ALTER TABLE statement

The ALTER TABLE statement modifies a table.

The ALTER TABLE statement allows you to:

- Add a column to a table
- Add a constraint to a table
- Drop a column from a table
- Drop an existing constraint from a table
- Increase the width of a BLOB, CLOB, VARCHAR, or VARCHAR FOR BIT DATA column
- Override row-level locking for the table (or drop the override)
- Change an identity column in any of the following ways:
  - Change the increment value and start value of the identity column
  - Change the generation condition from ALWAYS to DEFAULT BY behavior or vice-versa
  - Change the overflow handling from CYCLE to NO CYCLE behavior or vice-versa
- Change the nullability constraint for a column
- Change the default value for a column

Syntax

```
ALTER TABLE tableName
{
    ADD COLUMN columnDefinition |
    ADD CONSTRAINT clause |
    DROP [ COLUMN ] columnName [ CASCADE | RESTRICT ] |
    DROP { PRIMARY KEY |
           FOREIGN KEY constraintName |
           UNIQUE constraintName |
           CHECK constraintName |
           CONSTRAINT constraintName } |
    ALTER [ COLUMN ] columnAlteration |
    LOCKSIZE { ROW | TABLE }
}
```

columnAlteration

```
columnName SET DATA TYPE BLOB( integer ) |
columnName SET DATA TYPE CLOB( integer ) |
columnName SET DATA TYPE VARCHAR( integer ) |
columnName SET DATA TYPE VARCHAR( integer ) FOR BIT DATA |
columnName SET INCREMENT BY integerConstant |
columnName RESTART WITH integerConstant |
columnName SET GENERATED { ALWAYS | BY DEFAULT } |
columnName { SET | DROP } NOT NULL |
columnName [ NOT ] NULL |
columnName [ WITH | SET ] DEFAULT defaultValue |
columnName SET [ NO ] CYCLE |
columnName DROP DEFAULT
```

In the *columnAlteration*, SET INCREMENT BY *integerConstant* specifies the interval between consecutive values of the identity column. The next value to be generated for the identity column will be determined from the last assigned value with the increment applied. The column must already be defined with the IDENTITY attribute.

RESTART WITH *integerConstant* specifies the next value to be generated for the identity column. RESTART WITH is useful for a table that has an identity column that was defined as GENERATED BY DEFAULT and that has a unique key defined on that identity column. Because GENERATED BY DEFAULT allows both manual inserts and system generated values, it is possible that manually inserted values can conflict with system generated values. To work around such conflicts, use the RESTART WITH syntax to specify the next value that will be generated for the identity column. Consider the following example, which involves a combination of automatically generated data and manually inserted data:

```
CREATE TABLE tauto(i INT GENERATED BY DEFAULT AS IDENTITY, k INT)
CREATE UNIQUE INDEX tautoInd ON tauto(i)
INSERT INTO tauto(k) values 1,2
```

The system will automatically generate values for the identity column. But now you need to manually insert some data into the identity column:

```
INSERT INTO tauto VALUES (3,3)
INSERT INTO tauto VALUES (4,4)
INSERT INTO tauto VALUES (5,5)
```

The identity column has used values 1 through 5 at this point. If you now want the system to generate a value, the system will generate a 3, which will result in a unique key exception because the value 3 has already been manually inserted. To compensate for the manual inserts, issue an ALTER TABLE statement for the identity column with RESTART WITH 6:

```
ALTER TABLE tauto ALTER COLUMN i RESTART WITH 6
```

SET GENERATED ALWAYS causes Derby to not accept an overriding value for an identity column when a row is inserted or updated. SET GENERATED BY DEFAULT causes Derby to permit these overrides.

The CYCLE clause controls what happens when the identity column exhausts its range and wraps around (CYCLE) or throws an exception (NO CYCLE).

ALTER TABLE does not affect any view that references the table being altered. This includes views that have an "*" in their SELECT list. You must drop and re-create those views if you wish them to return the new columns.

Derby raises an error if you try to change the *DataType* of a generated column to a type which is not assignable from the type of the *generationClause*. Derby also raises an error if you try to add a DEFAULT clause to a generated column.

Adding columns

The syntax for the *columnDefinition* for a new column is the same as for a column in a CREATE TABLE statement. This syntax allows a column constraint to be placed on the new column within the ALTER TABLE ADD COLUMN statement. However, a column with a NOT NULL constraint can be added to an existing table if you give a default value; otherwise, an exception is thrown when the ALTER TABLE statement is executed.

Just as in CREATE TABLE, if the column definition includes a primary key constraint, the column cannot contain null values, so the NOT NULL attribute must also be specified (SQLSTATE 42831).

Note: If a table has an UPDATE trigger without an explicit column list, adding a column to that table in effect adds that column to the implicit update column list upon which the trigger is defined, and all references to transition variables are invalidated so that they pick up the new column.

If you add a generated column to a table, Derby computes the generated values for all existing rows in the table.

ALTER TABLE ADD COLUMN adds the new column at the end of the table row. If you need to change a column in a way not permitted by ALTER TABLE ALTER COLUMN (for example, if you need to change its data type), the only way to do so is to drop the column and add a new one, and this changes the ordering of the columns.

Adding constraints

ALTER TABLE ADD CONSTRAINT adds a table-level constraint to an existing table. Any supported table-level constraint type can be added via ALTER TABLE. The following limitations exist on adding a constraint to an existing table:

- When adding a foreign key or check constraint to an existing table, Derby checks the table to make sure existing rows satisfy the constraint. If any row is invalid, Derby throws a statement exception and the constraint is not added.
- All columns included in a primary key must contain non null data and be unique.

  ALTER TABLE ADD UNIQUE or PRIMARY KEY provide a shorthand method of defining a primary key composed of a single column. If PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause were specified as a separate clause. The column cannot contain null values, so the NOT NULL attribute must also be specified.

For information on the syntax of constraints, see CONSTRAINT clause. Use the syntax for table-level constraint when adding a constraint with the ADD TABLE ADD CONSTRAINT syntax.

Dropping columns

ALTER TABLE DROP COLUMN allows you to drop a column from a table.

The keyword COLUMN is optional.

The keywords CASCADE and RESTRICT are also optional. If you specify neither CASCADE nor RESTRICT, the default is CASCADE.

If you specify RESTRICT, then the column drop will be rejected if it would cause a dependent schema object to become invalid.

If you specify CASCADE, then the column drop should additionally drop other schema objects which have become invalid.

The schema objects which can cause a DROP COLUMN RESTRICT to be rejected include: views, triggers, primary key constraints, foreign key constraints, unique key constraints, check constraints, and column privileges. If one of these types of objects depends on the column being dropped, DROP COLUMN RESTRICT will reject the statement.

Derby also raises an error if you specify RESTRICT when you drop a column referenced by the *generationClause* of a generated column. However, if you specify CASCADE, the generated column is also dropped with CASCADE semantics.

You may not drop the last (only) column in a table.

CASCADE/RESTRICT doesn't consider whether the column being dropped is used in any indexes. When a column is dropped, it is removed from any indexes which contain it. If that column was the only column in the index, the entire index is dropped.

Dropping constraints

ALTER TABLE DROP CONSTRAINT drops a constraint on an existing table. To drop an unnamed constraint, you must specify the generated constraint name stored in *SYS.SYSCONSTRAINTS* as a delimited identifier.

Dropping a primary key, unique, or foreign key constraint drops the physical index that enforces the constraint (also known as a *backing index*).

Modifying columns

The *columnAlteration* allows you to alter the named column in the following ways:

- Increasing the width of an existing VARCHAR or VARCHAR FOR BIT DATA column. CHARACTER VARYING or CHAR VARYING can be used as synonyms for the VARCHAR keyword.

  To increase the width of a column of these types, specify the data type and new size after the column name.

  You are not allowed to decrease the width or to change the data type. You are not allowed to increase the width of a column that is part of a primary or unique key referenced by a foreign key constraint or that is part of a foreign key constraint.

- Specifying the interval between consecutive values of the identity column.

  To set an interval between consecutive values of the identity column, specify the *integerConstant*. You must previously define the column with the IDENTITY attribute (SQLSTATE 42837). If there are existing rows in the table, the values in the column for which the SET INCREMENT default was added do not change.

- Modifying the nullability constraint of a column.

  You can add the NOT NULL constraint to an existing column. To do so there must not be existing NULL values for the column in the table.

  You can remove the NOT NULL constraint from an existing column. To do so the column must not be used in a PRIMARY KEY constraint.

- Changing an identity column from GENERATED ALWAYS to GENERATED BY DEFAULT behavior or vice-versa.

  The SET GENERATED clause may only be applied to identity columns. It cannot be used to convert a non-identity column into an identity column. This clause can be useful if you need to preserve key values when bulk-loading a table from a snapshot or exported dump.

- Changing an identity column from CYCLE to NO CYCLE behavior or vice-versa.

  The CYCLE clause controls what happens when the identity column exhausts its range and wraps around (CYCLE) or throws an exception (NO CYCLE).

- Changing the default value for a column.

  You can use DEFAULT *default-value* to change a column default. To disable a previously set default, use DROP DEFAULT (alternatively, you can specify NULL as the *default-value*).

Setting defaults

You can specify a default value for a new column. A default value is the value that is inserted into a column if no other value is specified. If not explicitly specified, the default value of a column is NULL. If you add a default to a new column, existing rows in the table gain the default value in the new column.

For more information about defaults, see CREATE TABLE statement.

Changing the lock granularity for the table

The LOCKSIZE clause allows you to override row-level locking for the specific table, if your system uses the default setting of row-level locking. (If your system is set for table-level locking, you cannot change the locking granularity to row-level locking, although Derby allows you to use the LOCKSIZE clause in such a situation without throwing an exception.) To override row-level locking for the specific table, set locking for the table to TABLE. If you created the table with table-level locking granularity, you can change locking back to ROW with the LOCKSIZE clause in the ALTER TABLE STATEMENT. For information about why this is sometimes useful, see *Tuning Derby*.

Examples

```
-- Add a new column with a column-level constraint
-- to an existing table
-- An exception will be thrown if the table
-- contains any rows
-- since the newcol will be initialized to NULL
-- in all existing rows in the table
ALTER TABLE CITIES ADD COLUMN REGION VARCHAR(26)
CONSTRAINT NEW_CONSTRAINT CHECK (REGION IS NOT NULL);

-- Add a new unique constraint to an existing table
-- An exception will be thrown if duplicate keys are found
ALTER TABLE SAMP.DEPARTMENT
ADD CONSTRAINT NEW_UNIQUE UNIQUE (DEPTNO);

-- add a new foreign key constraint to the
-- Cities table. Each row in Cities is checked
-- to make sure it satisfied the constraints.
-- if any rows don't satisfy the constraint, the
-- constraint is not added
ALTER TABLE CITIES ADD CONSTRAINT COUNTRY_FK
Foreign Key (COUNTRY) REFERENCES COUNTRIES (COUNTRY);

-- Add a primary key constraint to a table
-- First, create a new table
CREATE TABLE ACTIVITIES (CITY_ID INT NOT NULL,
SEASON CHAR(2), ACTIVITY VARCHAR(32) NOT NULL);
-- You will not be able to add this constraint if the
-- columns you are including in the primary key have
-- null data or duplicate values.
ALTER TABLE Activities ADD PRIMARY KEY (city_id, activity);

-- Drop the city_id column if there are no dependent objects:
ALTER TABLE Cities DROP COLUMN city_id RESTRICT;
-- Drop the city_id column, also dropping all dependent objects:
ALTER TABLE Cities DROP COLUMN city_id CASCADE;

-- Drop a primary key constraint from the CITIES table

ALTER TABLE Cities DROP CONSTRAINT Cities_PK;
-- Drop a foreign key constraint from the CITIES table
ALTER TABLE Cities DROP CONSTRAINT COUNTRIES_FK;
-- add a DEPTNO column with a default value of 1
ALTER TABLE SAMP.EMP_ACT ADD COLUMN DEPTNO INT DEFAULT 1;
-- increase the width of a VARCHAR column
ALTER TABLE SAMP.EMP_PHOTO ALTER PHOTO_FORMAT SET DATA TYPE VARCHAR(30);
-- change the lock granularity of a table
ALTER TABLE SAMP.SALES LOCKSIZE TABLE;

-- Remove the NOT NULL constraint from the MANAGER column
ALTER TABLE Employees ALTER COLUMN Manager NULL;
-- Add the NOT NULL constraint to the SSN column
ALTER TABLE Employees ALTER COLUMN ssn NOT NULL;

-- Change the default value for the SALARY column
ALTER TABLE Employees ALTER COLUMN Salary DEFAULT 1000.0
ALTER TABLE Employees ALTER COLUMN Salary DROP DEFAULT

-- Enable CYCLE behavior for the message_id column
ALTER TABLE messages ALTER COLUMN message_id SET CYCLE

-- Bulk load a table by temporarily changing a GENERATED ALWAYS identity column
-- into a GENERATED BY default column.
-- After loading the table, reset the identity column to be GENERATED ALWAYS
-- and move its sequence number forward past the last inserted key.
ALTER TABLE targetTable ALTER COLUMN keyCol SET GENERATED BY DEFAULT;
INSERT INTO targetTable SELECT * FROM sourceTable;
ALTER TABLE targetTable ALTER COLUMN keyCol SET GENERATED ALWAYS;
ALTER TABLE targetTable ALTER COLUMN keyCol RESTART WITH 1234567;
```

Results

An ALTER TABLE statement causes all statements that are dependent on the table being altered to be recompiled before their next execution. ALTER TABLE is not allowed if there are any open cursors that reference the table being altered.

Parent topic: Statements