SQL
- intergalactic dataspeak

History of SQL

- SQL: Structured Query Language
- SQL is based on the relational tuple calculus
- SEQUEL: Structured English QUEry Language; part of SYSTEM R, 1974
- SQL/86: ANSI & ISO standard
- SQL/89: ANSI & ISO standard
- SQL/92 or SQL2: ANSI & ISO standard
- SQL3: in the works...
- SQL2 supported by ORACLE, SYBASE, INFORMIX, IBM DB2, SQL SERVER, OPENINGRES,...

SQL

SQL consists of the following parts:

- Data Definition Language (DDL)
- Interactive Data Manipulation Language (Interactive DML)
- Embedded Data Manipulation Language (Embedded DML)
- Views
- Integrity
- Transaction Control
- Authorization
- Catalog and Dictionary Facilities

AIRPORT

airportcode	name	city	state

FLT-SCHEDULE

flt#	airline	dtime	from-airportcode	atime	to-airportcode	miles	price

FLT-WEEKDAY

flt#	weekday

FLT-INSTANCE

flt#	date	plane#	#avail-seats

AIRPLANE

plane#	plane-type	total-#seats

CUSTOMER

cust#	first	middle	last	phone#	street	city	state	zip

RESERVATION

flt#	date	cust#	seat#	check-in-status	ticket#

DDL - Overview

- primitive types
- domains
- schema
- tables

DDL - Primitive Types

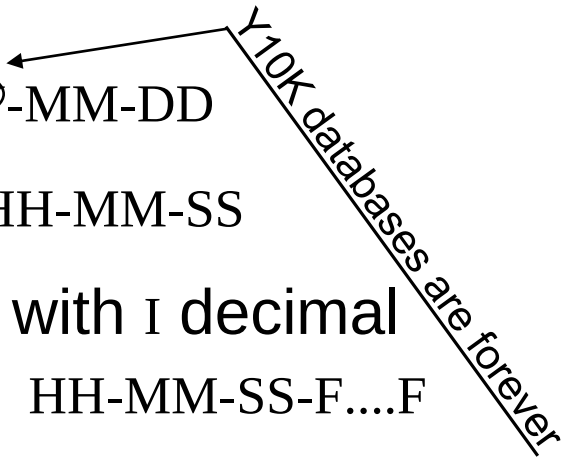
- numeric
  - INTEGER (or INT), SMALLINT are subsets of the integers (machine dependent)
  - REAL, DOUBLE PRECISION are floating-point and double-precision floating-point (machine dependent)
  - FLOAT(N) is floating-point with at least N digits
  - DECIMAL(P,D) (or DEC(P,D), or NUMERIC(P,D)), with P digits of which D are to the right of the decimal point

DDL - Primitive Types (cont.)

- character-string
  - CHAR(N) (or CHARACTER(N)) is a fixed-length character string
  - VARCHAR(N) (or CHAR VARYING(N), or CHARACTER VARYING(N)) is a variable-length character string with at most N characters
- bit-strings
  - BIT(N) is a fixed-length bit string
  - VARBIT(N) (or BIT VARYING(N)) is a bit string with at most N bits

DDL - Primitive Types (cont.)

- time
  - DATE is a date: YYYY-MM-DD
  - TIME, a time of day: HH-MM-SS
  - TIME(I), a time of day with I decimal fractions of a second: HH-MM-SS-F....F
  - TIME WITH TIME ZONE, a time with a time zone added: HH-MM-SS-HH-MM
  - TIME-STAMP, date, time, fractions of a second and an optional WITH TIME ZONE qualifier:
    YYYY-MM-DD-HH-MM-SS-F...F{-HH-MM}
  - INTERVAL, relative value used to increment or decrement DATE, TIME, or TIMESTAMP: YEAR/MONTH or DAY/TIME



DDL - Domains

- a domain can be defined as follows:

  CREATE DOMAIN AIRPORT-CODE CHAR(3);

  CREATE DOMAIN FLIGHTNUMBER CHAR(5);

- using domain definitions makes it easier to see which columns are related
- changing a domain definition one place changes it consistently everywhere it is used
- default values can be defined for domains
- constraints can be defined for domains (later)

DDL - Domains (cont.)

- all domains contain the value, NULL.
- to define a different default value:

CREATE DOMAIN AIRPORT-CODE CHAR(3)
DEFAULT '<literal>';

CREATE DOMAIN AIRPORT-CODE CHAR(3)
DEFAULT 'niladic function';

- literal, such as '???', 'NO-VALUE',...
- niladic function, such as USER, CURRENT-USER, SESSION-USER, SYSTEM-USER, CURRENT-DATE, CURRENT-TIME, CURRENT-TIMESTAMP
- defaults defined in a column takes precedence over the above

DDL - Domains (cont.)

- a domain is dropped as follows:

  DROP DOMAIN AIRPORT-CODE **RESTRICT;**

  DROP DOMAIN AIRPORT-CODE **CASCADE;**

- restrict: drop operation fails if the domain is used in column definitions
- cascade: drop operation causes columns to be defined directly on the underlying data type

DDL - Schema

- create a schema:

CREATE SCHEMA AIRLINE **AUTHORIZATION** LEO;

- the schema AIRLINE has now been created and is owner by the user "LEO"
- tables can now be created and added to the schema

- to drop a schema:

DROP SCHEMA AIRLINE **RESTRICT**;

DROP SCHEMA AIRLINE **CASCADE**;

- restrict: drop operation fails if schema is not empty
- cascade: drop operation removes everything in the schema

DDL - Tables

- to create a table in the AIRLINE schema:

```
CREATE TABLE AIRLINE.FLT-SCHEDULE
    (FLT#                    FLIGHTNUMBER NOT NULL,
    AIRLINE                  VARCHAR(25),
    FROM-AIRPORTCODE AIRPORT-CODE,
    DTIME                    TIME,
    TO-AIRPORTCODE     AIRPORT-CODE,
    ATIME                    TIME,
    PRIMARY KEY (FLT#),
    FOREIGN KEY (FROM-AIRPORTCODE)
    REFERENCES  AIRPORT(AIRPORTCODE),
    FOREIGN KEY (TO-AIRPORTCODE)
     REFERENCES AIRPORT(AIRPORTCODE));
```

DDL - Tables (cont.)

```
CREATE TABLE AIRLINE.FLT-WEEKDAY
    (FLT#              FLIGHTNUMBER    NOT NULL,
    WEEKDAY            CHAR(2),
    UNIQUE(FLT#, WEEKDAY),
    FOREIGN KEY (FLT#)
     REFERENCES  FLTT-SCHEDULE(FLT#));

CREATE TABLE AIRLINE.FLT-INSTANCE
    (FLT#              FLIGHTNUMBER    NOT NULL,
    DATE               DATE            NOT NULL,
    #AVAIL-SEATS   SMALLINT,
    PRIMARY KEY(FLT#, DATE),
    FOREIGN KEY FLT#
     REFERENCES FLT-SCHEDULE(FLT#));
```

DDL - Tables (cont.)

```
CREATE TABLE AIRLINE.RESERVATION
    (FLT#              FLIGHTNUMBER NOT NULL,
    DATE               DATE         NOT NULL,
    CUST#              INTEGER      NOT NULL,
     SEAT#              CHAR(4),
    CHECK-IN-STATUS    CHAR,
    UNIQUE(FLT#, DATE, CUST#),
    FOREIGN KEY (FLT#)
     REFERENCES FLT-INSTANCE(FLT#),
     FOREIGN KEY (DATE)
     REFERENCES FLT-INSTANCE(DATE),
    FOREIGN KEY (CUST#)
     REFERENCES CUSTOMER(CUST#));
```

DDL - Tables (cont.)

- to drop a table:

  DROP TABLE RESERVATION **RESTRICT;**

  DROP TABLE RESERVATION **CASCADE;**

- restrict: drop operation fails if the table is referenced by view/constraint definitions
- cascade: drop operation removes referencing view/constraint definitions

DDL - Tables (cont.)

- to add a column to a table:

  ALTER TABLE AIRLINE.FLT-SCHEDULE
  ADD PRICE DECIMAL(7,2);

- if no DEFAULT is specified, the new column will have NULL values for all tuples already in the database
- to drop a column from a table

  ALTER TABLE AIRLINE.FLT-SCHEDULE
  DROP PRICE **RESTRICT (**or **CASCADE)**;

- restrict: drop operation fails if the column is referenced
- cascade: drop operation removes referencing view/constraint definitions

Interactive DML - Overview

- select-from-where
- select clause
- where clause
- from clause
- tuple variables
- string matching
- ordering of rows
- set operations
- built-in functions
- nested subqueries
- joins
- recursive queries
- insert, delete, update

Interactive DML - select-from-where

SELECT $A_1, A_2, \ldots A_n$

FROM $R_1, R_2, \ldots R_m$

WHERE P

- the **SELECT** clause specifies the columns of the result
- the **FROM** clause specifies the tables to be scanned in the query
- the **WHERE** clause specifies the condition on the columns of the tables in the **FROM** clause
- equivalent algebra statement:

$$\pi_{A_1, A_2, \ldots A_n}(\sigma_P(R_1 x R_2 x \ldots R_m))$$

Interactive DML - select clause

- "Find the airlines in FLT-SCHEDULE"

```
SELECT AIRLINE
FROM FLT-SCHEDULE;
SELECT ALL AIRLINE
FROM FLT-SCHEDULE;
```

- "Find the airlines in FLT-SCHEDULE with duplicates removed"

```
SELECT DISTINCT AIRLINE
FROM FLT-SCHEDULE;
```

- "Find all columns in FLT-SCHEDULE"

```
SELECT *
FROM FLT-SCHEDULE;
```

- "Find FLT# and price raised by 10%"

```
SELECT FLT#, PRICE*1.1
FROM FLT-SCHEDULE;
```

Interactive DML - where clause

- "Find FLT# and price in FLT-SCHEDULE for flights out of Atlanta"

  SELECT FLT#, PRICE
  FROM FLT-SCHEDULE
  WHERE FROM-AIRPORTCODE="ATL";

- "Find FLT# and price in FLT-SCHEDULE for flights out of Atlanta with a price over $200"

  SELECT FLT#, PRICE
  FROM FLT-SCHEDULE
  WHERE FROM-AIRPORTCODE="ATL"
  AND PRICE > 200.00;

- connectives: AND, OR, NOT, ()
- comparisons: <, <=, >, >=, =, <>

Interactive DML - from clause

- "Find FLT#, WEEKDAY, and FROM-AIRPORTCODE in FLT-WEEKDAY and FLT-SCHEDULE"

```
SELECT FLT-SCHEDULE.FLT#,
                WEEKDAY, FROM-AIRPORTCODE
FROM FLT-WEEKDAY, FLT-SCHEDULE
WHERE FLT-WEEKDAY.FLT# =
FLT-SCHEDULE.FLT#;
```

- **dot-notation** disambiguates FLT# in FLT-WEEKDAY and FLT-SCHEDULE
- this is a **natural join**:

$$\pi_{\text{FLT-SCHEDULE.FLT\#, WEEKDAY, FROM-AIRPORTCODE}} (\text{FLT-SCHEDULE} \bowtie \text{FLT-WEEKDAY})$$

Interactive DML - tuple variables

- **alias** definition:

```
SELECT S.FLT#, WEEKDAY, T.FROM-AIRPORTCODE
FROM FLT-WEEKDAY S, FLT-SCHEDULE T
WHERE S.FLT# =  T.FLT#;
```

- S and T are **tuple variables**
- SQL's heritage as a tuple calculus language shows
- tuple variables are useful when one relation is used "twice" in a query:

```
SELECT S.FLT#
FROM FLT-SCHEDULE S, FLT-SCHEDULE T
WHERE S.PRICE > T.PRICE
AND T.FLT# = "DL212";
```

Interactive DML - string matching

- **wildcard** searches use:

  %: matches any substring

  _: matches any character

```
SELECT S.FLT#, WEEKDAY
FROM FLT-WEEKDAY S, FLT-SCHEDULE T
WHERE S.FLT# =  T.FLT#
AND T.AIRLINE LIKE "%an%";
```

- "%an%" matches American, Airtran, Scandinavian, Lufthansa, PanAm…
- "A%" matches American, Airtran, …
- "_ _ _ %" matches any string with at least three characters

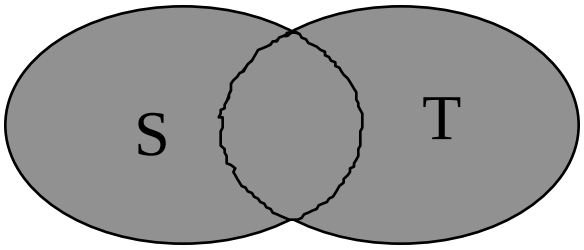
Interactive DML - ordering of rows

- the **order by** clause orders the rows in a query result in ascending (**asc**) or descending (**desc**) order
- "Find FLT#, airline, and price from FLT-SCHEDULE for flights out of Atlanta ordered by ascending airline and descending price:"

```
SELECT FLT#, AIRLINE, PRICE
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE="ATL"
ORDER BY AIRLINE ASC, PRICE DESC;
```

Interactive DML - set operations

$S \cup T$



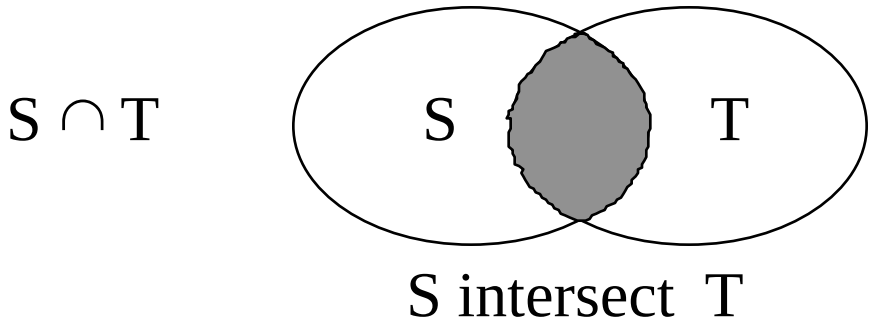
S union T

- "Find FLT# for flights on Tuesdays in FLT-WEEKDAY and FLT# with more than 100 seats in FLT-INSTANCE "

```
SELECT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY = "TU"
UNION
SELECT FLT#
FROM FLT-INSTANCE
WHERE #AVAIL-SEATS > 100;
```

- **UNION ALL** preserves duplicates

Interactive DML - set operation

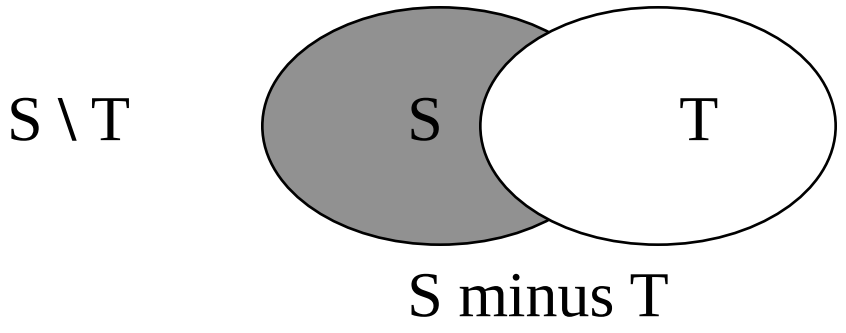


- "Find FLT# for flights on Tuesdays in FLT-WEEKDAY with more than 100 seats in FLT-INSTANCE"

```
SELECT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY = "TU"
INTERSECT
SELECT FLT#
FROM FLT-INSTANCE
WHERE #AVAIL-SEATS > 100;
```

- **INTERSECT ALL** preserves duplicates

Interactive DML - set operation



- "Find FLT# for flights on Tuesdays in FLT-WEEKDAY except FLT# with more than 100 seats in FLT-INSTANCE"

```
SELECT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY = "TU"
EXCEPT
SELECT FLT#
FROM FLT-INSTANCE
WHERE #AVAIL-SEATS > 100;
```

- **EXCEPT ALL** preserves duplicates

Interactive DML - built-in functions

- count (COUNT), sum (SUM), average (AVG), minimum (MIN), maximum (MAX)
- "Count flights scheduled for Tuesdays from FLT-WEEKDAY"

```
SELECT COUNT( *)
FROM FLT-WEEKDAY
WHERE WEEKDAY = "TU";
```

- "Find the average ticket price by airline from FLT-SCHEDULE"

```
SELECT AIRLINE, AVG(PRICE)
FROM FLT-SCHEDULE
GROUP BY AIRLINE;
```

Interactive DML - built-in functions

- "Find the average ticket price by airline for scheduled flights out of Atlanta for airlines with more than 5 scheduled flights out of Atlanta from FLT-SCHEDULE"

```
SELECT AIRLINE, AVG(PRICE)
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE = "ATL"
GROUP BY AIRLINE
HAVING COUNT (FLT#) >= 5;
```

- "Find the highest priced flight(s) out of Atlanta from FLT-SCHEDULE"

```
SELECT FLT#, MAX(PRICE)
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE = "ATL";
```

Interactive DML - nested subqueries

- Set membership: **IN, NOT IN**
- "Find airlines from FLT-SCHEDULE where FLT# is in the set of FLT#'s for flights on Tuesdays from FLT-WEEKDAY"

```
SELECT DISTINCT AIRLINE
FROM FLT-SCHEDULE
WHERE FLT# IN
        (SELECT FLT#
         FROM FLT-WEEKDAY
         WHERE WEEKDAY = "TU");
```

- "Find FLT#'s for flights on Tuesdays or Thursdays from FLT-WEEKDAY"

```
SELECT DISTINCT FLT#
FROM FLT-WEEKDAY
WHERE WEEKDAY IN ("TU", "TH");
```

Interactive DML - nested subqueries

- set comparison

  =, <, <=, >, >=, <> ALL

  =, <, <=, >, >=, <> SOME

- "Find FLT# for flights from Atlanta to Chicago with a price that is lower than all flights from Birmingham to Chicago"

```
SELECT  FLT#
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE="ATL"
AND TO-AIRPORTCODE="CHI" AND PRICE <
ALL   (SELECT PRICE
        FROM FLT-SCHEDULE
        WHERE FROM-AIRPORTCODE="BIR"
        AND TO-AIRPORTCODE="CHI");
```

Interactive DML - nested subqueries

- test empty relation: **EXISTS, NOT EXISTS**
- "Find FLT# for flights from Atlanta to Chicago with a price so low that there does not exist any cheaper flights from Birmingham to Chicago"

```
SELECT  S.FLT#
FROM FLT-SCHEDULE S
WHERE S.FROM-AIRPORTCODE="ATL"
AND S.TO-AIRPORTCODE="CHI" AND NOT
EXISTS
        (SELECT T.FLT#
         FROM FLT-SCHEDULE T
         WHERE T.FROM-AIRPORTCODE="BIR"
         AND T.TO-AIRPORTCODE="CHI"
          AND T.PRICE < S.PRICE);
```

- notice: **reference out of inner scope**

Interactive DML - joins

- **cross join:** Cartesian product
- [**inner**] **join:** only keeps rows that satisfy the join condition
- **left outer join:** keeps all rows from left table; fills in nulls as needed
- **right outer join:** keeps all rows from right table; fills in nulls as needed
- **full outer join:** keeps all rows from both tables; fills in nulls as needed
- **natural** or **on-condition** must be specified for all inner and outer joins
- **natural:** equi-join on columns with same name; one column preserved

Interactive DML - joins

- "Find all two-leg, one-day trips out of Atlanta; show also a leg-one even if there is no connecting leg-two the same day"

```
SELECT X.FLT# LEG-ONE, Y.FLT# LEG-TWO
FROM
((FLT-SCHEDULE NATURAL JOIN FLT-INSTANCE)
    X
 LEFT OUTER JOIN
 (FLT-SCHEDULE NATURAL JOIN FLT-INSTANCE) Y
 ON  (X.TO-AIRPORTCODE=Y.FROM-AIRPORTCODE
 AND X.DATE=Y.DATE AND X.ATIME<Y.DTIME))
WHERE X.FROM-AIRPORTCODE="ATL";
```

Interactive DML - recursive queries

- not in SQL2; maybe in SQL3...(?)

- "Find all reachable airports for multi-leg trips out of Atlanta"

```
WITH
    PAIRS AS SELECT FROM-AIRPORTCODE D, TO-
    AIRPORTCODE A FROM FLT-SCHEDULE,

    RECURSIVE REACHES(D, A) AS    /*initially empty*/
            PAIRS
            UNION
           (SELECT PAIRS.D, REACHES.A
            FROM PAIRS, REACHES
            WHERE PAIRS.A=REACHES.D)

SELECT A FROM REACHES WHERE D="ATL";
```

Interactive DML - **insert,** delete, update

INSERT INTO FLT-SCHEDULE
VALUES (“DL212”, “DELTA”, 11-15-00, “ATL”,
13-05-00, ”CHI”, 650, 00351.00);

INSERT INTO FLT-SCHEDULE(FLT#,AIRLINE)
VALUES (“DL212”, “DELTA”); /*default nulls added*/

“Insert into FLT-INSTANCE all flights scheduled for Thursday, 9/10/98”

INSERT INTO FLT-INSTANCE(FLT#, DATE)
(**SELECT** S.FLT#, 1998-09-10
FROM FLT-SCHEDULE S, FLT-WEEKDAY D
WHERE S.FLT#=D.FLT#
AND D.WEEKDAY=“TH”);

Interactive DML
- insert, **delete,** update

"Cancel all flight instances for Delta on 9/10/98"

```
DELETE FROM FLT-INSTANCE
WHERE DATE=1998-09-10
AND FLT# IN
    (SELECT FLT#
    FROM FLT-SCHEDULE
    WHERE AIRLINE="DELTA");
```

Interactive DML
- insert, delete, **update**

"Update all reservations for customers on DL212 on 9/10/98 to reservations on AA121 on 9/10/98"

UPDATE RESERVATION

SET FLT#="AA121"

WHERE DATE=1998-09-10

AND FLT#="DL212";

Embedded DML - Overview

- host languages
- precompilation
- impedance mismatch
- database access
- cursor types
- fetch orientation
- exception handling

Embedded DML - host languages

- SQL doesn't do iteration, recursion, report printing, user interaction, and SQL doesn't do Windows
- SQL may be embedded in host languages, like COBOL, FORTRAN, MUMPS, PL/I, PASCAL, ADA, C, C++, JAVA
- the exact syntax of **embedded** SQL depends on the host language

Embedded DML - precompilation

- the precompiler replaces embedded SQL with host language declarations and function calls to the SQL library that allow run-time execution of the database access
- to allow the precompiler to identify embedded SQL, the following construct is used:

EXEC SQL

< embedded SQL statement >**;**

Embedded DML - impedance mismatch

- SQL is a powerful, set-oriented, declarative language
- SQL queries return sets of rows
- host languages cannot handle large sets of structured data
- cursors resolve the mismatch:

```
EXEC SQL BEGIN DECLARE SECTION;
DECLARE FROM-AIRPORTCODE CHAR(3);
                                    /*input to query*/
.....
.....
EXEC SQL END DECLARE SECTION;
EXEC SQL
    DECLARE FLT CURSOR
    FOR SELECT FLT#, AIRLINE, PRICE
        FROM FLT-SCHEDULE
        WHERE FROM-AIRPORTCODE =
                        :FROM-AIRPORTCODE;
```

Embedded DML - database access

- to access the result of the query, one row at a time, the following is used:

```
EXEC SQL BEGIN DECLARE SECTION;
 ..........
 ..........
DECLARE FLT# CHAR(5);     /*targets for FETCH*/
DECLARE AIRLINE VARCHAR(25);
DECLARE PRICE DECIMAL(7,2);
EXEC SQL END DECLARE SECTION;
EXEC SQL OPEN FLT;
                                  /*:FROM-AIRPORTCODE value */
                                   /*query executed; cursor open;*/
WHILE MORE FLTs DO        /*first row is the current row */
   EXEC SQL FETCH FLT  /*one row of query placed in   */
   INTO :FLT#, :AIRLINE, :PRICE;          /*host variables*/
   DO YOUR THING WITH THE DATA;
END-WHILE;
EXEC SQL CLOSE FLT;                        /*cursor closed*/
```

Embedded SQL - cursor types

syntax for cursor definition:

DECLARE <NAME> [**INSENSITIVE**] [**SCROLL**] **CURSOR**
FOR <QUERY>
[**FOR** {**READ ONLY** | **UPDATE** }]

Embedded SQL - cursor types

	insensitive	scroll	insensitiv scroll	
read only	•fetch next only •separate copy •no update,delete •update,delete by others not visible	•several fetch orientations •no update,delete •update,delete by others may be visible	•several fetch orientations • separate copy •no update,delete •update,delete by others not visible	•fetch next only •no update,delete •update,delete by others may be visible
update		•several fetch orientations •update,delete ok •update,delete by others may be visible		•fetch next only •update,delete ok •update,delete by others may be visible
	•fetch next only •separate copy •no update,delete •update,delete by others not visible	•several fetch orientations •update,delete ok •update,delete by others may be visible	•several fetch orientations • separate copy •no update,delete •update,delete by others not visible	•fetch next only •update,delete ok •update,delete by others may be visible

Embedded DML - fetch orientation

- NEXT
- PRIOR
- FIRST
- LAST
- ABSOLUTE i
- RELATIVE i

"i": literal, parameter, or host variable

Embedded DML - exception handling

```
EXEC SQL BEGIN DECLARE SECTION;
 ..........
 ..........
DECLARE SQLCODE  INT;
EXEC SQL END DECLARE SECTION;
```

- a value is returned to SQLCODE each time an SQL library function is called
- the host language program uses SQLCODE in exception handling

SQLCODE	MEANING
= 0	successful
> 0	warning
< 0	error
.....	.....

Embedded DML - exception handling

- **explicit**: SQLCODE is checked after each SQL library function call; warnings and errors are processed; program is cluttered with exception handling code
- **implicit**: WHENEVER-statement makes precompiler insert exception handling code after each SQL library function call

Embedded DML - exception handling

EXEC SQL WHENEVER <condition> <action>;

- <condition>:
  - SQLWARNING
  - SQLERROR
- <action>
  - CONTINUE: ignore and continue
  - DO FUNCTION: call a function
  - DO BREAK: break out of control structure
  - DO RETURN: perform return statement
  - GOTO LABEL: branch to label
  - STOP: stop program and roll back

Embedded DML - exception handling

```
EXEC SQL BEGIN DECLARE SECTION;
 ..........
 ..........
DECLARE SQLCODE  INT;
EXEC SQL END DECLARE SECTION;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER SQLERROR GOTO QUIT;
..........
..........
EXEC SQL OPEN FLT;
WHILE TRUE DO
   EXEC SQL FETCH FLT
   INTO :FLT#, :AIRLINE, :PRICE;
   DO YOUR THING WITH THE DATA;
END-WHILE;
EXEC SQL CLOSE FLT;
QUIT: IF SQLCODE < 0 THEN EXEC SQL ROLLBACK
         ELSE EXEC SQL COMMIT;
```

Views
- definition, use, update

- a view is a **virtual** table
- how a view is defined:

```
CREATE VIEW ATL-FLT
AS SELECT FLT#, AIRLINE, PRICE
     FROM FLT-SCHEDULE
     WHERE FROM-AIRPORTCODE = "ATL";
```

- how a query on a view is written:

```
SELECT *
FROM ATL-FLT
WHERE PRICE <= 00200.00;
```

- how a query on a view is computed:

```
SELECT FLT#, AIRLINE, PRICE
FROM FLT-SCHEDULE
WHERE FROM-AIRPORTCODE="ATL"
AND PRICE<00200.00;
```

- how a view definition is dropped:

```
DROP VIEW ATL-FLT [RESTRICT|CASCADE];
```

Views
- definition, use, update

- views inherit column names of the base tables they are defined from
- columns may be explicitly named in the view definition
- column names must be named if inheriting them causes ambiguity
- views may have computed columns, e.g. from applying built-in-functions; these must be named in the view definition

Views
- definition, use, update

these views are **not** updatable

```
CREATE VIEW ATL-PRICES
    AS SELECT AIRLINE, PRICE
        FROM FLT-SCHEDULE
        WHERE FROM-AIRPORTCODE="ATL";

CREATE VIEW AVG-ATL-PRICES
    AS SELECT AIRLINE, AVG(PRICE)
        FROM FLT-SCHEDULE
        WHERE FROM-AIRPORTCODE="ATL"
        GROUP BY AIRLINE;
```

not unique

computed attribute

this view is theoretically updatable, but cannot be updated in SQL

```
CREATE VIEW FLT-SCHED-AND-DAY
AS SELECT S.*,  D.WEEKDAY
 FROM FLT-SCHEDULE S, FLT-WEEKDAY D
 WHERE D.FLT# = S.FLT#;
```

join of two tables

Views
- definition, use, update

a view is **updatable if and only if**:

- it does not contain any of the keywords JOIN, UNION, INTERSECT, EXCEPT
- it does not contain the keyword DISTINCT
- every column in the view corresponds to a uniquely identifiable base table column
- the FROM clause references exactly one table which must be a base table or an updatable view
- the table referenced in the FROM clause cannot be referenced in the FROM clause of a nested WHERE clause
- it does not have a GROUP BY clause
- it does not have a HAVING clause

updatable means insert,delete, update all ok

Views
- definition, use, update

```
CREATE VIEW LOW-ATL-FARES  /*updatable view*/
    AS SELECT *
        FROM FLT-SCHEDULE
        WHERE FROM-AIRPORTCODE=“ATL”
        AND PRICE<00200.00;

UPDATE      LOW-ATL-FARES     /*moves row        */
SET   PRICE = 00250.00        /* outside the view*/
WHERE TO-AIRPORTCODE = “BOS”;

INSERT INTO LOW-ATL-FARES     /*creates row      */
VALUES (“DL222”, ”DELTA”,     /*outside the view*/
              ”BIR”, 11-15-00, ”CHI”, 13-05-00, 00180.00);

CREATE VIEW LOW-ATL-FARES
    AS SELECT *
      FROM FLT-SCHEDULE
      WHERE FROM-AIRPORTCODE=“ATL”
      AND PRICE<00200.00
WITH CHECK OPTION;            /*prevents updates*/
                              /*outside the view*/
```

Integrity
- constraints

- constraint: a conditional expression required not to evaluate to *false*
- a constraint cannot be created if it is already violated
- a constraint is enforced from the point of creation forward
- a constraint has a unique name
- if a constraint is violated its name is made available to the user
- constraints cannot reference parameters or host variables; they are application independent
- data type checking is a primitive form of constraint

Integrity
- domain constraints

- associated with a domain; applies to all columns defined on the domain

```
CREATE DOMAIN WEEKDAY CHAR(2)
CONSTRAINT IC-WEEKDAY
 CHECK (VALUE IN
        ( “MO”, “TU”, “WE”, “TH”, “FR”, “SA”, “SU”));

CREATE DOMAIN PRICE DECIMAL(7,2)
CONSTRAINT IC-PRICE
CHECK (VALUE > 00000.00 );

CREATE DOMAIN FLT# CHAR(5)
CONSTRAINT IC-FLT#
CHECK (VALUE NOT NULL);
```

Integrity
- base table, column constraints

- associated with a specific base table

```
CREATE TABLE AIRLINE.FLT-SCHEDULE
    (FLT#                   FLIGHTNUMBER NOT NULL,
    AIRLINE                 VARCHAR(25),
    FROM-AIRPORTCODE        AIRPORT-CODE,
    DTIME                   TIME,
    TO-AIRPORTCODE          AIRPORT-CODE,
    ATIME                   TIME,
    CONSTRAINT  FLTPK PRIMARY KEY (FLT#),
    CONSTRAINT FROM-AIRPORTCODE-FK
    FOREIGN KEY (FROM-AIRPORTCODE)
    REFERENCES AIRPORT(AIRPORTCODE)
        ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (FROM-AIRPORTCODE)
    REFERENCES AIRPORT(AIRPORTCODE)
        ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT IC-DTIME-ATIME
    CHECK DTIME < ATIME);
```

Integrity
- general constraints

- applies to an arbitrary combination of columns and tables
- connecting RESERVATIONS for a customer must make sense:

```
CREATE ASSERTION IC-CONNECTING-FLIGHTS
CHECK (NOT EXISTS
 (SELECT *
 FROM FLT-SCHEDULE FS1 FS2, RESERVATION R1 R2
 WHERE FS1.FLT#=R1.FLT#
 AND FS2.FLT#=R2.FLT#
 AND R1.DATE=R2.DATE
 AND FS1.TO-AIRPORTCODE=FS2.FROM-AIRPORTCODE
 AND FS1.ATIME+ INTERVAL “30” MINUTE
        > FS2.DTIME));
```

Integrity
- (not so) general constraints

- not all constraints can be specified

```
CREATE TABLE AIRLINE.FLT-WEEKDAY
    (FLT#          FLIGHTNUMBER          NOT NULL,
    WEEKDAY    CHAR(2),
    .... ));
CREATE TABLE AIRLINE.FLT-INSTANCE
    (FLT#              FLIGHTNUMBER    NOT NULL,
    DATE               DATE            NOT NULL,
    .... ));

CREATE ASSERTION DATE-WEEKDAY-CHECK
  (NOT EXISTS
     (SELECT *
     FROM FLT-INSTANCE FI, FLT-WEEKDAY FSD
     WHERE FI.FLT#=FSD.FLT#
     AND weekday-of(FI.DATE) <> FSD.WEEKDAY));
```

- weekday-of: DATE → WEEKDAY

Integrity
- deferred, immediate checking

- sometimes constraint checking must be deferred, e.g. when there is a referential cycle
- a constraint definition may optionally include either or both of:

  INITIALLY { **DEFERRED** | **IMMEDIATE** }

  [**NOT**] **DEFERRABLE**

- constraint checking is turned on/off by:

  SET CONSTRAINTS { <list> | **ALL }**

  { **DEFERRED** | **IMMEDIATE** }

- the constraints in <list> and ALL must all be deferrable

Integrity
- deferred, immediate checking

- possible combinations:

	INITIALLY DEFERRED	INITIALLY IMMEDIATE	(INITIALLY IMMEDIATE IMPLIED)
NOT DEFERRABLE	NOT ALLOWED		
DEFERRABLE			
	DEFERRABLE IMPLIED	NOT DEFERRABLE IMPLIED	NOT DEFERRABLE IMPLIED

Integrity
- deferred, immediate checking

- example:

CREATE TABLE AIRLINE.FLT-SCHEDULE
(FLT# FLIGHTNUMBER **NOT NULL,**
....))
CONSTRAINT FS-FK **FOREIGN KEY....**
INITIALLY DEFERRED;

CREATE TABLE AIRLINE.FLT-WEEKDAY
(FLT# FLIGHTNUMBER **NOT NULL,**
....))
CONSTRAINT FSD-FK **FOREIGN KEY....**
INITIALLY DEFERRED;

INSERT INTO FLT-WEEKDAY **VALUES** (...);
INSERT INTO FLT-SCHEDULE **VALUES** (...);
SET CONSTRAINT FS-FK, FSD-FK **IMMEDIATE;**
IF SQLCODE="SET CONSTRAINTS FAILED"
THEN **ROLLBACK**
ELSE **COMMIT**;

Transaction Control

- atomic, consistent, isolated, durable (ACID) transactions are supported by:
  - COMMIT and
  - ROLLBACK

- **EXEC SQL OPEN** FLT**;**
- WHILE TRUE DO
- **EXEC SQL FETCH** FLT
- **INTO** :FLT#, :AIRLINE, :PRICE;
- DO YOUR THING WITH THE DATA;
- END-WHILE;
- **EXEC SQL CLOSE** FLT**;**
- QUIT: IF SQLCODE < 0 THEN **EXEC SQL ROLLBACK**
- ELSE **EXEC SQL COMMIT;**

Authorization

- Discretionary Access Control (DAC) is supported by GRANT and REVOKE:

GRANT <privileges>
ON <table>
TO <users>
[**WITH GRANT OPTION**];

REVOKE [**GRANT OPTION FOR**] <privileges>
ON <table>
FROM <users> {**RESTRICT** | **CASCADE**};

<privileges>: SELECT, INSERT(X), INSERT,
UPDATE(X), UPDATE, DELETE

CASCADE: revoke cascades through its subtree

RESTRICT: revoke succeeds only if there is no subtree

Authorization

GRANT INSERT, DELETE
ON FLT-SCHEDULE
TO U1, U2
WITH GRANT OPTION;

GRANT UPDATE(PRICE)
ON FLT-SCHEDULE
TO U3;

REVOKE GRANT OPTION FOR DELETE
ON FLT-SCHEDULE
FROM U2 **CASCADE**;

REVOKE DELETE
ON FLT-SCHEDULE
FROM U2 **CASCADE**;

Catalog and Dictionary Facilities

an INFORMATION_SCHEMA contains the following tables (or rather views) for the CURRENT_USER:

- INFORMATION-_SCHEMA_CATALOG_NAME: single-row, single-column table with the name of the catalog in which the INFORMATION_SCHEMA resides
- SCHEMATA created by CURRENT_USER
- DOMAINS accessible to CURRENT_USER
- TABLES accessible to CURRENT_USER
- VIEWS accessible to CURRENT_USER
- COLUMNS of tables accessible to CURRENT_USER
- TABLE_PRIVILEGES granted by or to CURRENT_USER
- COLUMN_PRIVILEGES granted by or to CURRENT_USER
- USAGE_PRIVILEGES granted by or to CURRENT_USER
- DOMAIN_CONSTRAINTS
- TABLE_CONSTRAINTS
- REFERENTIAL_CONSTRAINTS
- CHECK_CONSTRAINTS
- **and 18 others ...**