Module 4: Advanced SQL

Fall 2026

Cheriton School of Computer Science

CS 338: Intro to Database Management

Outline

General Integrity Constraints in SQL

Syntax:

```
CREATE ASSERTION <assertion-name>
CHECK (<condition>)
```

- Delegates task of ensuring `<condition>` is always true to RDBMS.
  $\Rightarrow$ a transaction *fails* if it would make `<condition>` *false*
- `<condition>` can contain subqueries in the SQL standard.
  $\Rightarrow$ are at least as expressive as conditions in RC
  $\Rightarrow$ subqueries not allowed with most SQL engines (including DB2)

Example: Uniqueness Constraints

No pair of publications have the same pubid.

```
create assertion unique-pubid
check (
   not exists (
      select * from publication p1, publication p2
      where p1.pubid = p2.pubid
      and p1.title != p2.title ) )
```

In RC using universal quantification and shared variables:

$$\forall p, t_1, t_2.(\texttt{PUBLICATION}(p, t_1) \wedge \texttt{PUBLICATION}(p, t_2) \rightarrow t_1 = t_2).$$

Called an *equality generating dependency* in the literature.

Example: Foreign Keys

Every `author` *value in a* `WROTE` *tuple occurs as an* `aid` *value of some tuple in the* `AUTHOR` *table.*

```
create assertion author-foreign-key
check (
   not exists (
      select * from wrote w
      where not exists (
         select * from author a
         where a.aid = w.author ) ) )
```

In RC using universal quantification and shared variables:

$$\forall a.(\texttt{WROTE}(a, -) \rightarrow \texttt{AUTHOR}(a, -)).$$

Called a *tuple generating dependency* in the literature.

Views in SQL

Syntax:

```
CREATE VIEW <view-name> [AS] (<query>)
```

A view has many of the same properties as a table.

- Its definition appears in the database schema;
- Access controls can be applied to it;
- Other views can be defined in terms of it; and
- It can be queried as if it were a table.

Unlike tables, only *some* views are updatable.

Also unlike tables, an RDBMS adds updates to a transaction to ensure view consistency:

1. Updates to a view if underlying tables and views are updated; and
2. Updates to underlying tables and views if the view is updatable and itself updated.

Example: "CREATE VIEW"

A view called PUBSWITHAUTHORS *with ids, titles and author count of publications with at least one author.*

```
create view pubswithauthors as ( \
     select distinct pubid, title, count(*) as acnt \
     from publication, wrote where pubid = publication \
     group by publid, title )

DB20000I  The SQL command completed successfully.

select * from pubswithauthors

PUBID       TITLE                          ACNT
----------- ------------------------------ -----------
          1 Mathematical Logic                       1
          2 Trans. on Databases                      2
          4 Query Languages                          1

  3 record(s) selected.
```

Updating Views

- Modifications to a view's instance must be propagated to tables and other views referenced by the view query.
- Some views cannot be updated unambiguously:

Tables:

PERSON

NAME	CITIZENSHIP
Ed	Canadian
Dave	Canadian
Wes	American

NATIONAL-PASTIME

CITIZENSHIP	PASTIME
Canadian	Hockey
Canadian	Curling
American	Hockey
American	Baseball

View: $\{(n, p) \mid \exists c.\text{PERSON}(n, c) \wedge \text{NATIONAL-PASTIME}(c, p)\}$

PERSONAL-PASTIME

NAME	PASTIME
Ed	Hockey
Ed	Curling
Dave	Hockey
Dave	Curling
Wes	Hockey
Wes	Baseball

1. What does it mean to insert (Darryl, Hockey)?
2. What does it mean to delete (Dave, Curling)?

View Updates in SQL

According to SQL-92, a view is updatable only if its definition satisfies a variety of conditions:

- The query references exactly one table;
- The query only outputs simple attributes (no expressions);
- There is no grouping/aggregation/`DISTINCT`;
- There are no nested queries; and
- There are no set operations.

These rules are more restrictive than necessary.

The rules ensure that updates to underlying tables and views are implicitly defined by their old contents and by updates to the view itself.

Determining implicit definability in general is *undecidable*.

Outline

Multisets and Duplicates

- SQL has always had a more general *multiset* or *bag* semantics that *allows duplicates*, in contrast to the simpler *set* semantics of RC:

  $\Rightarrow$ SQL tables, views and query results are *multisets* of tuples

  $\Rightarrow$ originally adopted for efficiency reasons

- What does "allows duplicates" mean?

DEPT
cs
cs
pmath
stats
stats

$\Longleftrightarrow$

DEPT	CNT
cs	2
pmath	1
stats	2

Each $R/k \in \rho$ (table R with arity k) has an extra (hidden) column that represents a CNT of the number of occurrences of a tuple in any extension of R. Such *repetition counts* can be inferred from visualizations such as on the left.

How does multiset semantics impact efficiency?

EMP

NAME	DEPT
bob	cs
sue	cs
fred	pmath
barb	stats
jim	stats

$$\{(y) \mid \exists x.\mathrm{EMP}(x, y)\} \Longrightarrow$$

DEPT
cs
cs
pmath
stats
stats

$\Longleftrightarrow$

DEPT	CNT
cs	2
pmath	1
stats	2

EMP

NAME	DEPT
bob	cs
sue	cs
fred	pmath
barb	stats
jim	stats

$$\{(y) \mid \text{elim } \exists x.\mathrm{EMP}(x, y)\} \Longrightarrow$$

DEPT
cs
pmath
stats

$\Longleftrightarrow$

DEPT	CNT
cs	1
pmath	1
stats	1

A multiset semantics is assumed for existential quantification ($\exists$) in the RC query.

An additional kind of condition "elim φ" *removes duplicates*.

Range Restricted RC with Duplicate Elimination

Definition

Given a database signature $\rho = (R_1/k_1, \ldots, R_n/k_n)$, a set of variable names $\{x_1, x_2, \ldots\}$ and a set of constants $\{c_1, c_2, \ldots\}$, *range restricted conditions* are *formulas* defined by the grammar:

$$
\begin{array}{rcll}
\varphi & ::= & R_i(x_{i,1}, \ldots, x_{i,k_i}) & \\
& | & \varphi_1 \wedge (x_i = x_j) & \text{where } \{x_i, x_j\} \cap \mathrm{Fv}(\varphi_1) \neq \emptyset \\
& | & x_i = c_j & \\
& | & \varphi_1 \wedge \varphi_2 & \\
& | & \exists x_i . \varphi_1 & \\
& | & \varphi_1 \vee \varphi_2 & \text{where } \mathrm{Fv}(\varphi_1) = \mathrm{Fv}(\varphi_2) \\
& | & \varphi_1 \wedge \neg\varphi_2 & \text{where } \mathrm{Fv}(\varphi_1) = \mathrm{Fv}(\varphi_2) \\
& | & \text{elim } \varphi &
\end{array}
$$

A *range restricted RC query* has the form $\{(x_1, \ldots, x_n) \mid \varphi\}$, where $\{x_1, \ldots, x_n\} = \mathrm{Fv}(\varphi)$ and where φ is a range restricted condition.

Note: No *direct* access to the *repetition counts* is possible.

Multiset Semantics for Range Restricted RC

Semantics

A formula φ over a signature $\rho = (R_1/k_1, \ldots R_n/k_n)$ is true m *times* respect to *database instance* $\mathbf{DB} = (\mathbf{D}, \approx, \mathbf{R}_1, \ldots, \mathbf{R}_n)$ and *valuation* $\theta : \{x_1, x_2, \ldots\} \to \mathbf{D}$, written $\mathbf{DB}, \theta, m \models \varphi$, as given by the following:

$$
\begin{array}{lll}
\mathbf{DB}, \theta, 0 \models R_i(x_{i,1}, \ldots, x_{i,k_i}) & \text{if} & (\theta(x_{i,1}), \ldots, \theta(x_{i,k_i}), m) \notin \mathbf{R}, \text{ for any } m > 0 \\
\mathbf{DB}, \theta, m \models R_i(x_{i,1}, \ldots, x_{i,k_i}) & \text{if} & (\theta(x_{i,1}), \ldots, \theta(x_{i,k_i}), m) \in \mathbf{R} \\
\mathbf{DB}, \theta, m \models \varphi \wedge (x_i = x_j) & \text{if} & \mathbf{DB}, \theta, m \models \varphi \text{ and } \theta(x_i) \approx \theta(x_j) \\
\mathbf{DB}, \theta, 1 \models (x_i = c_j) & \text{if} & \theta(x_i) \approx c_j \\
\mathbf{DB}, \theta, m_1 \cdot m_2 \models \varphi \wedge \psi & \text{if} & \mathbf{DB}, \theta, m_1 \models \varphi \text{ and } \mathbf{DB}, \theta, m_2 \models \psi \\
\mathbf{DB}, \theta, \sum_{v \in D} m_v \models \exists x.\varphi & \text{if} & \mathbf{DB}, \theta[x := v], m_v \models \varphi \\
\mathbf{DB}, \theta, m_1 + m_2 \models \varphi \vee \psi & \text{if} & \mathbf{DB}, \theta, m_1 \models \varphi \text{ and } \mathbf{DB}, \theta, m_2 \models \psi \\
\mathbf{DB}, \theta, \max(0, m_1 - m_2) \models \varphi \wedge \neg\psi & \text{if} & \mathbf{DB}, \theta, m_1 \models \varphi \text{ and } \mathbf{DB}, \theta, m_2 \models \psi \\
\mathbf{DB}, \theta, 1 \models \text{elim } \varphi & \text{if} & \mathbf{DB}, \theta, m \models \varphi, \text{ where } m > 0
\end{array}
$$

The *answers* to a *query* $\{(x_1, \ldots, x_n) \mid \varphi\}$ over **DB** is given as follows, when $\{x_1, \ldots, x_n\} = \text{Fv}(\varphi)$:

$$\{(\theta(x_1), \ldots, \theta(x_n), m) \mid m > 0 \wedge \mathbf{DB}, \theta, m \models \varphi\}.$$

SQL: The Basic "SELECT Block" Revisited

Syntax:

```
3.  SELECT [DISTINCT]  <results>
1.  FROM               <tables>
2.  WHERE              <condition>
```

- Allows formulation of conjunctive ($\exists, \wedge$) range restricted RC queries with multiset semantics of the form:

$$\left\{ \texttt{<results>} \mid \text{[elim]}\ \exists \texttt{<unused>}.\ \left(\bigwedge \texttt{<tables>} \right) \wedge \texttt{<condition>} \right\}.$$

  $\Rightarrow$ a conjunction of `<tables>` with `<condition>`

  $\Rightarrow$ `<results>` specifies values in the resulting tuples

  $\Rightarrow$ `<unused>` are variables not used in `<results>`

  $\Rightarrow$ duplicates eliminated from result if DISTINCT included

Multiset Operations: Set Operations Revisited

Answers to `SELECT` blocks are *multisets of tuples*.
$\Rightarrow$ can apply *multiset operations* on them

- Multiset union: Q_1 `UNION ALL` Q_2.
  $\Rightarrow$ behaves like "$\vee$" in range restricted RC with multiset semantics
- Multiset difference: Q_1 `EXCEPT ALL` Q_2.
  $\Rightarrow$ behaves like "$\wedge\neg$" in range restricted RC with multiset semantics
- Multiset intersection: Q_1 `INTERSECT ALL` Q_2.
  $\Rightarrow$ maximum number of tuples common to Q_1 and Q_2
  $\Rightarrow$ rarely used

Q_1 and Q_2 must again have *union-compatible* signatures (same number and types of attributes).

Example: Removing DISTINCT

For every ordered pair of authorships on the same publication, list the id of the publication.

```
select r1.publication \
from wrote r1, wrote r2 \
where r1.publication = r2.publication \
and r1.author != r2.author

PUBLICATION
-----------
          2
          2

  2 record(s) selected.
```

Note: Publications with n authors would produce $O(n^2)$ answers.

Example: UNION ALL

For every book and article, list all authors.

```
( select author \
  from wrote, book \
  where publication = pubid ) \
union all \
( select author \
  from wrote, article \
  where publication = pubid )

AUTHOR
-----------
          1
          1

  2 record(s) selected.
```

“Pure” SQL Equivalence, Revisited

Recall how nesting in the WHERE clause is syntactic sugar:

```
select r.b                 select r.b
from r                     from r, (
where  r.a in (              select distinct b
  select b                   from s
  from s                   ) as s
)                          where  r.a = s.b
```

Rewriting does not generally hold if `DISTINCT` is removed.

SQL Query Summary: with Multisets

- Simple SELECT blocks;
- Set operations;
- Formulation of complex queries and nesting of queries;
- Aggregation;
- Duplicates and multiset operations.

On subqueries with duplicates:

1. Such subqueries occurring in `WHERE` clauses will not change the results computed by the top-level query.
2. Such subqueries occurring in `WITH` or `FROM` clauses *can* change the results computed by the top-level query.

Outline

What is a “null” value?

Phone

Name	Office	Home
Joe	1234	3456
Sue	1235	?

- Value inapplicable: *Sue doesn’t have home phone.*
- Value unknown: *Sue has home phone, but we don’t know her number.*

Value Inapplicable

- Essentially *poor schema design*.
- Better design:

OfficePhone

Name	Office
Joe	1234
Sue	1235

HomePhone

Name	Home
Joe	3456

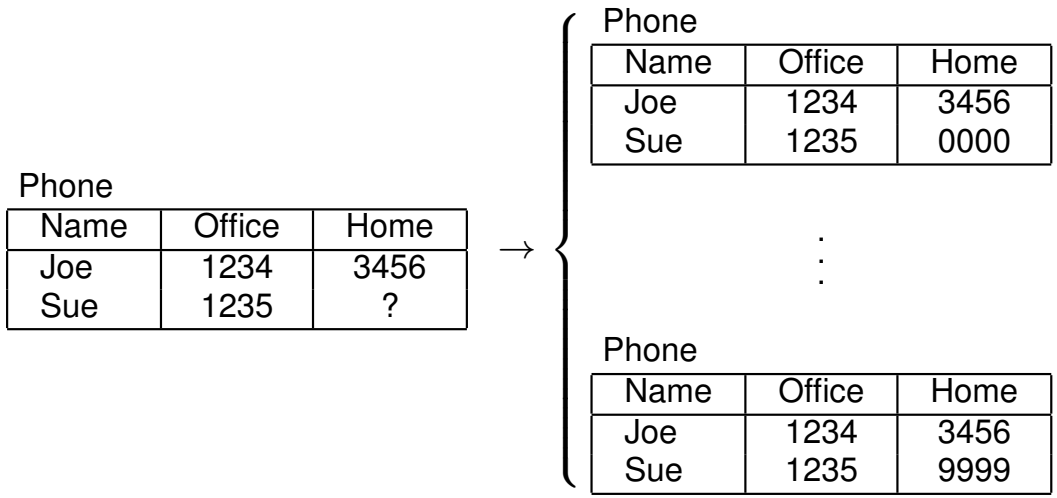
- Queries should behave *as if asked* over the above decomposition.
  $\Rightarrow$ (relatively) easy to implement

Value Unknown

Idea

Unknown values can be replaced by any domain value (that satisfies integrity constraints).

$\Rightarrow$ many possibilities (possible worlds)



Value Unknown and Queries

How do we answer queries?

Idea

Answers true in *all possible worlds* **W** of an *incomplete* instance **D**.

Certain Answer

$$Q(\mathbf{D}) = \bigcap_{\mathbf{W} \text{ world of } \mathbf{D}} Q(\mathbf{W})$$

Answer common to all possible worlds.

Is this (computationally) feasible?

No; NP-hard to *undecidable* except in trivial cases.

SQL's solution?

An approximation.

What can we do with `NULL`s in SQL?

expressions

- General rule: a `NULL` as a parameter to an operation makes the result `NULL`.
- $1 + \texttt{NULL} \rightarrow \texttt{NULL}$, $\texttt{'foo'} \,||\, \texttt{NULL} \rightarrow \texttt{NULL}$, etc.

predicates/comparisons

- Three-valued logic.

  $\Rightarrow$ approximation of *value unknown*

set operations

- Unique *special value* for *duplicates*.

aggregate operations

- Do not "*count*" (i.e., "value inapplicable").

Comparisons Revisited

Idea

Comparisons with a `NULL` value return `UNKNOWN`

$1 = 1$	`TRUE`
$1 =$ `NULL`	`UNKNOWN`
$1 = 2$	`FALSE`

Still short of *proper logical* behaviour.

Example:

$$(x = 0) \vee \neg(x = 0)$$

should always be `true` regardless of the value of x (including `NULL`), but . . .

UNKNOWN and Boolean Connectives

Idea

Boolean operations need to defined for UNKNOWN.

$\Rightarrow$ *extended truth tables* for Boolean connectives

$\wedge$	T	U	F
T	T	U	F
U	U	U	F
F	F	F	F

$\vee$	T	U	F
T	T	T	T
U	T	U	U
F	T	U	F

$\neg$	
T	F
U	U
F	T

. . . for tuples in which x is assigned the NULL value we get:

$$(x = 0) \vee \neg(x = 0) \rightarrow \texttt{UNKNOWN} \vee \neg\texttt{UNKNOWN} \rightarrow \texttt{UNKNOWN},$$

which is not the same as TRUE.

UNKNOWN in WHERE Clauses

How is this used in a WHERE clause?

- Additional syntax IS TRUE, IS FALSE, and IS UNKNOWN.
  ⇒ WHERE <cond> is shorthand for WHERE <cond> IS TRUE
- Special comparison IS NULL

Add a new author URI attribute and tuple.

```
alter table author add column URI varchar (20)

DB20000I  The SQL command completed successfully.

insert into AUTHOR values (3, 'Mary', 'uwaterloo.ca')

DB20000I  The SQL command completed successfully.
```

UNKNOWN in WHERE Clauses (cont'd)

List all author information.

```
select * from author

AID         NAME       URI
----------- ---------- --------------------
          1 Sue        -
          2 John       -
          3 Mary       uwaterloo.ca

  3 record(s) selected.
```

DB2 uses – to denote a NULL values.

UNKNOWN in WHERE Clauses (cont'd)

List all author ids and names for which we don't know a URL of their home page.

```
select aid, name from author \
where uri IS NULL

AID         NAME
----------- ----------
          1 Sue
          2 John

  2 record(s) selected.
```

Counting NULLs

How do NULLs interact with counting (and aggregates in general)?

- `COUNT(URL)` counts only non-NULL URL's.

  $\Rightarrow$ `COUNT(*)` counts "rows"

```
select count(*) as Rcnt, count(url) as Vcnt \
from author

RCNT        VCNT
----------- -----------
          3           1

  1 record(s) selected.
```

Outer Join

Idea

Allow "NULL-padded" answers that "fail to satisfy" a conjunct in a conjunction.

- Extension of syntax for the `FROM` clause:

  `FROM` T_1 `<j-type> JOIN` T_2 `ON` <cond>

  $\Rightarrow$ `<j-type>` is one of `FULL`, `LEFT`, `RIGHT`, or `INNER`

- Semantics, for $T_1 = R(x, y)$, $T_2 = S(y, z)$, and `<cond>` $= (r.y = s.y)$:

  1. $\{(x, y, z) \mid (R(x, y) \wedge S(y, z))$
  2. $\quad \vee \ ((z = \text{NULL}) \wedge R(x, y) \wedge \neg(\exists z.S(y, z)))$ (for `LEFT` and `FULL`)
  3. $\quad \vee \ ((x = \text{NULL}) \wedge S(y, z) \wedge \neg(\exists x.R(x, y)))\}$ (for `RIGHT` and `FULL`)

Example

```
select aid, publication \
from author left join wrote on aid=author

AID         PUBLICATION
----------- -----------
          1           1
          1           2
          1           4
          2           2
          3           -

  5 record(s) selected.
```

Counting with Outer Join

Author ids and a count of the number of publications for each.

```
select aid, count(publication) as pcnt \
from author left join wrote on aid = author \
group by aid
```

```
AID         PCNT
----------- -----------
          1           3
          2           1
          3           0

  3 record(s) selected.
```

SQL Query Summary: with NULLs

- `NULL`s are a necessary evil.
  $\Rightarrow$ used to account for irregularities in data
  $\Rightarrow$ should be used sparingly
- `NULL`s can *always* be avoided.
  $\Rightarrow$ however some of the solutions may be inefficient
- It is not possible to avoid `NULL`s in practice.
  $\Rightarrow$ *easy fix* for mistakes in schema design
  $\Rightarrow$ also surface because of schema evolution

Outline

Ordering Results in SQL

- No particular ordering on the rows of a table can be assumed when queries are written. (This is important!)
- No particular ordering of rows of an intermediate result in the query can be assumed either.
- However, it is possible to order the final result of a query with an `ORDER BY` clause at the end of the query.

Syntax:

$$\texttt{ORDER BY}\ e_1\ [Dir_1], \ldots, e_k\ [Dir_k]$$

where Dir_i is either `ASC` or `DESC`.

$\Rightarrow$ `ASC` is the default

Example

List all authors in the database in ascending order of their name.

```
select distinct * from author \
order by name

AID          NAME       URI
-----------  ---------- --------------------
           2 John       -
           3 Mary       uwaterloo.ca
           1 Sue        -

  3 record(s) selected.
```

Note: The `asc` keyword is optional, and is assumed by default.

$\Rightarrow$ a *descending* order would be obtained with the `desc` keyword

Minor sorts, minor minor sorts, etc., can be added.

Limiting the Number of Results

The number of results of a query can be *limited* by appending a `LIMIT` clause to a query.

Syntax, where e_i is a numeric expression:

<query> `LIMIT` e_1 [`OFFSET` e_2]

Semantics is non-deterministic:

- As many of the first e_1 answers to a query that exist for *any total order* to which the results of <query> can be extended, if there is no `OFFSET` given.
- As many of the next e_1 answers to a query that exist following the first e_2 answers for *any total order* to which the results of <query> can be extended, if `OFFSET` is given.

Semantics becomes deterministic *only when* <query> has an `ORDER BY` clause inducing a *total order*.

Example

List at most the first two entries of a sorted list of authors by name.

```
select distinct * from author \
order by name \
limit 2

AID         NAME       URI
----------- ---------- --------------------
          2 John       -
          3 Mary       uwaterloo.ca

  2 record(s) selected.
```

Example (cont'd)

List at most the next two entries following the second entry of a sorted list of authors by name.

```
select distinct * from author \
order by name \
limit 2 offset 2

AID         NAME       URI
----------- ---------- --------------------
          1 Sue        -

  1 record(s) selected.
```

Outline

Triggers in SQL

Syntax (for individual tuple events, and simplified):

```
CREATE TRIGGER <trigger-name>
AFTER <event> ON <table-or-view>
      [ REFERENCING OLD AS <corelation-old> ]
      [ REFERENCING NEW AS <corelation-new> ]
      FOR EACH ROW [ WHEN <condition> ] BEGIN ATOMIC
           <DML-action-1>;
           ...
           <DML-action-n>;
END
```

where `<event>` can be one of:

- `INSERT`,
- `DELETE`, or
- `UPDATE OF <attribute>`

Triggers in SQL: Example

When deleting an author, also delete related wrote tuples.

```
create trigger delete_author \
after delete on author \
referencing old as a \
for each row begin atomic \
   delete from wrote where wrote.author = a.aid; \
end
```

Triggers in SQL (cont')

Semantics (overview):

- Implements *event/condition/action* (ECA) rules.
- Adds additional operations to the ongoing transaction issuing the operation that *triggered* the execution of the rule.
- Timing of integrity constraint checking must be carefully managed.
  $\Rightarrow$ deferring all to `COMMIT` time always works, but impacts performance
- Makes the SQL DML Turing complete.

ECA Rules in Foreign Keys

Special syntax exists for common rules related to foreign key constraints:

```
FOREIGN KEY ( <from-attribute-list> )
REFERENCES <table> [ ( <to-attribute-list> > ) ]
     [ ON DELETE <action> ]
     [ ON UPDATE <action> ]
```

where `<action>` can be:

- `RESTRICT` (produce an error),
- `CASCADE` (propagate the delete), or
- `SET NULL` (set to “unknown”).

ECA Rules in Foreign Keys: Example

```
create table WROTE (
     author integer not null,
     publication integer not null,
     primary key (author, publication),
     foreign key (author) references AUTHOR
          on delete cascade,
     foreign key (publication) references PUBLICATION )
```

Authorization in SQL

The SQL DML includes a *data control language* (DCL) to manage *access rights* to database objects by users and groups of users.

Syntax (simplified):

```
GRANT  <role> TO <user>
GRANT  <what> ON <object> TO <user-or-role>
REVOKE <role> FROM <user>
REVOKE <what> ON <object> FROM <user-or-role>
```

where "`<what> ON <object>`" can be:

`DATABASE`

`CONNECT`

`TABLE` or `VIEW`

`ALTER`, `REFERENCES`, `SELECT`, `INSERT`, `DELETE`, or `UPDATE`

... and where `CREATE ROLE <role>` commands create new roles.

$\Rightarrow$ role `DBADM` is a *superuser*

Authorization in SQL: Examples

Create the `PAT` *role, and grant ability to access the* `PAYROLL` *database to* `PAT`*.*

```
create role PAT
grant connect on PAYROLL to PAT
```

Grant ability to query table `EMPLOYEE` *to the payroll project team.*

```
grant select on EMPLOYEE to PAT
```

Add Jim to the payroll project team.

```
grant PAT to Jim
```