at-a-time:* Perform the updates one tuple at a time, checking for satisfaction of the integrity constraints after each tuple operation. These are called *immediate constraints* in SQL.

  - *Transaction at a time:* Perform all requested updates as a block, and verify that the integrity of the database is satisfied only upon conclusion of the block operation. These are called *deferred constraints* in SQL. (Not supported in Access.)

    Comparison:

| Tuple-at-a-time: | Transaction-at-a-time: |
|---|---|
| - Hinders realistic updates | + Allows most realistic updates |
| + Simpler to implement | - More complex to implement |
| Used in low-end systems | Used in high-end systems |

  - Transaction-at-a-time processing is not available in Microsoft Access.

- In Access, when a directive mandates insertion, deletion, or update of a set of tuples, the following rules (seem to) hold.
    - The tuples are inserted, deleted, or updated in the order in which the appear in the source relation, or are generated in the source command.
    - Satisfaction of integrity constraints depends upon the order in which the tuples are fetched from the source relation. Integrity constraints must hold at each intermediate step.

- In a system with tuple-at-a-time update, database initialization may involve a “chicken-and-egg” problem.
- In the Company database example from the textbook:
    - Every department must have a manager.
    - Every employee must work in some department.
- How is the database initialized?
    - The (not particularly elegant) solution is to build the database first, without constraints, and then install the constraints.

- Some other tricky problems on the Company database which may occur in the absence of transactions:
  - The following update could pose a problem if the manager of an employee to be deleted is deleted first.

    ```
    Delete   Employee.*
    From     Employee
    Where    Salary < 30000
    ```

    - Whether or not this will work depends upon this order in which tuples are processed.
    - A better solution is to delete all non-managers first.
    - This example ignores the further constraints that every department must have a manager.
- Here are some others to think about.
  - Swap managers for two departments.
  - Hire a new employee who is to be the manager of a new department.
- Since different systems may process tuples in different orders, only those solutions which are independent of the order of processed tuples should be used in code which has the possibility to be ported.

# Transactions in SQL and PostgreSQL

- These problems may be solved via the notion of a *transaction,* an SQL construct which Access does not support.
- An SQL *transaction* is a block of statements surrounded by Begin ... Commit markers.
- In a transaction, certain integrity checking is deferred until the Commit directive is encountered.
- Observe that, in the previous examples, the problems which arise are due to foreign-key constraints.
- In PostgreSQL, the deferred checking applies only to foreign-key constraints; other constraints are checked immediately.
- This may or may not be true in other systems; the detailed semantics of transactions are not standardized.
- In the absence of transaction directives, each SQL statement is taken to be a distinct transaction.
- Transactions will be discussed in more detail later in the course.

# Some general issues regarding the computational complexity of supporting updates

- Consider the difference:
  - Checking an entire database for integrity
  - Checking a database for integrity after an update operation, assuming that it was correct before the operation

Time complexity for verifying constraints on a database:

- Candidate/primary key constraint, with n the number of tuples in the relation.
  - Sequential access: $O(n^2)$
  - Log access: $O(n \bullet \log(n))$
  - Constant-time access: $O(n)$

- Foreign key constraint, with $n_1$ tuples in the relation with the foreign key and and $n_2$ tuples in the relation with the corresponding primary key.
  - Sequential access: $O(n_1 \bullet n_2)$
  - Log access: $O(n_1 \bullet \log(n_2))$
  - Constant-time access: $O(n_1)$

Time complexity for verifying constraints after an update to a legal database:

- Candidate/primary key constraint, with n the number of tuples in the relation.

| Access | Deletion | Insertion | Update |
|---|---|---|---|
| Sequential | O(0) | O(n) | O(n) / O(0) |
| Log | O(0) | O(log(n)) | O(log(n)) / O(0) |
| Constant | O(0) | O(1) | O(1) / O(0) |

  - Update complexity is the same as insertion if the primary key or a candidate key is altered. No checking is necessary if no primary or candidate key is altered.

- Foreign key constraint, with $n_1$ tuples in the relation $R_1$ with the foreign key and $n_2$ tuples in the relation $R_2$ with the corresponding primary key.

Sequential Access:

| | Deletion | Insertion | Update |
|---|---|---|---|
| $R_1$ | $O(0)$ | $O(n_2)$ | $O(n_2)$ / $O(0)$ |
| $R_2$ | $O(n_1)$ | $O(0)$ | $O(n_1)$ / $O(0)$ |

Log Access:

| | Deletion | Insertion | Update |
|---|---|---|---|
| $R_1$ | $O(0)$ | $O(\log(n_2))$ | $O(\log(n_2))$ / $O(0)$ |
| $R_2$ | $O(\log(n_1))$ | $O(0)$ | $O(\log(n_1))$ / $O(0)$ |

Constant Access:

| | Deletion | Insertion | Update |
|---|---|---|---|
| $R_1$ | $O(0)$ | $O(1)$ | $O(1)$ / $O(0)$ |
| $R_2$ | $O(1)$ | $O(0)$ | $O(1)$ / $O(0)$ |

More information on this topic: Ke Wang and Marc H. Graham, *Constant-Time Maintainability: A Generalization of Independence,* ACM Transactions on Database Systems 17(2), June 1992, pp. 201-246.

Classification of constraints:

- For a positive integer n, call a constraint *n-easy* if it is possible to check the constraint by looking at at most n tuples as a time.

- Examples: Both primary/candidate key constraints and foreign key constraints are 2-easy.

- Example of a constraint which is m-easy, but not k-easy for any $k < m$:
  - Every department must have at least m employees.

Remark: With an update, usually one tuple is fixed, so, an n-easy constraint only needs to look at n-1 other tuples. This is why 2-easy is so nice.

Remark:
- Key constraints are called *equality-generating,* because the condition to be checked is the equality of fields.
- Join constraints are called *tuple-generating,* because the condition to be checked is the existence of further tuples.

- Further classification of (equality-generating) constraints:

- A primary/candidate key constraint is a (∀)(∀)-constraint, because it requires that any two tuples with matching keys match everywhere.

- A foreign key constraint is a (∀)(∃)-constraint, because it checks that for any tuple in the main relation, there is (exists!) a corresponding tuple in the relation whose primary key corresponds to the foreign key of the main relation.

This impacts the complexity of certain update operations:

- If tuples are deleted from a legal database, a (∀)(∀)-constraint cannot be violated as a result.

- If tuples are deleted from the "∀" relation corrsponding to a (∀)(∃)-constraint in a legal database, that constraint cannot be violated.

- If tuples are added to the "∃" relation corrsponding to a (∀)(∃)-constraint in a legal database, that constraint cannot be violated.