**Create database**

The syntax of the SQL CREATE DATABASE statement is:

CREATE DATABASE DB\_NAME;

* CREATE DATABASE command creates a database
* DB\_NAME is the name of the database created

**CREATE DATABASE IF NOT EXISTS**

If a database already exists, SQL will throw an error while creating another database of the same name.

In such situations, we can use the CREATE DATABASE IF NOT EXISTS statement to create a database only if there is no existing database with the same name. For example,

CREATE DATABASE IF NOT EXISTS my\_db;

Here, the SQL command creates a database named my\_db only if there is no existing database with the same name.

**List all Databases**

There could be multiple databases in a database management system. To show the list of databases, we can run the following statement.

SHOW DATABASES;

Here, the SQL command lists all the available databases in the DBMS.

**Switch Databases**

We often have to work with multiple databases. To switch between available databases, we can run the following statement.

USE my\_db;

This code selects the my\_db database, and all SQL operations will be performed inside this database.

# CREATE ROLE statement

The CREATE ROLE statement allows you to create an SQL role.

Only the [database owner](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefattrib26867.html#rrefattrib26867) can create a role.

For more information on roles, see "Using SQL roles" in the Java DB Developer's Guide.

## Syntax

**CREATE ROLE** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename)

Before you issue a CREATE ROLE statement, verify that the *[derby.database.sqlAuthorization](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefpropersqlauth.html" \l "rrefpropersqlauth)* property is set to TRUE. The derby.database.sqlAuthorization property enables SQL authorization mode.

You cannot create a role name if there is a user by that name. An attempt to create a role name that conflicts with an existing user name raises the SQLException X0Y68.

If user names are not controlled by the database owner (or administrator), it may be a good idea to use a naming convention for roles to reduce the possibility of collision with user names.

Derby tries to avoid name collision between user names and role names, but this is not always possible, because Derby has a pluggable authorization architecture. For example, an externally defined user may exist who has never yet connected to the database, created any schema objects, or been granted any privileges. If Derby knows about a user name, it will forbid creating a role with that name. Correspondingly, a user who has the same name as a role will not be allowed to connect. Derby built-in users are checked for collision when a role is created.

A role name cannot start with the prefix SYS (after case normalization). The purpose of this restriction is to reserve a name space for system-defined roles at a later point. Use of the prefix SYS raises the SQLException 4293A.

You cannot create a role with the name PUBLIC (after case normalization). PUBLIC is a reserved authorization identifier. An attempt to create a role with the name PUBLIC raises SQLException 4251B.

## Example of creating a role

**CREATE ROLE purchases\_reader;**

## Examples of invalid role names

**CREATE ROLE public; -- throws SQLException;**

**CREATE ROLE "PUBLIC"; -- throws SQLException;**

**CREATE ROLE sysrole; -- throws SQLException;**

## Example of creating a role using a naming convention

The following example uses the convention of giving every role name the suffix \_role.

**CREATE ROLE purchases\_reader\_role;**

# **Create a New Oracle User and Grant Privileges: Syntax and Examples**

In this chapter, we will talk about how to create a user in Oracle. You will learn how to add new database users, figure out which supplemental aspects this job involves: from the initial user creation to dropping it. Moreover, you will find some useful tips on working with IDENTIFY and TABLESPACE clauses, as well as learn how to GRANT roles and permissions in Oracle.

Before we start, you need to check if you have the necessary system privilege to create users. If not, make sure to get them assigned to your account. After that, you can proceed to the practical tasks. The examples in this article relate to the create user Oracle 19c version, but the methods are the same for all Oracle versions in use (including Oracle 10g, 11g, 12c, etc.).

## Oracle CREATE USER Syntax Examples

For starters, we will be looking into Oracle CREATE USER syntax. First, we will discuss how to create one with default settings. After that, we will move on to the different variations of the IDENTIFIED clause, tablespace clause, and other peculiarities of the CREATE USER syntax in Oracle.

## How to Create Default Users with Default Settings

It is always best to start with the basics. Thus, let us focus on the CREATE USER command by itself. As is, it will create a user with default attributes. Further in this article, we will look at how to configure users more finely and how it boosts the safety of the database in general.

## Create User Identified by Clauses

The IDENTIFIED clause lets you indicate how the Oracle database authenticates a user. Let us take a closer look at different examples of the IDENTIFIED syntax in Oracle.

### **Create User Identified by Password Clause**

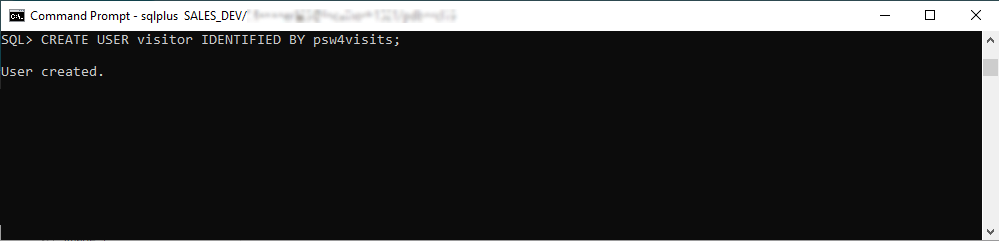
In the most straightforward case, we are creating a new local user under the username. The user will be required to enter the password to log into the system:

CREATE USER <username> IDENTIFIED BY <password>;

The username can be anything. However, the password must consist of single-byte characters from the database character set. If the character set also has multibyte characters, it does not change the password requirement – use only single-byte characters.

CREATE USER visitor

IDENTIFIED BY psw4visits;



### **Externally and Globally Clauses**

Besides identifying by password, you may use one of the two other means of user authentication. It will be configuring an external user or a global user. To do it, you need to include the EXTERNALLY or GLOBALLY clause in the CREATE USER Oracle command.

EXTERNALLY allows for creating an external user. In this case, the user is authenticated by an external system, such as the operating system. For instance, an Oracle database user is a Windows user. Thus, they can access the database after getting authenticated by Windows without entering other passwords. Working under the external user is a standard option for regular database users. But such users only have standard roles (CONNECT and RESOURCE), without administrator or database operator privileges.

To create an external user, we execute the below statement:

CREATE USER external\_user1

IDENTIFIED EXTERNALLY

DEFAULT TABLESPACE tbs\_new\_10

QUOTA 10M ON tbs\_new\_10

PROFILE external\_user\_profile1;

This way, we have made a new external user for our database. The name is external\_user1. No additional password is needed. We assigned this user the default tablespace tbs\_new\_10 with a quota of 10 Mb. Other limitations are defined by the external\_user\_profile1 applied to this user.

As we mentioned earlier, different external systems can maintain and manage external users in the Oracle database. Using the capabilities of the operating system is the most common option. Thus, if we want to create an external database user accessible by the system account in the operating system, we only need to modify our statement slightly. We’ll add the ops$ prefix to the username:

CREATE USER ops$external\_user1

IDENTIFIED EXTERNALLY

DEFAULT TABLESPACE tbs\_new\_10

QUOTA 10M ON tbs\_new\_10

PROFILE external\_user\_profile1;

GLOBALLY allows for creating global users. It means that their logins and passwords are stored on the Central Oracle Security Server instead of the specific database. Besides, roles assigned to global users on that central Server apply to this user in any database. It won’t be necessary to configure the user role in a separate database. Note that you need to enable the single sign-on option for global users.

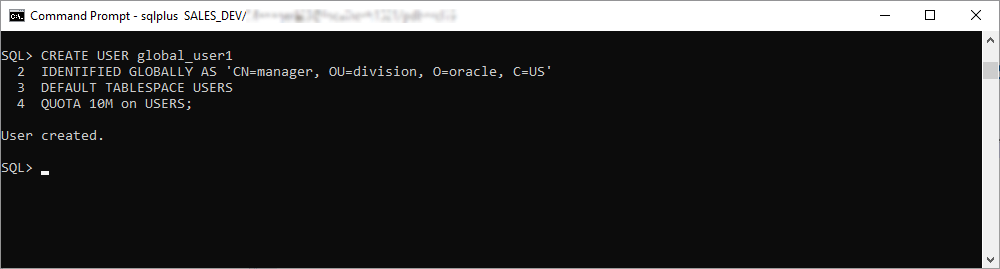
To create a global database user, we use the following statement:

CREATE USER global\_user1

IDENTIFIED GLOBALLY AS 'CN=manager, OU=division, O=oracle, C=US'

DEFAULT TABLESPACE USERS

QUOTA 10M on USERS;



Now we have a new global database user under the name of global\_user1. We assigned **USERS**default tablespace to that user with a quote of 10M.

## CREATE USER with Tablespace Clause

Now, let us review the basic Oracle create new user script. It is below:

CREATE USER username

IDENTIFIED BY password

DEFAULT TABLESPACE tablespace

TEMPORARY TABLESPACE tbs\_temp\_01

QUOTA {size | UNLIMITED} ON tablespace;

As you see, the script includes several clauses that we should take into consideration:

#### **Default Tablespace**

This clause specifies the default tablespace for objects created by the user. Otherwise, such objects are stored in the default tablespace of the database. If there are not any default tablespaces specified for this particular database, the objects will get into the system tablespace.

Restriction: don’t specify the locally managed temporary tablespace (such as undo tablespace or dictionary-managed temporary tablespace) to be the Oracle create user default tablespace.

#### **Temporary Tablespace**

This clause specifies the tablespace/tablespace group meant to contain the temporary segments of the user. Without it, those users’ temporary segments are stored in the default temporary tablespace of the database of the system tablespace. When you specify the tablespace group including the tablespace\_group\_name value in the script, users’ temporary segments can be saved in any tablespace of that group.

**Note*:***  
Make sure to specify the temporary tablespace with standard block size. It cannot be the undo tablespace or the tablespace with automatic segment-space management.

#### **Quota**

This clause specifies how much space this user can allocate in the tablespace. Multiple QUOTA clauses in one Oracle CREATE USER command can be present if you need to specify several tablespaces.  
The clause can include the UNLIMITED definition to allow this definite user to allocate the tablespace as much as needed, without bounds.

**Restriction**: the QUOTA clause does not apply to temporary tablespaces.

## Create User Attributes

There are additional, optional Oracle CREATE USER attributes you can include in the syntax. Have a look at the following example:

CREATE USER username

IDENTIFIED BY password

[DEFAULT TABLESPACE tablespace]

[QUOTA {size | UNLIMITED} ON tablespace]

[PROFILE profile]

[PASSWORD EXPIRE]

[ACCOUNT {LOCK | UNLOCK}];

Let us review these optional clauses.

#### **Profile**

This optional clause lets you limit the database resources for this specific user at once when the limitations are defined in the particular profile. Without this clause, a new user automatically comes under the default profile.

#### **Password Expire**

The clause is optional, but many database administrators set it for more effective security. If included, this clause will determine the forced change of the password on the user’s side. Usually, it happens when the user tries to log into the database for the first time.

#### **Account Lock/Account Unlock**

You may use one of these clauses. With LOCK applied, Oracle creates the user account, but that account won’t have access to the database. If you apply the UNLOCK clause or don’t specify any of these two clauses, the account will be usable at once. The unlocked status is the default.

The CREATE USER statement with these additional parameters would be as follows:

CREATE USER visitor

IDENTIFIED BY migzw23ter

DEFAULT TABLESPACE tbs\_new\_10

QUOTA 50M ON tbs\_new\_10

TEMPORARY TABLESPACE tbs\_temp\_10

QUOTA 5M ON system

PROFILE qualified\_user

PASSWORD EXPIRE;

ACCOUNT UNLOCK

Here, the statement creates a new Oracle database user named visitor, with the password migzw23ter. This user is assigned the default tablespace tbs\_new\_10 with a quota of 50Mb. This user is also allowed to use the temporary tablespace tbs\_temp\_10.

## Grant Role to User

The first step is the creation of a user. The next one is to set the user’s rights. A newly created user is not allowed to do anything, even to connect to the database.

Working with Oracle databases inevitably includes the task of creating database users. There are the system user accounts that Oracle creates itself – hr, OE, sys, etc. These accounts have predefined configurations with rights and limitations. However, daily work will always require other users.

One of the DBA’s duties is to create additional database users. The job includes configuring the user accounts, setting privileges, and managing users according to the business goals.

## Granting Permission in Oracle

By using the GRANT command, you can provide the users with certain privileges and configure their roles according to your needs. In Oracle, you can grant your permission to others so that they can manipulate and manage the data in your database. GRANT is a very powerful statement with many possible options, but the core functionality is to manage the privileges of both users and roles throughout the database.

## GRANT Command Syntax

The basic syntax of the query to grant certain privileges to the user is the following:

GRANT <permission> to <user>;

## Oracle User Privileges

The GRANT command can give the users privileges to create, alter, drop and manage database objects. For instance, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges.

Oracle has more than 100 system privileges that can be found in the SYSTEM\_PRIVILEGE\_MAP table.

|  |  |
| --- | --- |
| **CLUSTER** | CREATE/CREATE ANY/ALTER ANY/DROP ANY CLUSTER |
| **DATABASE** | ALTER DATABASE, ALTER SYSTEM, AUDIT SYSTEM |
| **INDEX** | CREATE ANY/ALTER ANY/DROP ANY INDEX |
| **PROFILE** | CREATE/ALTER/DROP PROFILE |
| **ROLE** | CREATE/ALTER ANY/DROP ANY /GRANT ANY (allows REVOKE) |
| **Rollback Segment** | CREATE/ALTER/DROP ROLLBACK SEGMENT |
| **USER** | CREATE/ALTER/BECOME/DROP USER |
| **VIEW** | CREATE/CREATE ANY/DROP ANY VIEW |
| **SYNONYM** | CREATE/CREATE ANY/CREATE PUBLIC/DROP ANY/DROP PUBLIC SYNONYM |
| **SESSION** | CREATE/ALTER/RESTRICTED SESSION, ALTER RESOURCE COST |
| **TABLE** | CREATE/CREATE ANY/ALTER ANY/DROP ANY/SELECT ANY/INSERT ANY/UPDATE ANY/DELETE ANY/LOCK ANY TABLE |
| **TABLESPACE** | CREATE/ALTER/DROP/MANAGE TABLESPACE |

Usually, the administrator of a database grants the privileges to the users. However, there are cases when the administrator needs to transfer their Oracle user privileges. This is when DBA privileges come in. If a DBA needs to provide system privilege to another person, it has to be done with the admin option:

GRANT create session TO user;

GRANT create session TO user with admin option;

Revoke create session from user;

Besides the Oracle system privileges, object privileges are granted upon database objects: tables, views, procedures, and so on.

## How to Create and Grant All Privileges to Oracle User

First, we need to grant our users the system privilege to log into the database. We use the following statement for that:

GRANT CREATE SESSION to visitor;

There are many permissions the database administrator can provide to the user. But it is essential to stick to the primary concept of security, which is to give users the minimum of privileges necessary to do the job efficiently. That’s why it is not recommended to provide all privileges to the user.

You can apply other privileges one by one, each by a separate statement. Or, it is possible to combine these permissions into one, as shown below:

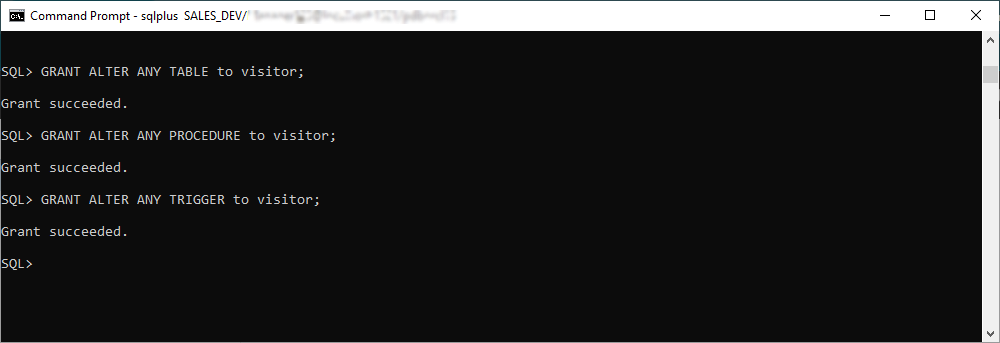
GRANT CREATE VIEW, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TRIGGER to visitor;

If this definite user is allowed to change tables, procedures, triggers, etc., the syntax to set the necessary privilege for each case is below. Again, be very careful when allowing the user to change any elements, as this permission is global.

GRANT ALTER ANY TABLE to visitor;

GRANT ALTER ANY PROCEDURE to visitor;

GRANT ALTER ANY TRIGGER to visitor;



To allow the user to delete elements, we use the below statements:

GRANT DELETE ANY TABLE to visitor;

GRANT DROP ANY PROCEDURE to visitor;

GRANT DROP ANY TRIGGER to visitor;

GRANT DROP ANY VIEW to visitor;

## How to Grant Table Privilege to User in Oracle

Before you set the privileges to the particular user, you should consider which tasks that person must perform in the database. The most common scenarios include creating tables, views, procedures, triggers. Some cases require the possibility to change or delete those elements. Depending on the situation, the administrator defines which system privileges to provide.

Let us take a closer look at how to grant CREATE TABLE privilege to a user in Oracle. If we are willing to allow our user – visitor – to create tables in the database, we will use the following query:

GRANT CREATE TABLE to visitor;

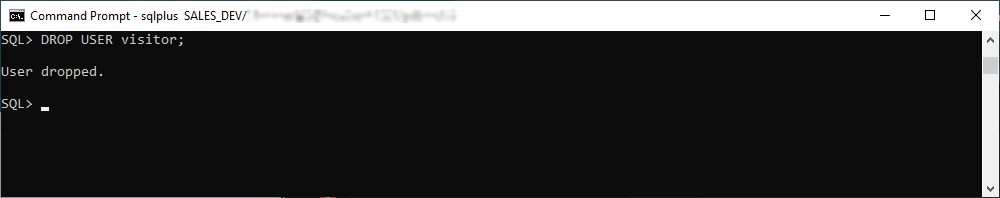
## How to Delete (Drop) User in Oracle

In case you need to remove any user for any reason, you should use the DROP USER command with the following syntax:

DROP USER <username>;

In our test case, we are removing the user visitor created earlier:

DROP USER visitor;



However, there are several restrictions that you need to pay attention to before dropping the user:

* You can’t remove users without deleting all the related objects. Thus, you must drop all tables, views, procedures, etc. that this user created before proceeding to the DROP command.
* You can’t remove users that are connected to the database. First, you have to clear up all sessions that the user had. After that, you can drop the user itself.

There is a special command that allows for dropping the user with all its database objects in one shot:

DROP USER <username> CASCADE;

**Create table**

In Oracle, CREATE TABLE statement is used to create a new table in the database.

To create a table, you have to name that table and define its columns and datatype for each column.

**Syntax:**

1. **CREATE** **TABLE** table\_name
2. (
3. column1 datatype [ NULL | NOT NULL ],
4. column2 datatype [ NULL | NOT NULL ],
5. ...
6. column\_n datatype [ NULL | NOT NULL ]
7. );

* **table\_name:**It specifies the name of the table which you want to create.
* **column1, column2, ... column n:** It specifies the columns which you want to add in the table. Every column must have a datatype. Every column should either be defined as "NULL" or "NOT NULL". In the case, the value is left blank; it is treated as "NULL" as default.

## Oracle CREATE TABLE Example

Here we are creating a table named customers. This table doesn't have any primary key.

**CREATE** **TABLE** customers

1. ( customer\_id number(10) NOT NULL,
2. customer\_name varchar2(50) NOT NULL,
3. city varchar2(50)
4. );

This table contains three columns

* **customer\_id:** It is the first column created as a number datatype (maximum 10 digits in length) and cannot contain null values.
* **customer\_name:** it is the second column created as a varchar2 datatype (50 maximum characters in length) and cannot contain null values.
* **city:**This is the third column created as a varchar2 datatype. It can contain null values.

## Oracle CREATE TABLE Example with primary key

1. **CREATE** **TABLE** customers
2. ( customer\_id number(10) NOT NULL,
3. customer\_name varchar2(50) NOT NULL,
4. city varchar2(50),
5. **CONSTRAINT** customers\_pk **PRIMARY** **KEY** (customer\_id)
6. );

## What is Primary key

A primary key is a single field or combination of fields that contains a unique record. It must be filled. None of the field of primary key can contain a null value. A table can have only one primary key.

#### **In Oracle, total number of columns cannot be more than 32.**

# **CREATE TABLE AS Statement**

The CREATE TABLE AS statement is used to create a table from an existing table by copying the columns of existing table.

#### **Note: If you create the table in this way, the new table will contain records from the existing table.**

**Syntax:**

1. **CREATE** **TABLE** new\_table
2. **AS** (**SELECT** \* **FROM** old\_table);

## Create Table Example: copying all columns of another table

In this example, we are creating a "newcustomers" table by copying all the columns from the already existing table "Customers".

1. **CREATE** **TABLE** newcustomers
2. **AS** (**SELECT** \*   **FROM** customers  **WHERE** customer\_id < 5000);

Table created.

This table is named as "newcustomers" and having the same columns of "customers" table.

## Create Table Example: copying selected columns of another table

**Syntax:**

1. **CREATE** **TABLE** new\_table
2. **AS** (**SELECT** column\_1, column2, ... column\_n
3. **FROM** old\_table);

Let's take an example:

1. **CREATE** **TABLE** newcustomers2
2. **AS** (**SELECT** customer\_id, customer\_name
3. **FROM** customers
4. **WHERE** customer\_id < 5000);

The above example will create a new table called "newcustomers2". This table includes the specified columns customer\_id and customer\_name from the customers table.

## Create Table Example: copying selected columns from multiple tables

**Syntax:**

1. **CREATE** **TABLE** new\_table
2. **AS** (**SELECT** column\_1, column2, ... column\_n
3. **FROM** old\_table\_1, old\_table\_2, ... old\_table\_n);

Let's take an example: Consider that you have already created two tables "regularcustomers" and "irregularcustomers".

The table "regularcustomers" has three columns rcustomer\_id, rcustomer\_name and rc\_city.

1. **CREATE** **TABLE**  "regularcustomers"
2. (    "RCUSTOMER\_ID" NUMBER(10,0) NOT NULL ENABLE,
3. "RCUSTOMER\_NAME" VARCHAR2(50) NOT NULL ENABLE,
4. "RC\_CITY" VARCHAR2(50)
5. )
6. /

The second table "irregularcustomers" has also three columns ircustomer\_id, ircustomer\_name and irc\_city.

1. **CREATE** **TABLE**  "irregularcustomers"
2. (    "IRCUSTOMER\_ID" NUMBER(10,0) NOT NULL ENABLE,
3. "IRCUSTOMER\_NAME" VARCHAR2(50) NOT NULL ENABLE,
4. "IRC\_CITY" VARCHAR2(50)
5. )
6. /

In the following example, we will create a table name "newcustomers3" form copying columns from both tables.

**Example:**

1. **CREATE** **TABLE** newcustomers3
2. **AS** (**SELECT** regularcustomers.rcustomer\_id, regularcustomers.rc\_city, irregularcustomers.ircustomer\_name
3. **FROM** regularcustomers, irregularcustomers
4. **WHERE** regularcustomers.rcustomer\_id = irregularcustomers.ircustomer\_id
5. AND regularcustomers.rcustomer\_id < 5000);

# LOCK TABLE statement

The LOCK TABLE statement allows you to explicitly acquire a shared or exclusive table lock on the specified table. The table lock lasts until the end of the current transaction.

To lock a table, you must either be the [database owner](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefattrib26867.html#rrefattrib26867) or the table owner.

Explicitly locking a table is useful to:

* Avoid the overhead of multiple row locks on a table (in other words, user-initiated lock escalation)
* Avoid deadlocks

You cannot lock system tables with this statement.

## Syntax

**LOCK TABLE** [***table-Name***](https://docs.oracle.com/javadb/10.8.3.0/ref/rreftablename.html#rreftablename) **IN { SHARE | EXCLUSIVE } MODE**

After a table is locked in either mode, a transaction does not acquire any subsequent row-level locks on a table. For example, if a transaction locks the entire Flights table in share mode in order to read data, a particular statement might need to lock a particular row in exclusive mode in order to update the row. However, the previous table-level lock on the Flights table forces the exclusive lock to be table-level as well.

If the specified lock cannot be acquired because another connection already holds a lock on the table, a statement-level exception is raised (SQLState X0X02) after the deadlock timeout period.

## Examples

To lock the entire Flights table in share mode to avoid a large number of row locks, use the following statement:

LOCK TABLE Flights IN SHARE MODE;

SELECT \*

FROM Flights

WHERE orig\_airport > 'OOO';

You have a transaction with multiple UPDATE statements. Since each of the individual statements acquires only a few row-level locks, the transaction will not automatically upgrade the locks to a table-level lock. However, collectively the UPDATE statements acquire and release a large number of locks, which might result in deadlocks. For this type of transaction, you can acquire an exclusive table-level lock at the beginning of the transaction. For example:

LOCK TABLE FlightAvailability IN EXCLUSIVE MODE;

UPDATE FlightAvailability

SET economy\_seats\_taken = (economy\_seats\_taken + 2)

WHERE flight\_id = 'AA1265' AND flight\_date = DATE('2004-03-31');

UPDATE FlightAvailability

SET economy\_seats\_taken = (economy\_seats\_taken + 2)

WHERE flight\_id = 'AA1265' AND flight\_date = DATE('2004-04-11');

UPDATE FlightAvailability

SET economy\_seats\_taken = (economy\_seats\_taken + 2)

WHERE flight\_id = 'AA1265' AND flight\_date = DATE('2004-04-12');

UPDATE FlightAvailability

SET economy\_seats\_taken = (economy\_seats\_taken + 2)

WHERE flight\_id = 'AA1265' AND flight\_date = DATE('2004-04-15');

If a transaction needs to look at a table before updating the table, acquire an exclusive lock before selecting to avoid deadlocks. For example:

LOCK TABLE Maps IN EXCLUSIVE MODE;

SELECT MAX(map\_id) + 1 FROM Maps;

-- INSERT INTO Maps . . .

Create VIEW

# **Oracle View**

In Oracle, view is a virtual table that does not physically exist. It is stored in Oracle data dictionary and do not store any data. It can be executed when called.

A view is created by a query joining one or more tables.

## Oracle CREATE VIEW

**Syntax:**

1. **CREATE** **VIEW** view\_name **AS**
2. **SELECT** columns
3. **FROM** tables
4. **WHERE** conditions;

* **view\_name:** It specifies the name of the Oracle VIEW that you want to create.

**Example:**

Let's take an example to create view. In this example, we are creating two tables suppliers and orders first.

**Suppliers table:**

2. **CREATE** **TABLE**  "SUPPLIERS"
3. (    "SUPPLIER\_ID" NUMBER,
4. "SUPPLIER\_NAME" VARCHAR2(4000),
5. "SUPPLIER\_ADDRESS" VARCHAR2(4000)
6. )
7. /

**Orders table:**

1. **CREATE** **TABLE**  "ORDERS"
2. (    "ORDER\_NO." NUMBER,
3. "QUANTITY" NUMBER,
4. "PRICE" NUMBER
5. )
6. /

Execute the following query to create a view name sup\_orders.

**Create View Query:**

1. **CREATE** **VIEW** sup\_orders **AS**
2. **SELECT** suppliers.supplier\_id, orders.quantity, orders.price
3. **FROM** suppliers
4. **INNER** JOIN orders
5. **ON** suppliers.supplier\_id = supplier\_id
6. **WHERE** suppliers.supplier\_name = 'VOJO';

**Output:**

View created.

0.21 seconds

You can now check the Oracle VIEW by this query:

1. **SELECT** \* **FROM** sup\_orders;

**Output:**

SUPPLIER\_ID QUANTITY PRICE

3 35 70

3 26 125

3 18 100

3 rows returned in 0.00 seconds

## Oracle Update VIEW

In Oracle, the CREATE OR REPLACE VIEW statement is used to modify the definition of an Oracle VIEW without dropping it.

**Syntax:**

1. **CREATE** OR REPLACE **VIEW** view\_name **AS**
2. **SELECT** columns
3. **FROM** **table**
4. **WHERE** conditions;

**Example:**

Execute the following query to update the definition of Oracle VIEW called sup\_orders without dropping it.

1. **CREATE** or REPLACE **VIEW** sup\_orders **AS**
2. **SELECT** suppliers.supplier\_id, orders.quantity, orders.price
3. **FROM** suppliers
4. **INNER** JOIN orders
5. **ON** suppliers.supplier\_id = supplier\_id
6. **WHERE** suppliers.supplier\_name = 'HCL';

You can now check the Oracle VIEW by this query:

1. **SELECT** \* **FROM** sup\_orders;

Output:

SUPPLIER\_ID QUANTITY PRICE

1 35 70

1 26 125

1 18 100

row(s) 1 - 3 of 3

## Oracle DROP VIEW

The DROP VIEW statement is used to remove or delete the VIEW completely.

**Syntax:**

1. **DROP** **VIEW** view\_name;

**Example:**

1. **DROP** **VIEW** sup\_orders;

Alter

In Oracle, ALTER TABLE statement specifies how to add, modify, drop or delete columns in a table. It is also used to rename a table.

How to add column in a table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. **ADD** column\_name **column**-definition;

**Example:**

Consider that already existing table customers. Now, add a new column customer\_age into the table customers.

1. **ALTER** **TABLE** customers
2. **ADD** customer\_age varchar2(50);

Now, a new column "customer\_age" will be added in customers table.

How to add multiple columns in the existing table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. **ADD** (column\_1 **column**-definition,
3. column\_2 **column**-definition,
4. ...
5. column\_n column\_definition);

**Example**

1. **ALTER** **TABLE** customers
2. **ADD** (customer\_type varchar2(50),
3. customer\_address varchar2(50));

Now, two columns customer\_type and customer\_address will be added in the table customers.

How to modify column of a table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. **MODIFY** column\_name column\_type;

**Example:**

1. **ALTER** **TABLE** customers
2. **MODIFY** customer\_name varchar2(100) not null;

Now the column column\_name in the customers table is modified

to varchar2 (100) and forced the column to not allow null values.

How to modify multiple columns of a table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. **MODIFY** (column\_1 column\_type,
3. column\_2 column\_type,
4. ...
5. column\_n column\_type);

**Example:**

1. **ALTER** **TABLE** customers
2. **MODIFY** (customer\_name varchar2(100) not null,
3. city varchar2(100));

This will modify both the customer\_name and city columns in the table.

How to drop column of a table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. **DROP** **COLUMN** column\_name;

**Example:**

1. **ALTER** **TABLE** customers
2. **DROP** **COLUMN** customer\_name;

This will drop the customer\_name column from the table.

How to rename column of a table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. RENAME **COLUMN** old\_name **to** new\_name;

**Example:**

1. **ALTER** **TABLE** customers
2. RENAME **COLUMN** customer\_name **to** cname;

This will rename the column customer\_name into cname.

How to rename table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. RENAME **TO** new\_table\_name;

**Example:**

1. **ALTER** **TABLE** customers
2. RENAME **TO** retailers;

This will rename the customer table into "retailers" table.

Drop

Oracle DROP TABLE statement is used to remove or delete a table from the Oracle database.

**Syntax**

1. **DROP** [schema\_name].**TABLE** table\_name
2. [ **CASCADE** CONSTRAINTS ]
3. [ PURGE ];

### **Parameters**

**schema\_name:**It specifies the name of the schema that owns the table.

**table\_name:** It specifies the name of the table which you want to remove from the Oracle database.

**CASCADE CONSTRAINTS:**It is optional. If specified, it will drop all referential integrity constraints as well.

**PURGE:**It is also optional. If specified, the table and its dependent objects are placed in the recycle bin and can’t be recovered.

#### **If there are referential integrity constraints on table\_name and you do not specify the CASCADE CONSTRAINTS option, the DROP TABLE statement will return an error and Oracle will not drop the table.**

## DROP TABLE Example

1. **DROP** **TABLE** customers;

This will drop the table named customers.

## DROP TABLE Example with PURGE parameter

1. **DROP** **TABLE** customers PURGE

This statement will drop the table called customers and issue a PURGE so that the space associated with the customers table is released and the customers table is not placed in recycle bin. So, it is not possible to recover that table if required.

Grant

Use the GRANT statement to give privileges to a specific user or role, or to all users, to perform actions on database objects. You can also use the GRANT statement to grant a role to a user, to PUBLIC, or to another role.

Privileges are granted to users and/or roles, where any user can be assigned to one or more roles. The *public* and *admin* roles have special meaning:

* The *public* role includes all users. This role can be used in a GRANT statement to assign a privilege to every user.
* The *admin* role is required to run certain commands, such as the ADD JAR command, and it also allows access to certain objects even where access may not have been explicitly granted to the user.

Syntax: GRANT

priv\_type [, priv\_type ] ...

ON table\_or\_view\_name

TO principal\_specification [, principal\_specification] ...

[WITH GRANT OPTION];

The following types of privileges can be granted:

* Delete data from a specific table.
* Insert data into a specific table.
* Create a foreign key reference to the named table or to a subset of columns from a table.
* Select data from a table, view, or a subset of columns in a table.
* Create a trigger on a table.
* Update data in a table or in a subset of columns in a table.
* Run a specified function or procedure.
* Use a sequence generator or a user-defined type.

Before you issue a GRANT statement, check that the [derby.database.sqlAuthorization](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefpropersqlauth.html" \l "rrefpropersqlauth) property is set to true. The derby.database.sqlAuthorization property enables the SQL Authorization mode.

You can grant privileges on an object if you are the owner of the object or the [database owner](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefattrib26867.html#rrefattrib26867). See the CREATE statement for the database object that you want to grant privileges on for more information.

The syntax that you use for the GRANT statement depends on whether you are granting privileges to a schema object or granting a role.

For more information on using the GRANT statement, see "Using SQL standard authorization" in the Java DB Developer's Guide.

## Syntax for tables

**GRANT** [***privilege-type***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljgrant.html#rrefsqljgrant__grantprivtype) **ON [TABLE] {** [***table-Name***](https://docs.oracle.com/javadb/10.8.3.0/ref/rreftablename.html#rreftablename) **|** [***view-Name***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefviewname.html#rrefviewname) **} TO** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljgrant.html#rrefsqljgrant__grantgrantees)

## Syntax for routines

**GRANT EXECUTE ON { FUNCTION | PROCEDURE }** [***routine-designator***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljgrant.html#rrefsqljgrant__grantroutinename) **TO** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljgrant.html#rrefsqljgrant__grantgrantees)

## Syntax for sequence generators

**GRANT USAGE ON SEQUENCE [** [***schemaName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefschemaname.html#rrefschemaname)**. ]** [***SQL92Identifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/crefsqlj34834.html#crefsqlj34834) **TO** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljgrant.html#rrefsqljgrant__grantgrantees)

In order to use a sequence generator, you must have the USAGE privilege on it. This privilege can be granted to users and to roles. See [CREATE SEQUENCE statement](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljcreatesequence.html#rrefsqljcreatesequence) for more information.

The sequence name is composed of an optional schemaName and a SQL92Identifier. If a schemaName is not provided, the current schema is the default schema. If a qualified sequence name is specified, the schema name cannot begin with SYS.

## Syntax for user-defined types

**GRANT USAGE ON TYPE [** [***schemaName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefschemaname.html#rrefschemaname)**. ]** [***SQL92Identifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/crefsqlj34834.html#crefsqlj34834) **TO** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljgrant.html#rrefsqljgrant__grantgrantees)

In order to use a user-defined type, you must have the USAGE privilege on it. This privilege can be granted to users and to roles. See [CREATE TYPE statement](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljcreatetype.html#rrefsqljcreatetype) for more information.

The type name is composed of an optional schemaName and a SQL92Identifier. If a schemaName is not provided, the current schema is the default schema. If a qualified type name is specified, the schema name cannot begin with SYS.

## Syntax for roles

**GRANT** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **[ {,** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **}\* ] TO** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljgrant.html#rrefsqljgrant__grantgrantees)

Before you can grant a role to a user or to another role, you must create the role using the [CREATE ROLE statement](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefcreaterole.html#rrefcreaterole). Only the [database owner](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefattrib26867.html#rrefattrib26867) can grant a role.

A role A contains another role B if role B is granted to role A, or is contained in a role C granted to role A. Privileges granted to a contained role are inherited by the containing roles. So the set of privileges identified by role A is the union of the privileges granted to role A and the privileges granted to any contained roles of role A.

## privilege-types

**ALL PRIVILEGES |**

**privilege-list**

## privilege-list

**table-privilege {, table-privilege }\***

## table-privilege

**DELETE |**

**INSERT |**

**REFERENCES [column list] |**

**SELECT [column list] |**

**TRIGGER |**

**UPDATE [column list]**

## column list

**( column-identifier {, column-identifier}\* )**

Use the ALL PRIVILEGES privilege type to grant all of the privileges to the user or role for the specified table. You can also grant one or more table privileges by specifying a privilege-list.

Use the DELETE privilege type to grant permission to delete rows from the specified table.

Use the INSERT privilege type to grant permission to insert rows into the specified table.

Use the REFERENCES privilege type to grant permission to create a foreign key reference to the specified table. If a column list is specified with the REFERENCES privilege, the permission is valid on only the foreign key reference to the specified columns.

Use the SELECT privilege type to grant permission to perform [SELECT statements](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqlj41360.html#rrefsqlj41360) or *[SelectExpressions](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefselectexpression.html" \l "rrefselectexpression)* on a table or view. If a column list is specified with the SELECT privilege, the permission is valid on only those columns. If no column list is specified, then the privilege is valid on all of the columns in the table.

For queries that do not select a specific column from the tables involved in a SELECT statement or SelectExpression (for example, queries that use COUNT(\*)), the user must have at least one column-level SELECT privilege or table-level SELECT privilege.

Use the TRIGGER privilege type to grant permission to create a trigger on the specified table.

Use the UPDATE privilege type to grant permission to use the UPDATE statement on the specified table. If a column list is specified, the permission applies only to the specified columns. To update a row using a statement that includes a WHERE clause, you must have the SELECT privilege on the columns in the row that you want to update.

## grantees

**{** [***AuthorizationIdentifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrauthid.html#rrefrauthid) **|** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **| PUBLIC }**

**[, {** [***AuthorizationIdentifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrauthid.html#rrefrauthid) **|** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **| PUBLIC } ] \***

You can grant privileges or roles to specific users or roles or to all users. Use the keyword PUBLIC to specify all users. When PUBLIC is specified, the privileges or roles affect all current and future users. The privileges granted to PUBLIC and to individual users or roles are independent privileges. For example, a SELECT privilege on table t is granted to both PUBLIC and to the authorization ID harry. The SELECT privilege is later revoked from the authorization ID harry, but Harry can access the table t through the PUBLIC privilege.

Either the object owner or the database owner can grant privileges to a user or to a role. Only the database owner can grant a role to a user or to another role.

## routine-designator

**{**

**function-name | procedure-name**

**}**

## Examples

To grant the SELECT privilege on table t to the authorization IDs maria and harry, use the following syntax:

**GRANT SELECT ON TABLE t TO maria,harry**

To grant the UPDATE and TRIGGER privileges on table t to the authorization IDs anita and zhi, use the following syntax:

**GRANT UPDATE, TRIGGER ON TABLE t TO anita,zhi**

To grant the SELECT privilege on table s.v to all users, use the following syntax:

**GRANT SELECT ON TABLE s.v to PUBLIC**

To grant the EXECUTE privilege on procedure p to the authorization ID george, use the following syntax:

**GRANT EXECUTE ON PROCEDURE p TO george**

To grant the role purchases\_reader\_role to the authorization IDs george and maria, use the following syntax:

**GRANT purchases\_reader\_role TO george,maria**

To grant the SELECT privilege on table t to the role purchases\_reader\_role, use the following syntax:

**GRANT SELECT ON TABLE t TO purchases\_reader\_role**

To grant the USAGE privilege on the sequence generator order\_id to the role sales\_role, use the following syntax:

**GRANT USAGE ON SEQUENCE order\_id TO sales\_role;**

To grant the USAGE privilege on the user-defined type price to the role finance\_role, use the following syntax:

**GRANT USAGE ON TYPE price TO finance\_role;**

Revoke

# REVOKE statement

Use the REVOKE statement to remove privileges from a specific user or role, or from all users, to perform actions on database objects. You can also use the REVOKE statement to revoke a role from a user, from PUBLIC, or from another role.

Syntax: REVOKE [GRANT OPTION FOR]

priv\_type [, priv\_type ] ...

ON table\_or\_view\_name

FROM principal\_specification [, principal\_specification] ... ;

principal\_specification

: USER user

| ROLE role

priv\_type

: INSERT | SELECT | UPDATE | DELETE | ALL

The following types of privileges can be revoked:

* Delete data from a specific table.
* Insert data into a specific table.
* Create a foreign key reference to the named table or to a subset of columns from a table.
* Select data from a table, view, or a subset of columns in a table.
* Create a trigger on a table.
* Update data in a table or in a subset of columns in a table.
* Run a specified routine (function or procedure).
* Use a sequence generator or a user-defined type.

The [derby.database.sqlAuthorization](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefpropersqlauth.html" \l "rrefpropersqlauth) property must be set to true before you can use the GRANT statement or the REVOKE statement. The derby.database.sqlAuthorization property enables SQL Authorization mode.

You can revoke privileges for an object if you are the owner of the object or the [database owner](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefattrib26867.html#rrefattrib26867).

The syntax that you use for the REVOKE statement depends on whether you are revoking privileges to a schema object or revoking a role.

For more information on using the REVOKE statement, see "Using SQL standard authorization" in the Java DB Developer's Guide.

## Syntax for tables

**REVOKE** [***privilege-type***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljrevoke.html#rrefsqljrevoke__revokeprivtype) **ON [ TABLE ] {** [***table-Name***](https://docs.oracle.com/javadb/10.8.3.0/ref/rreftablename.html#rreftablename) **|** [***view-Name***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefviewname.html#rrefviewname) **} FROM** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljrevoke.html#rrefsqljrevoke__revokegrantees)

Revoking a privilege without specifying a column list revokes the privilege for all of the columns in the table.

## Syntax for routines

**REVOKE EXECUTE ON { FUNCTION | PROCEDURE }** [***routine-designator***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljrevoke.html#rrefsqljrevoke__revokeroutinename) **FROM** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljrevoke.html#rrefsqljrevoke__revokegrantees) **RESTRICT**

You must use the RESTRICT clause on REVOKE statements for routines. The RESTRICT clause specifies that the EXECUTE privilege cannot be revoked if the specified routine is used in a view, trigger, or constraint, and the privilege is being revoked from the owner of the view, trigger, or constraint.

## Syntax for sequence generators

**REVOKE USAGE ON SEQUENCE [** [***schemaName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefschemaname.html#rrefschemaname)**. ]** [***SQL92Identifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/crefsqlj34834.html#crefsqlj34834) **FROM** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljrevoke.html#rrefsqljrevoke__revokegrantees) **RESTRICT**

In order to use a sequence generator, you must have the USAGE privilege on it. This privilege can be revoked from users and roles. Only RESTRICTed revokes are allowed. This means that the REVOKE statement cannot make a view, trigger, or constraint unusable by its owner. The USAGE privilege cannot be revoked from the schema owner. See [CREATE SEQUENCE statement](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljcreatesequence.html#rrefsqljcreatesequence) for more information.

The sequence name is composed of an optional schemaName and a SQL92Identifier. If a schemaName is not provided, the current schema is the default schema. If a qualified sequence name is specified, the schema name cannot begin with SYS.

## Syntax for user-defined types

**REVOKE USAGE ON TYPE [** [***schemaName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefschemaname.html#rrefschemaname)**. ]** [***SQL92Identifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/crefsqlj34834.html#crefsqlj34834) **FROM** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljrevoke.html#rrefsqljrevoke__revokegrantees) **RESTRICT**

In order to use a user-defined type, you must have the USAGE privilege on it. This privilege can be revoked from users and roles. Only RESTRICTed revokes are allowed. This means that the REVOKE statement cannot make a view, trigger, or constraint unusable by its owner. The USAGE privilege cannot be revoked from the schema owner. See [CREATE TYPE statement](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljcreatetype.html#rrefsqljcreatetype) for more information.

The type name is composed of an optional schemaName and a SQL92Identifier. If a schemaName is not provided, the current schema is the default schema. If a qualified type name is specified, the schema name cannot begin with SYS.

## Syntax for roles

**REVOKE** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **[ {,** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **}\* ] FROM** [***grantees***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqljrevoke.html#rrefsqljrevoke__revokegrantees)

Only the [database owner](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefattrib26867.html#rrefattrib26867) can revoke a role.

## privilege-types

**ALL PRIVILEGES |**

**privilege-list**

## privilege-list

**table-privilege {, table-privilege }\***

## table-privilege

**DELETE |**

**INSERT |**

**REFERENCES [column list] |**

**SELECT [column list] |**

**TRIGGER |**

**UPDATE [column list]**

## column list

**( column-identifier {, column-identifier}\* )**

Use the ALL PRIVILEGES privilege type to revoke all of the privileges from the user or role for the specified table. You can also revoke one or more table privileges by specifying a privilege-list.

Use the DELETE privilege type to revoke permission to delete rows from the specified table.

Use the INSERT privilege type to revoke permission to insert rows into the specified table.

Use the REFERENCES privilege type to revoke permission to create a foreign key reference to the specified table. If a column list is specified with the REFERENCES privilege, the permission is revoked on only the foreign key reference to the specified columns.

Use the SELECT privilege type to revoke permission to perform SELECT statements on a table or view. If a column list is specified with the SELECT privilege, the permission is revoked on only those columns. If no column list is specified, then the privilege is valid on all of the columns in the table.

Use the TRIGGER privilege type to revoke permission to create a trigger on the specified table.

Use the UPDATE privilege type to revoke permission to use the UPDATE statement on the specified table. If a column list is specified, the privilege is revoked only on the specified columns.

## grantees

**{** [***AuthorizationIdentifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrauthid.html#rrefrauthid) **|** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **| PUBLIC }**

**[,{** [***AuthorizationIdentifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrauthid.html#rrefrauthid) **|** [***roleName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefrolename.html#rrefrolename) **| PUBLIC } ] \***

You can revoke the privileges from specific users or roles or from all users. Use the keyword PUBLIC to specify all users. The privileges revoked from PUBLIC and from individual users or roles are independent privileges. For example, a SELECT privilege on table t is granted to both PUBLIC and to the authorization ID harry. The SELECT privilege is later revoked from the authorization ID harry, but the authorization ID harry can access the table t through the PUBLIC privilege.

You can revoke a role from a role, from a user, or from PUBLIC.

**Restriction:**You cannot revoke the privileges of the owner of an object.

## routine-designator

**{**

**qualified-name [ signature ]**

**}**

## sequenceName

**[** [***schemaName***](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefschemaname.html#rrefschemaname)**. ]** [***SQL92Identifier***](https://docs.oracle.com/javadb/10.8.3.0/ref/crefsqlj34834.html#crefsqlj34834)

If schemaName is not provided, the current schema is the default schema. If a qualified sequence name is specified, the schema name cannot begin with SYS.

## Prepared statements and open result sets/cursors

Checking for privileges happens at statement execution time, so prepared statements are still usable after a revoke action. If sufficient privileges are still available for the session, prepared statements will be executed, and for queries, a result set will be returned.

Once a result set has been returned to the application (by executing a prepared statement or by direct execution), it will remain accessible even if privileges or roles are revoked in a way that would cause another execution of the same statement to fail.

## Cascading object dependencies

For views, triggers, and constraints, if the privilege on which the object depends on is revoked, the object is automatically dropped. Derby does not try to determine if you have other privileges that can replace the privileges that are being revoked. For more information, see "Using SQL standard authorization" and "Privileges on views, triggers, and constraints" in the Java DB Developer's Guide.

## Limitations

The following limitations apply to the REVOKE statement:

**Table-level privileges**

All of the table-level privilege types for a specified grantee and table ID are stored in one row in the SYSTABLEPERMS system table. For example, when user2 is granted the SELECT and DELETE privileges on table user1.t1, a row is added to the SYSTABLEPERMS table. The GRANTEE field contains user2 and the TABLEID contains user1.t1. The SELECTPRIV and DELETEPRIV fields are set to Y. The remaining privilege type fields are set to N.

When a grantee creates an object that relies on one of the privilege types, the Derby engine tracks the dependency of the object on the specific row in the SYSTABLEPERMS table. For example, user2 creates the view v1 by using the statement SELECT \* FROM user1.t1, the dependency manager tracks the dependency of view v1 on the row in SYSTABLEPERMS for GRANTEE(user2), TABLEID(user1.t1). The dependency manager knows only that the view is dependent on a privilege type in that specific row, but does not track exactly which privilege type the view is dependent on.

When a REVOKE statement for a table-level privilege is issued for a grantee and table ID, all of the objects that are dependent on the grantee and table ID are dropped. For example, if user1 revokes the DELETE privilege on table t1 from user2, the row in SYSTABLEPERMS for GRANTEE(user2), TABLEID(user1.t1) is modified by the REVOKE statement. The dependency manager sends a revoke invalidation message to the view user2.v1 and the view is dropped even though the view is not dependent on the DELETE privilege for GRANTEE(user2), TABLEID(user1.t1).

**Column-level privileges**

Only one type of privilege for a specified grantee and table ID are stored in one row in the SYSCOLPERMS system table. For example, when user2 is granted the SELECT privilege on table user1.t1 for columns c12 and c13, a row is added to the SYSCOLPERMS. The GRANTEE field contains user2, the TABLEID contains user1.t1, the TYPE field contains S, and the COLUMNS field contains c12, c13.

When a grantee creates an object that relies on the privilege type and the subset of columns in a table ID, the Derby engine tracks the dependency of the object on the specific row in the SYSCOLPERMS table. For example, user2 creates the view v1 by using the statement SELECT c11 FROM user1.t1, the dependency manager tracks the dependency of view v1 on the row in SYSCOLPERMS for GRANTEE(user2), TABLEID(user1.t1), TYPE(S). The dependency manager knows that the view is dependent on the SELECT privilege type, but does not track exactly which columns the view is dependent on.

When a REVOKE statement for a column-level privilege is issued for a grantee, table ID, and type, all of the objects that are dependent on the grantee, table ID, and type are dropped. For example, if user1 revokes the SELECT privilege on column c12 on table user1.t1 from user2, the row in SYSCOLPERMS for GRANTEE(user2), TABLEID(user1.t1), TYPE(S) is modified by the REVOKE statement. The dependency manager sends a revoke invalidation message to the view user2.v1 and the view is dropped even though the view is not dependent on the column c12 for GRANTEE(user2), TABLEID(user1.t1), TYPE(S).

**Roles**

Derby tracks any dependencies on the definer's current role for views, constraints, and triggers. If privileges were obtainable only via the current role when the object in question was defined, that object depends on the current role. The object will be dropped if the role is revoked from the defining user or from PUBLIC, as the case may be. Also, if a contained role of the current role in such cases is revoked, dependent objects will be dropped. Note that dropping may be too pessimistic. This is because Derby does not currently make an attempt to recheck if the necessary privileges are still available in such cases.

## Revoke examples

To revoke the SELECT privilege on table t from the authorization IDs maria and harry, use the following syntax:

**REVOKE SELECT ON TABLE t FROM maria,harry**

To revoke the UPDATE and TRIGGER privileges on table t from the authorization IDs anita and zhi, use the following syntax:

**REVOKE UPDATE, TRIGGER ON TABLE t FROM anita,zhi**

To revoke the SELECT privilege on table s.v from all users, use the following syntax:

**REVOKE SELECT ON TABLE s.v FROM PUBLIC**

To revoke the UPDATE privilege on columns c1 and c2 of table s.v from all users, use the following syntax:

**REVOKE UPDATE (c1,c2) ON TABLE s.v FROM PUBLIC**

To revoke the EXECUTE privilege on procedure p from the authorization ID george, use the following syntax:

**REVOKE EXECUTE ON PROCEDURE p FROM george RESTRICT**

To revoke the role purchases\_reader\_role from the authorization IDs george and maria, use the following syntax:

**REVOKE purchases\_reader\_role FROM george,maria**

To revoke the SELECT privilege on table t from the role purchases\_reader\_role, use the following syntax:

**REVOKE SELECT ON TABLE t FROM purchases\_reader\_role**

To revoke the USAGE privilege on the sequence generator order\_id from the role sales\_role, use the following syntax:

**REVOKE USAGE ON SEQUENCE order\_id FROM sales\_role;**

To revoke the USAGE privilege on the user-defined type price from the role finance\_role, use the following syntax:

**REVOKE USAGE ON TYPE price FROM finance\_role;**