

**11. ANY**

Query: “find all SNRs whose scodes are greater than some SNRs whose name is CB.”

SQL: **SELECT snr FROM s x WHERE scodes > ANY ( SELECT scode FROM s y WHERE y.sname='CB');**

snr
S2
S4
S5

**12. ALL**

Query: “Find all salespersons with the greatest scode.”

SQL: **SELECT snr FROM s x WHERE scode >= ALL (SELECT scode FROM s y);**

snr
S2
S5

**13. LIKE (% , \_)**

Query: “Find snr & sname for salesperson whose name starts with ‘C’ and ends with ”N” and where the first character in SNR is ‘S’.

SQL: **SELECT \* FROM s WHERE (sname LIKE 'C%N') AND (snr LIKE 'S\_');**

SNR	SCode	City	sname
S1	10	Athens	CHARLIE BROWN

**14.**

Query: “Find PNR & Price for products not in the price range from (50 → 80)

SQL:

- a. **SELECT pnr, price FROM p WHERE price NOT BETWEEN 50 AND 80;**
- b. **SELECT pnr, price FROM p WHERE NOT (price >=50 AND price <= 80);**
- c. **SELECT pnr, price FROM p WHERE (price < 50 OR price > 80);**

pnr	price
P3	90
P4	20

**15.**

Query: “Output pricelist in which the products are grouped according to the price of the product including a 22% RST.”

SQL: **SELECT pnr, price \* 1.22, pname FROM p ORDER BY 2, pnr;**

(Note: I prefer this: **SELECT pnr, price \* 1.22 AS prodprice, pname FROM p ORDER BY 2, pnr;**)

pnr	Price*1.22	Pname
P4	24.4	Socks
P1	61	Shirt
P5	61	Blouse
P8	73.2	Blouse
P3	109.8	Trouser

pnr	prodprice	pname
P4	24.4	Socks
P1	61	Shirt
P5	61	Blouse
P8	73.2	Blouse
P3	109.8	Trouser

### 16. Built in Functions

Query: Display number of salespersons who have sold product P3, and find the sum, max quality, min. quality and average quality.

SQL: **SELECT COUNT(\*), SUM(qty), MAX(qty), MIN(qty), AVG(qty) FROM sp WHERE pnr='P3';**

Count(*)	SUM(qty)	MAX(qty)	MIN(qty)	AVG(qty)
2	600	500	100	300

### 17. Group by & Having

Query: Display a list of cities with more than one salesperson

SQL: **SELECT city, COUNT(\*), AVG(scode), COUNT (DISTINCT scode) FROM s WHERE NOT city IS NULL GROUP BY city HAVING COUNT(\*) > 1 ORDER BY city;**

city	COUNT(*)	AVG(scode)	COUNT (DISTINCT scode)
Toronto	2	30	2

Note: COUNT(DISTINCT scode) does not work in Microsoft Access.

### 18.

Query: Display a list of individual salespersons' total sales

SQL: **SELECT sp, snr, SUM(qty\*price) FROM sp, p WHERE sp.pnr=p.pnr GROUP BY sp.snr;**

snr	sum(qty*price)
S2	14000
S4	16000
S5	94500

**TABLES USED TO GET SQL #11-#18 (ALSO, SOME FROM #1-#11)**

SP

SNR	PNR	QTY
S2	P1	100
S2	P3	100
S4	P5	200
S4	P8	100
S5	P1	110
S5	P3	500
S5	P4	500
S5	P5	500
S5	P8	150

S

SNR	SCode	City	sname
S1	10	Athens	CHARLIE BROWN
S2	30	Toronto	BUGS BUNNY
S4	20	Kingston	ROAD RUNNER
S5	30	Toronto	PO

PNR	pname	size	price
P1	Shirt	5	50
P3	Trouser	6	90
P4	Socks	6	20
P5	Blouse	8	50
P8	Blouse	5	60

P

## UPDATING TABLES WITH SQL

### *INSERT INTO*

#### 19.

Query: To create a new salesperson entry with SNR=S6, sname=RR, scode=0, city=wonderland

- a. **INSERT INTO s VALUES** ('S6', 'RR', 0, 'Wonderland');
- b. **INSERT INTO s** (snr, sname, scode, city) **VALUES** ('S6', 'RR', 0, 'Wonderland');

#### 20. A set of new entries can be created:

```
CREATE TABLE T3
(snr char (5));
```

```
INSERT INTO T3
SELECT SNR FROM s WHERE city='Toronto';
```

#### 21.

Query: Delete all salesperson records who live in Toronto.

```
SQL: DELETE FROM s
      WHERE city='Toronto';
```

?? to see the DDL a bit further.

#### 22. Update Query.

Query: Increase the scode of S2 by 5.

```
SQL: UPDATE s
      SET scode = scode + 5
      WHERE snr = 'S2';
```

#### 23.

Query: Same as above, but for all salesperson in Toronto and at the same time change city to Raptors.

```
SQL: UPDATE s
      SET scode = scode + 5,
          city = 'Raptors'
      WHERE city = 'Toronto';
```

---

## Further on delete in DDL

Eg. **Create table** SP ( SNR char(5) **NOT NULL**,  
PNR char(6) **NOT NULL**,  
QTY **integer**,  
**primary key** (SNR, PNR),  
**foreign key** SFK(SNR) **references** S  
**ON DELETE CASCADE**,  
**foreign key** PFK(PNR) **references** P  
**ON DELETE RESTRICT**)

**Note:** The “on delete” clause defines the delete rule for the target table with respect to this foreign key, ie. it defines what happens if an attempt is made to delete a row from the target table.

**Restrict:** the delete is restricted to the case where there are no matching rows in the second table (it is rejected if any such rows exists)

**Cascade:** Deletes all matching rows in 2<sup>nd</sup> table also.

**Note:** if the key in the 2<sup>nd</sup> table references to a third table then the 3<sup>rd</sup> table referenced rows will be deleted too. It's like a chain reaction. Therefore, a single delete can cascade thru a large number of tables.

## Integrity revisited:

1. **Primary Keys (NOT NULL)** *(LS Side Note: primary keys are unique and not null)*

primary key (col-name [, col-name]... )

2. Foreign keys: need to specify both the column that is a foreign key and the table it is to match.

3. Integrity in SQL:

Legal values: Ensures that only values that satisfies a particular condition are allowed in a given column. (Command in SQL is **CHECK**)

SQL: **CHECK** city **IN** ('Athens', 'Kingston', 'Toronto');

October 17, 2000

Today cover: Catalog, views, index, (triggers, assertions) → not in test.

## CATALOG

- Info concerning the database known to the system: tables, views, etc...
- here, we are going to only see 3 tables:
  - **systables**: all tables known to SQL
  - **syscolumns**: all the columns within those tables
  - **sysindexes**: about the indexes.

systables		
name	creator	colcount
S	Mastorus	4
P	Mastorus	4
SP	Mastorus	5

syscolumns		
columns	tbname	coltype
SNR	S	Char (4)
sname	S	Char (20)
scode	S	Smallint
city	S	Char (13)
pur	P	Char (4)
pname	P	Char (20)

Example:

1. "List the name and creator of all tables known to the system"  
`SELECT name, creator FROM systables`
2. "List all columns in S as well as their associated data types."  
`SELECT columns, coltype FROM syscolumns WHERE tbname='S'`
3. "List all tables that contain a column called sname"  
`SELECT tbname FROM syscolumns WHERE columns='sname'`

Note:

- The updating of these tables occurs automatically: create, drop, alter, etc...
- The user should NOT ever update these tables, if no provision.

## INDEX

Purpose: Speed up queries

- An index can be created and maintain from any column or combination of columns in a database.
- Once the index has been created, it can be used for retrieval.
- It may be dropped at will.
- Good alternative to sorting
- But the indexes occupy a lot of space and could be used for something else.

Example:

1. "Create an unique index on SNR column within the S table, called SNR-IN"  
`CREATE UNIQUE INDEX SNR-IN ON S (SNR)`
2. "Create two unique indexes"  
`CREATE UNIQUE INDEX SR-IND ON SP (SNR, PNR)`
3. "Delete the index called SNR-IN"  
`DROP INDEX SNR-IN`

**Aside: Create Index Syntax for SQL Server**  
`CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]  
INDEX index_name ON table (column [,...n])  
[WITH [PAD_INDEX] [,,] FILLFACTOR = fillfactor]  
[,,] IGNORE_DUP_KEY [,,] DROP_EXISTING  
[,,] STATISTICS_NORECOMPUTE ]  
[ON filegroup]`

## VIEWS

- Security considerations  
→ Conceptual schema: some users are not allowed to see the whole conceptual schema
- Personalized collection of queries that watch certain users in tuition (derived from the conceptual schema)
- Any relation that is not part of the conceptual schema, but it is made visible to the user as a virtual relation.

General Form: **Create View** view\_name **as** <query-expression>

Consider the following database:

Tables: Customer, branch, deposit, borrow, account

Example:

1. **CREATE VIEW** all-customers **AS**  
 (SELECT branch-name, customer-name FROM deposit)  
**UNION**  
 (SELECT branch-name, customer-name FROM borrow)
2. Find all customers of Wonderland branch using view all-customers  
**SELECT** customer-name **FROM** all-customers **WHERE** branch-name="Wonderland"

*(Note: Many SQL DBMS impose on a constraint on modifications. Modification (update, insert, delete) is permitted thru a view only if the view is defined in terms of one relation of the actual database - conceptual level)*

3. **CREATE VIEW** loan-info **AS**  
**SELECT** branch-name, loan-number, customer-name **FROM** borrow  
**INSERT INTO** loan-info **VALUES** ("wonderland", 3, CB)

## ASSERTION

An assertion is a predicate expressing a condition that we wish a database to satisfy

- functional dependency
- Multivalued dependency
- Domain constraints
- Referential integrity
- Relational integrity
- Check in SQL

General Form: **ASSERT** <name> **ON** <relation-name>: predicate

Example: **ASSERT** balance\_constraint **ON** deposit: balance >= 0

## TRIGGER

Is a statement that is executed automatically by the system as the side effect of a modification to the database.

To design a trigger we must:

1. Specify the conditions under which the T is executed.
2. Specify the actions to be taken when the T is executed

Example: An account overdraft trigger (system R):

```

DEFINE TRIGGER overdraft ON UPDATE OF deposit T
(IF new T.balance < 0 THEN
  (
    INSERT INTO borrow VALUES (T.branch-name, T.account-number, T.customer-name,
                               new T.balance)
    UPDATE deposit S
      SET S.balance = 0
    WHERE S.account-number = T.account-number
  )
)
    
```

**Security in SQL**

- grant and revoke
- prevention for unauthorized access of database

**Examples:**

1. "User CB must be able to retrieve data from the customer table"  
SQL: grant select on customer to CB.
2. Users BB & CB must be able to add new customer  
SQL: grant insert on customer to BB, CB
3. User CB is to longer allowed to retrieve data from customer table  
SQL: revoke select on customer from CB

**Commit & rollback**

**Commit**

- the commit indicate a successful completion of a transaction
- updates are permanent
- locks are released
- cursors are closed

**Rollback**

- signals unsuccessful termination of a transaction
- updates are being rollback; according to log (reverted)

QUEL, QBE (Query of Example), Internals

**QBE (Query by Example)**

- Early 70s
- Now part of users Query Management Facility (QMF) DMF
- Skeleton tables
- Fill them in
- Relational Calculus

Examples on the banking system:

1. Find all customers who have an account at the Wonderland branch

Deposit	Branch Name	Account Number	Customer Name	Balance
	Wonderland		P. _x	

P = Print  
 \_ = All Match  
 x = domain variable

P.  
 P.ALL ← suppress duplicates

**2. Display the entire deposit relation**

Deposit	...	...	...	...
P.	...	...	...	...

**3. Find the account number of all accounts with balance of more than \$1500**

Deposit	Branch Name	Account Number	Customer Name	Balance
		P.		> 1500

**4. Find the names of all branches that are not located in Toronto**

Branch	Branch Name	Assets	City	...
	P.		<> Toronto	

**5. Find all customers who have an account at both Wonderland branch and Moon Branch**

Deposit	Branch Name	Account Number	Customer Name	Balance
	Wonderland		P._x	
	Moon		P._y	

**6. On several relation & relationships**

Find the **name** and **city** of all customers who have a loan from wonderland branch

Borrow	Branch Name	Loan #	Customer Name	Amount
	Wonderland		_x	

Customer	Customer Name	Customer Street	Customer City	Customer Postal
	P._x			

**7. Condition Box**

Find all customers not named Mastores who have an account at two different branches

Condition Box  
 x:= Mastores

**8. Find all account numbers with balance between 1300 and 1500**

Deposit	Branch Name	Account Number	Customer Name	Balance
		P.		_x

Condition Box
_x >= 1300
_x <= 1500

**9. Find all account numbers with a balance between 1300 and 2000 but not 1500.**

Deposit	Branch Name	Account Number	Customer Name	Balance
		P.		_x

Condition Box
_x >= 1300 AND _x <= 2000 AND _x <> 1500

**10. Ordering**

Branch Name	...	Customer Name	...	Balance
		P.AO(1)		P.DO(2)

- Ascending (1)
- Descending (2)

Do (1) first then sort result with column #2 afterwards (based on results of (1) – grouping)

**11. Modifying**

D – Delete, I – Insert, U – Update

“Delete all of CB’s Account”

Deposit	Branch Name	Account Number	Customer Name	Balance
D.			CB	

**12. Delete the branch city value of the branch whose name is Wonderland.**

Branch	Branch Name	Assets	Branch City
			D.

**13. Delete all loan number between 1300-1500**

Borrow	Branch Name	Loan Number	Customer Name	Amount
D.		_x		

Condition Box
_x = ( >= 1300 AND <= 1500

**14. Delete all account at branch located in Toronto**

Deposit	Account Number	Branch Number	Customer Name	Balance
D.		_x		

branch	Branch Name	Branch City	Assets	Balance
	_x	Toronto		

**15. Insert Customer**

Deposit	Branch Name	Account Number	Customer Name	Balance
I.	Wonderland	1234	RR	1200

**16. Increase all balance with 8%**

Deposit	Branch Name	Account Number	Customer Name	Balance
U.				$\_x * 1.08$ $\_x$

**17. Aggregates**

AVG, MAX, MIN, SUM, CNT, UNQ etc.. (post fixed with ALL to ensure that all has appropriate values are considered.)

“Find the total balance of all account’s belonging to CB”

Deposit	...	Customer Name	Balance
		CB	P.SUM.ALL

**18. Find the total number of customer having an account at Wonderland**

P . CNT . UNQ . ALL

**QUEL**

- started with Ingress → Postgres  
     QUEL      SQL

Most queries are expressed using 3 types of clauses

**Range of**  
**Retrieve**  
**Where**

- + Relational Calculus
- + Each tuple variable is declared in a range of clause:  
     range of t in r  
     to declare it to be a tuple variable restricted to take on values relation r.

1. Find the name of all customers having an account at the Wonderland branch.

```

Range of t is deposit
Retrieve (t.customer_name)
Where t.branchname = "Wonderland"

Range of t is deposit
Retrieve unique (t.customer_name)
    
```

2. Find the name and city of all customers having a loan at the Wonderland branch

```

Range of t is borrow
Range of s is customer
Retrieve unique (t.customername, s.customercity)
Where t.branch_name = "Wonderland" and t.customername=s.customername)
    
```

---

3. Two distinct tuple variables ranging over the same relation.

“Find the name of all customers who have an account at the same branch at which CB has an account.”

Note: compare tuples pertaining to CB with every deposit tuple -- we need two distinct tuple variables ranging are deposit.

```

Range of s is deposit
Range of t is deposit
Retrieve unique (t.customer_name)
Where s.customer_name="CB" and s.branch_name=t.branchname.
    
```

---

4. Find the name of all customers who have both a loan and an account at the Wonderland branch. (#1 & #2)

**Note #1:** (same as #1) Here the query requires only one tuple variables ranging over a relation. Here we may omit the range of statements.

```

Retrieve unique (borrow.customer_name)
Where (deposit.branchname="Wonderland" and borrow.branchname =
      "Wonderland" and deposit.customername = borrow.customername)
    
```

**Note #2:** For the relation  $\rightarrow$  implicitly declared variable. The original academic QUEL does not allow the use of implicitly declared tuple variables  $\rightarrow$  commercial.

---

5. Modifying the database

Deletion:	Range of t is r Delete t Where P	Insertion:	Range of t is r Append to t Where P
-----------	--	------------	---

Update:	Range of t is r Replace t Where P
---------	---

5. Modifying the database (cont...)

**a. Deletion**

1. Delete all of RR account

```
Range of t is deposit
Delete t
Where t.customername = RR
```

2. Delete all account at branches located in Toronto

```
Range of s is branch
Range of t is deposit
Delete t
Where t.branchname=s.branchname AND s.branchcity="Toronto"
```

**b. Insert**

1. Insert EF withdraws \$1200 in account 1234 at the Wonderland branch

```
Append to deposit (branch_name="Wonderland",
Account_number = 1234,
CustomerName = "EF",
Balance = 1200)
```

2. Provide all loan customers in the Wonderland branch with a \$200 savings account (deposit). Here the loan-number will serve as the account number of the new saving account

```
Range of t is borrow
Append to deposit
(t.branch_number, t.loan_number, t. customername, balance=200)
Where t.branchname = "Wonderland"
```

**c. Update**

1. Give 8% interest to all account balances

```
Range of t is deposit
Replace t (balance=1.08*balance)
```

2. Give a 6% interest on account balances over \$10,000 and 4% on all other accounts

```
Range of t is deposit
Replace t (balance=1.06*balance)
Where (balance > 10000)
Replace t(balance=1.04*balance)
Where (balance <=10000)
```

**Note: Order of the two Replace & Where**

## 6. Aggregate Functions

Aggregate (t.A)

Aggregate (t.A Where P)

Aggregate (t.A by S.B<sub>1</sub>, S.B<sub>2</sub>, ..., S.B<sub>n</sub> where P)

Where Aggregate count, sum, avg, max, min

countu, sumu, avgu, → Unique (removes duplicates – distinct)

- a. Find the average account balance for all account at the Wonderland branch

**Range of t is** deposit

**Retrieve** avg (t.balance **where** t.branchname="Wonderland")

- b. Aggregate in the where clause

"Find all account whose balance is higher than the average of all balances at the bank."

**Range of t is** deposit

**Range of s is** deposit

**Retrieve** s.accountnumber

**Where** s.balance > **avg**(t.balance)

- c. "Find all account whose balance is higher than the average of all balances at the Wonderland branch."

**Range of t is** deposit

**Range of s is** deposit

**Retrieve** s.accountnumber

**Where** s.balance > **avg**(t.balance **where** t.branchname="Wonderland")

- d. by

"Find all account whose balance is higher than the average balance at the branch where the account is held."

**Range of u is** deposit

**Range of t is** deposit

**Retrieve** t.accountnumber

**Where** (t.balance > **avg**(u.balance **by** t.branchname **where**  
u.branchname=t.branchname)

---

A small place for your own notes:

6. Aggregate Functions (cont...)

e. Any

Find the name of all customers who have an account at Wonderland branch, but do not have a loan from the Wonderland branch.

- any = to EXISTS in SQL  
which QUEL does not support, but it doesn't lose power.

```

Range of u is borrow
Range of t is deposit
Retrieve unique (t.customername)
Where Any (u.loan_number by t.customername where
           u.branchname="Wonderland" and
           u.customername=t.customername)
    
```

Note: I can do it with count. This applies to SQL too.

Change the query a bit:

“For whom the count of the number of loans from the Wonderland branch is 0.”

```

Range of u is borrow
Range of t is deposit
Retrieve unique (t.customername)
Where Count(u.loan_number by t.customername where
           u.branchname="Wonderland" and
           u.customername=t.customername) = 0
    
```

Note: when using “any” the differences is that there is no count. So the first found tuple, the system stops the searching so it returns 1, if count > 0 otherwise 0.

A spot to fill in the space and to have fun: (All relations are based with Abu... next time it will be someone else)  
Fill in the blanks \_\_\_ and give information regarding the relation

